

**Author: Patrick Franz ([software\\_support@xia.com](mailto:software_support@xia.com))**

**Date: December 18, 2003**

Handel, versions 0.3.3 and greater, now support the ability to set and acquire Single-Channel Analyzer (SCA) data from supported XIA hardware. The aim of the document is to show how to use these new features in Handel. Note that Regions Of Interest (ROIs) are equivalent to the SCA regions discussed below.

## 1. Supported Hardware and Firmware

As of this writing, only the DXP-2X and Saturn/X10P hardware support SCA operations. Beginning with DSP code version 1.08, the standard firmware version supports the definition of up to 16 SCA regions. In the standard firmware, the number of events in each SCA is determined at the end of the run and is stored in a special SCA data buffer. In addition, several special timing versions of the DXP DSP code support SCA processing; in all cases, the SCA limits are defined in the same manner (described below). The output data produced by the timing versions are defined in a separate document.

## 2. Setting the SCAs

The first step is to define the number of SCAs for each channel using `xiaSetAcquisitionValues()`:

```
int status;

double nSCAs = 4.0;

status = xiaSetAcquisitionValues(0, "number_of_scas", (void *)&nSCAs);

if (status != XIA_SUCCESS) {
    /* Error */
}
```

The previous example sets the number of SCAs for detChan 0 to "4". If you are running a multichannel system, e.g. DXP-2X, and you would like to use the same number of SCAs for each channel you may reduce the number of calls to Handel by substituting the following code:

```
status = xiaSetAcquisitionValues(-1, "number_of_scas", (void *)&nSCA);
```

The "-1" indicates that Handel will set the number of SCAs on all channels to "nSCA".

Once the number of SCAs has been set you may proceed to set the limits for each SCA:

```
int status;

double scaLo = 500.0;
double scaHi = 700.0
```

# SCA Support In Handel

---

```
status = xiaSetAcquisitionValues(0, "sca0_lo", (void *)&scaLo);

if (status != XIA_SUCCESS) {
    /* Error */
}

status = xiaSetAcquisitionValues(0, "sca0_hi", (void *)&scaHi);

if (status != XIA_SUCCESS) {
    /* Error */
}
```

As with the number of SCAs, if each channel is going to be using the same limits a *detChan* of “-1” should be used. The individual SCA limits are set using `xiaSetAcquisitionValues()` with the *name* parameter set to “sca[n]\_[lo|hi]”, where *n* is the SCA to set the limit for (starting from 0) and *lo/hi* indicates if the low or high limit is being set. For example, to set the lower limit for SCA 3, the string “sca3\_lo” should be passed to `xiaSetAcquisitionValues()`. See Section (5) for some suggestions on how to reduce the number of calls to `xiaSetAcquisitionValues()` when setting the limits on SCAs.

### 3. Reading the SCA Buffer

Once all of the SCA limits are defined, the next step is to acquire some data and read out the SCAs. In the standard firmware, the SCA totals are stored in the SCA data buffer as two 16-bit words per SCA. Handel provides the routine `xiaGetRunData()` to read out the SCA buffer. `xiaGetRunData()` takes 3 arguments: the *detChan*, the *name* of the data type to read out and the *data* buffer to read the data into. The data buffer needs to have its memory allocated prior to calling `xiaGetRunData()`. However, if the number of SCAs is known at compile time it is sufficient to declare the SCA data buffer as a fixed-size array (see Section (4) for an example of this technique).

### 4. Complete Example

The following is a sample program<sup>1</sup> that illustrates how to set the SCAs for a system with a Saturn module and how to read out the SCAs after a short run:

```
int status;

unsigned short i;

double nSCAs = 2.0;

double scaLowLimits[] = {10.0, 500.0};
double scaHighLimits[] = {20.0, 700.0};

unsigned long SCAs[2];

char scaStr[80];

/* Do initial system configuration and startup */
```

---

<sup>1</sup> Please see the package `handel_scas_src-031029.zip` for the complete source code.

# SCA Support In Handel

---

```
/* Configure SCAs */
status = xiaSetAcquisitionValues(0, "number_of_scas", (void *)&nSCAs);

if (status != XIA_SUCCESS) {
    /* Error */
}

for (i = 0; i < (unsigned short)nSCAs; i++) {

    sprintf(scaStr, "sca%u_lo", i);
    status = xiaSetAcquisitionValues(0, scaStr, (void *)&(scaLowLimits[i]));

    if (status != XIA_SUCCESS) {
        /* Error */
    }

    sprintf(scaStr, "sca%u_hi", i);
    status = xiaSetAcquisitionValues(0, scaStr, (void *)&(scaHighLimits[i]));

    if (status != XIA_SUCCESS) {
        /* Error */
    }
}

/* Start the run */
status = xiaStartRun(0, 0);

if (status != XIA_SUCCESS) {
    /* Error */
}

/* Wait */

/* Stop the run */
status = xiaStopRun(0);

if (status != XIA_SUCCESS) {
    /* Error */
}

/* Read out the SCAs */
status = xiaGetRunData(0, "sca", (void *)SCAs);

if (status != XIA_SUCCESS) {
    /* Error */
}

/* Display SCA values */
for (i = 0; i < (unsigned short)nSCAs; i++) {
    printf("SCA %u = %lu\n", i, SCAs[i]);
}
}
```

## 5. Notes

- Currently, Handel does not preserve the SCA limit data when the “number\_of\_scas” is changed dynamically. For instance, if you create 3 SCAs and then change “number\_of\_scas” to 4 later in the session, the original limits on SCAs 0...2 will be lost.

### **6. Support**

All support requests should be directed to: [software\\_support@xia.com](mailto:software_support@xia.com)