Handel Programmer's Guide - Saturn

Contents

Intended Audience

This document is intended for users of the XIA Saturn hardware who would like to interface to it using the Handel driver library. Users of the Handel driver library should be reasonably familiar with the C programming language and this document assumes the same.

Conventions Used In This Document

- `Fixed width style` is used to indicate source code.
- `CHECK_ERROR` is a placeholder for user-defined error handling.

Preliminary Details

Header Files

Before introducing the details of programming with the Handel API, it is important to discuss the relevant header files and other external details related to Handel. All code that wishes to call a routine in Handel needs to include the file *handel.h* as a header. Furthermore, to gain access to the logging constants used to define the logging level, the file *md_generic.h* should also be included. Finally, *handel_error.h* contains all of the status codes returned by Handel, *handel_constants.h* contains some of the constants used with various Handel routines and *xia_common.h* defines some custom types and macros that are also useful when developing applications with Handel.

Error Codes

A good programming practice with Handel is to compare the returned status value from a Handel function call with XIA_SUCCESS and then deal with any returned errors before proceeding. All Handel routines (except for some of the debugging routines) return an integer value indicating success or failure. While not discussed in great detail in this document, Handel does provide a comprehensive logging and error reporting mechanism that allows an error to be traced back to a specific line of code in Handel.

.INI Files

The final piece of information external to the actual Handel source code is the initialization file. The simplest configuration method is to use the supplied Saturn initialization file (saturn.ini) and to modify it for your configuration.

Customization

The .ini file may need to be customized for the specific details of your setup. The typical customizations that are required are to the detector configuration, the firmware file path and the hardware device configuration.

Detector Configuration

The detector configuration parameters most likely need to be modified to match your experimental setup. Each of the detector configuration parameters is explained in detail below:

- **number_of_channels**: The physical number of preamplifers on > the detector. This value may be different then the total number of > channels supplied by XIA hardware.
- **type**: The preamplifier type. The current supported values are > reset and rc_feedback.
- **type_value**: For reset preamplifiers, this is the reset > delay time. For rc_feedback preamplifiers, this is the RC > decay time.
- **channel{n}_polarity**: The preamplifier polarity for detector > channel n. The accepted values are +, -, pos, positive, neg > and negative.
- **channel{n}_gain**: The preamplifier gain for detector channel n > in mV/keV. See Section 3.6 in the manual “Digital X-ray Processor > User’s Manual: Model DXP Saturn”[1] for a detailed description of > how to measure the preamplifier gain.

[1]The Saturn User’s Manual and other documents referenced in this guide are available in the docs directory of your Saturn Handel source distribution or ProSpect installation directory.

Firmware Configuration

XIA distributes firmware for the Saturn in self-contained files called “FDD” files. Each FDD file contains all of the firmware necessary to run the Saturn in the configuration. Note that there may be multiple FDDs that each support a different Saturn application. Please consult the latest Saturn software release or contact XIA to see what FDDs are available.

To configure Handel to use a specific FDD file, the following parameters can be set:

- **filename**: A valid path to the desired FDD file.
- **fdd_tmp_path**: A valid > path to a directory that Handel can use to store temporary files > it generates while using the FDD file. If you select a directory > for which your user account does not have the proper read/write > permissions, Handel routines that attempt to use the directory > will return an error code. If you do not set this parameter, > Handel uses the operating system default temporary directory.

Module Configuration

The module configuration settings allow you to connect the proper firmware and detector configurations to your hardware. Below are descriptions of the common parameters for the module configuration:

- **module_type**: Always set to dxpx10p for the Saturn[2].
- **number_of_channels**: Always set to 1 for the Saturn. If you > have multiple Saturn modules attached to your computer, you will > have multiple module configurations, not a single configuration > with multiple channels.
- **interface**: XIA has released three versions of the Saturn, each > with a different communication interface: EPP (epp), USB1.1 (usb) > and USB2 (usb2). The additional configuration parameters for each > interface are described below.
- **epp_address** (epp): The > EPP address for the Saturn. The address is usually 0x378, but can > be 0x278 on some computers.
- **daisy_chain_id** (epp): The EPP daisy chain id for the Saturn. > Set this parameter if you have more than one device attached to > your parallel port.

[2]The Saturn is referred to as a dxpx10p or x10p device for legacy reasons. We may support saturn as an official name in the future.

- **device_number** (usb, usb2): The USB/USB2 device number for > the Saturn. On Windows, this value can be found by locating the > Saturn in the Device Manager and opening the properties dialog.
- **channel{n}_alias**: Also commonly referred to in Handel as > the "detChan". Every Saturn channel must be given a > unique detChan. The detChan is used in most Handel function calls > to reference the device.
- **channel{n}_detector**: Binds a module channel to a detector > and preamplifier. The format of this string is "{detector > alias}:{detector preamplifier channel}". For instance, to bind the > Saturn to the first channel in your detector, you would write: > > channel0_detector = "detector1:0" > > assuming that you had a detector with the alias "detector1" defined in > your .ini file.
- **channel{n}_gain**: This should always be set to 1.0.
- **firmware_set_all**: The alias for a firmware definition that > contains the FDD file you want to use with this module.

Example Code

Included with this document is a file called hqsg-saturn.c that is meant to illustrate all of the lessons presented in this tutorial. hqsg-saturn.c is sample code that initializes Handel, configures the Saturn hardware, defines some SCA regions, starts a run, stops a run and reads out the MCA spectrum and SCA counts. A sample .ini file for a USB2 Saturn is included with the source code. The most recent FDD file is available in the latest Handel Saturn release or the latest release of ProSpect.

Initializing Handel

The first step in any program that uses Handel is to initialize the software library. Handel provides two routines to achieve this goal: xiaInit() and xiaInitHandel().[3] The difference between these two initialization methods is that the former requires the name of an initialization file. In fact, xiaInit() is nothing more then a wrapper around the following two functions: xiaInitHandel() and xiaLoadSystem().

```
/*
 * Example1: Emulating xiaInit() using
 * xiaInitHandel() and xiaLoadSystem().
 */
```

[3]Complete descriptions of all the routines discussed in this manual are in the *Handel API*, available on the XIA website: http://www.xia.com/DXP_Software.html

```
int status;

status = xiaInitHandel();
CHECK_ERROR(status);

status = xiaLoadSystem("handel_ini", "saturn.ini");
CHECK_ERROR(status);
```

The above example has the exact same behavior as

```
int status;

status = xiaInit("saturn.ini");
CHECK_ERROR(status);
```

Calling xiaInit() is the preferred method for initializing the library.

Starting The System

Once the initialization task has been completed, the next step is to start the system. Starting the system performs several operations including validating the hardware information supplied in the initialization file, testing the specified communication interface (EPP, USB or USB2 for the Saturn) and downloading the specified firmware to the hardware. Calling xiaStartSystem() is straightforward:

```
status = xiaStartSystem();
CHECK_ERROR(status);
```

Once xiaStartSystem() has been called successfully, the system is ready to perform the standard DAQ operations such as starting a run, stopping a run and reading out the MCA. If a call is made to a routine like xiaLoadSystem() or xiaInit() after xiaStartSystem() is called, then xiaStartSystem() needs to be called again to account for the data modified by loading a new .ini file.

Configuring the Saturn for Data Acquisition

Setting Acquisition Values

By default, the hardware starts up with all of its acquisition values in a nominal state. For most systems, the default values will be sufficient to observe some results from the hardware. However, in order to optimize the hardware for better results, Handel provides access to several “acquisition values” that represent various controls over the hardware. A partial list of the critical acquisition values is below[4]:

- peaking_time
- trigger_threshold
- calibration_energy

In this example, the following operating conditions will be assumed: A peaking time of 16 µs, 1000 eV threshold and a calibration energy of 5900 eV (x-rays from an Fe-55 source).

The routine used to control the acquisition values is called xiaSetAcquisitionValues() and it takes three arguments: a detChan, the name of the acquisition value to set and the value to set the acquisition value to. The acquisition value argument is prototyped as a pointer to a void so that the single routine name may support values of different types. Other routines in Handel use this technique more fully, but currently all acquisition values are doubles. Call the routine with this format to set the calibration energy to 5900 eV:

```
int status;
double calib = 5900.0;

status = xiaSetAcquisitionValues(0, "calibration_energy", (void *)&calib);
CHECK_ERROR(status);
```

In other routines, some of the values are integers while others are unsigned long arrays. Using a pointer to a void, all of these types can be accommodated in a single routine.

The following code illustrates how to set the acquisition values listed above:

```
int status;
double pt = 16.0; /* microseconds */
double thresh = 1000.0; /* eV */
double calib = 5900.0; /* eV */

status = xiaSetAcquisitionValues(0, "peaking_time", (void *)&pt);
CHECK_ERROR(status);

status = xiaSetAcquisitionValues(0, "trigger_threshold", (void *)&thresh);
CHECK_ERROR(status);

status = xiaSetAcquisitionValues(0, "calibration_energy", (void *)&calib);
CHECK_ERROR(status);
```

[4]A complete list of the acquisition values for the Saturn can be found at the end of this application note.

Defining SCAs

The standard Saturn firmware includes 16 “software” SCAs. They are called “software” SCAs because they are calculated by the firmware when the run is over and, as such, my not be read out from the board until the run is over[5]. Handel provides an API for setting the SCA limits

Controlling The MCA

At this stage, the board is configured and ready to begin data acquisition tasks. For this example, the tasks we are interested in are starting a run, stopping a run and reading out the MCA spectrum data.

Starting/Stopping a Run

The Handel interface to starting and stopping the run consists of two simple routines: xiaStartRun() and xiaStopRun(). Both routines require a detector channel number (like xiaSetAcquisitionValues()) as their first argument, while xiaStartRun() also requires an unsigned short that determines if the MCA is to be cleared when the run is started. To start a run with the MCA cleared, run for 5 seconds and then stop the run, the following code may be used:

```
int status;

status = xiaStartRun(0, 0);
CHECK_ERROR(status);

/* If not on Windows, use the appropriate system routine.
 */
Sleep((DWORD)5000);

status = xiaStopRun(0);
CHECK_ERROR(status);
```

Reading out the MCA Spectrum

The final step in the example program is to read out the collected MCA spectrum. There are two methods in which this can be done: the first is to statically allocate memory for the spectrum array and the second is to dynamically allocate memory for the spectrum at run-time. The latter method is covered in the sample source code included with this document (hqsg-saturn.c).

[5]Please contact XIA if your application requires more advanced SCA controls.

The Saturn hardware, in most cases, has a maximum MCA spectrum length of 8k bins (or 8192), so to safely readout the spectrum without dynamically allocating any memory, an array of length 8192 needs to be statically allocated at compile time:

```
int status;
unsigned long mca[8192];

status = xiaGetRunData(0, "mca", (void *)mca);
CHECK_ERROR(status);
```

At this point, the spectrum may be processed as required.

Where To Go Next

The information presented above should serve as a basic introduction to operating the Saturn hardware using Handel. Handel, of course, has many more features that were not discussed in this document. The next step to learn more about Handel is to explore the *Handel API* and become more familiar with what Handel has to offer.

If you have a question or suggestion regarding this document or the included sample code, please contact XIA at software_support@xia.com.

Appendix A – Acquisition Values List

This section lists all of the acquisition values for the Saturn. All of these values are stored as type double by Handel.

peaking_time Peaking time of the energy filter, specified in microseconds. Peaking time is roughly equal to twice the shaping time, which is commonly used in analog systems.

trigger_threshold Trigger filter threshold, specified in eV. The trigger threshold is sometimes referred to as the “threshold”.

energy_threshold Energy filter threshold, specified in eV. The energy threshold is sometimes referred to as the “slow threshold”.

adc_percent_rule Percent of ADC used for a single x-ray step with energy equal to the calibration energy.

calibration_energy Expected energy of the calibration peak, specified in eV.

mca_bin_width Width of an individual bin in the MCA, specified in eV.

number_mca_channels The number of bins in the MCA spectrum, specified in bins.

mca_low_limit Energy cut-off for the lowest MCA bin, specified in eV.

gap_time The gap time of the energy filter, specified in microseconds. This value should be treated as a minimum gap time, since it is unlikely that Handel will set the exact value when the DECIMATION is greater then 0. To get the actual gap time that is set on the hardware, read out the acquisition value `actual_gap_time`.

actual_gap_time A read-only value that returns the current gap time on the hardware. This value takes into account the current decimation, compared to `gap_time` which does not. See `gap_time` for more information.

trigger_peaking_time The peaking time of the trigger filter, specified in microseconds. This value is used to calculate FASTLEN. See the Mercury User Manual for more information on filtering.

trigger_gap_time The gap time of the trigger filter, specified in microseconds. This value is used to calculate FASTGAP.

preset_type Set the preset run type. See :file:`handel_constants.h` for the constants that can be used.

preset_value When a preset run type other then :const:`XIA_PRESET_NONE` is set, this value is either the number of counts or a time (specified in seconds).

peakint_offset_ptrr{n} The constant from the PEAKINT recipe. (See Filter Parameters for more details.) If an FDD file is being used then the "_ptrr{n}" part of the name can be omitted.

peaksam_offer_ptrr{n} The constant from the PEAKSAM recipe. (See Appendix C for more details.) If an FDD file is being used then the "_ptrr{n}" part of the name can be omitted.

baseline_filter_length Sets the baseline filter length in terms of the number of samples. The default value is 128.0 samples.

reset_delay The amount of time that the processor should wait after the detector resets before processing the input signal, specified in microseconds. This value mirrors, and is synchronized with, the value set in the .ini file under `\[detector definitions\]`.

decay_time The decay time of the RC circuit in the preamplifier. This is the same setting used during configuration of RC feedback preamplifiers.

preamp_gain Preamplifier gain, specified in mV/keV. This value mirrors, and is synchronized with, the value set in the .ini file under `\[detector definitions\]` and via xiaAddDetectorItem() and xiaModifyDetectorItem().

detector_polarity The input signal polarity, specified as positive (1.0) or negative (0.0). This value mirrors, and is synchronized with, the value set in the .ini file under `\[detector definitions\]` and via xiaAddDetectorItem() and xiaModifyDetectorItem(). In the .ini file the values are "+", "-", "pos" or "neg".

enable_gate Enables use of the gate connection on the back of the Saturn hardware. Consult the *Digital X-ray Processor User's Manual: Model DXP Saturn* for further details.

enable_baseline_cut Enables the baseline cut, which excludes baseline samples that are outside the cut range, expressed in % of the maximum peak value of the baseline histogram. See also `baseline_cut`. The default setting is “enabled” (1.0). To disable the baseline cut, set this value to 0.0.

baseline_cut Sets the fraction of the peak value of the baseline histogram at which baseline samples are rejected. This value is specified in %. The default setting is 5%. Requires that the baseline cut be enabled; see enable_baseline_cut for more information.

Appendix B – Filter Parameters List

Each product potentially interprets the filter information stored in the Firmware structure differently. Handel stores filter data but makes no assumption about what the data means. The burden is on the individual user/programmer to verify that the information is entered correctly.

The index is the order in which it should be added as an item to a Firmware alias using xiaAddFirmwareItem.

Name	**Index**	**Description**
peakint_offset	0	The constant in the PEAKINT recipe: PEAKINT = SLOWLEN + SLOWGAP + peakint_offset
peaksam_offset	1	The constant in the PEAKSAM recipe: PEAKSAM = PEAKINT − peaksam_offset

Appendix C – Run Data List

This section lists the run data supported by xiaGetRunData for Saturn applications. In some cases, there are also special types associated with various firmware configurations.

Name	Type	Firmware	Description
mca_length	unsigned long	Standard	The length of the MCA spectrum in unsigned long words.
mca	unsigned long *	Standard	An array containing the MCA spectrum. The user is expected to pass in an array of size “mca_length” to Handel.
livetime	double	Standard	The total time that the processor was able to acquire new events in seconds.
runtime	double	Standard	The total time that the processor was taking data in seconds. Some XIA manuals refer to this as the “realtime”.
input_count_rate	double	Standard	This is the total number of triggers divided by the runtime in counts per second.

output_count_rate	double	Standard	This is the total number of events in the run divided by the runtime in counts per second.
events_in_run	unsigned long	Standard	The total number of events in a run that are written into the MCA spectrum.
triggers	unsigned long	Standard	The total number of triggers obtained in a run. This quantity includes pile-up inspection.
baseline_length	unsigned long	Standard	The length of the baseline histogram.
baseline	unsigned long *	Standard	An array containing the baseline histogram. The user is expected to pass in an array of size “baseline_length” to Handel.
run_active	unsigned long	Standard	Returns run active status of a channel. The following constants, defined in handel.h, are returned: - XIA_RUN_HARDWARE – The hardware thinks that a run is active. - XIA_RUN_HANDEL – Handel thinks that a run is active. - XIA_RUN_CT – Handel thinks that a control task run is active.

Appendix D – Special Run Types List

This section lists the special runs supported by xiaDoSpecialRun for Saturn applications.

Each special run accepts a different set of parameters via the info array. The first element of the info array is the same for all of the special runs and should be set to the number of times the special run will execute. (For most special runs, this should be set to 1.) The Read Data column indicates if the appropriate xiaGetSpecialRunData() routine must be called in order to stop execution of the special run.

Name	Read Data?	Type	Info
adc_trace	No	double	info[1]: Amount of time to wait between ADC samples in nanose
baseline_history	Yes	int	N/A
open_input_relay	No	int	N/A
close_input_relay	No	int	N/A

Below is the table of special run data that can be read:

Name	Type	Description
adc_trace_length	unsigned long	The length of the ADC trace to be read from the processor in u

adc_trace	unsigned long *	An array containing the ADC trace data. The user is expected t
baseline_history_length	unsigned long	The length of the baseline history buffer to be read from the pro
baseline_history	long *	An array containing the baseline history data. The user is expec

Legal

Copyright 2005-2018 XIA LLC

All rights reserved

All trademarks and brands are property of their respective owners.

Licenses

Handel

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of XIA LLC nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS

SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Documentation

Redistribution and use in source (Markdown) and 'compiled' forms (HTML, PDF, LaTeX and so forth) with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code (Markdown) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.
- Redistributions in compiled form (transformed to other DTDs, converted to PDF, PostScript, HTML, LaTeX, RTF and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS DOCUMENTATION IS PROVIDED BY XIA LLC "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL XIA LLC BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Disclaimer

Information furnished by XIA LLC is believed to be accurate and reliable. However, XIA assumes no responsibility for its use, nor any infringements of patents or other rights of third parties, which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of XIA. XIA reserves the right to change specifications at any time without notice. Patents have been applied for to cover various aspects of the design of the DXP Digital X-ray Processor.

Patents

Patent Notice