



XAPP1182 (v1.0)  
November 18, 2013

# System Monitoring using the Zynq-7000 AP SoC Processing System with the XADC AXI Interface

Authors: Mrinal J. Sarmah and Radhey S. Pandey

## Summary

This application note describes how a Xilinx analog-to-digital converter (XADC) can be used for system monitoring applications. The XADC Wizard IP offers an AXI4-Lite<sup>(1)</sup> interface which is connected to a Zynq®-7000 All Programmable SoC (AP SoC) processing system AXI general purpose (GP) port to get system control information from the XADC. The XADC block provides dedicated alarm output signals that trigger based on preset events. This application note has a Linux application running on an ARM® Cortex™-A9 CPU on a Zynq-7000 All Programmable SoC device that controls the alarm threshold of the XADC and monitors the alarm output.

## Introduction

The Zynq-7000 family is based on the Xilinx All Programmable SoC architecture. These products integrate a feature-rich dual-core ARM Cortex-A9 based processing system (PS) and 28 nm Xilinx programmable logic (PL) in a single device. The ARM Cortex-A9 CPUs are the heart of the PS and include on-chip memory, external memory interfaces, and a rich set of peripheral connectivity interfaces.

The Zynq-7000 AP SoC PS can establish connectivity to the XADC, an integrated 12-bit, 17-channel, 1 MSPS analog-to-digital converter using the AXI interface when the XADC is instantiated in the PL. The XADC is an embedded block offered in all Zynq-7000 All Programmable SoC devices.

The LogiCORE™ XADC Wizard IP provides an AXI4-Lite compatible interface and an optional AXI4-Stream interface. The AXI4-Lite interface can be used to configure the XADC, and the AXI4-Stream interface can be used for data communication. The AXI4-Stream interface provides an option to interface the XADC data interface to other signal processing IP. This application note demonstrates the use of the AXI4-Lite interface for system monitoring applications using the XADC.

The XADC has a power supply and temperature sensor that can be used for system monitoring. Each of these sensors has configurable minimum and maximum threshold limits. When the measured physical parameter (voltage or temperature) crosses the threshold condition, an alarm signal is asserted. For more details on the on-chip sensors, see *7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide (UG480)* [Ref 1].

Xilinx provides an Industrial Input/Output (IIO) framework-based Linux driver which acts as a device driver for the XADC applications that use the AXI interface. The driver can configure the XADC for various operating modes, collect data from XADC, and make data available in the user space layer.

This application note shows how the IIO based Linux driver can be used to configure the XADC and provides a hardware design in the PL that establishes the datapath between the XADC and the PS using the AXI GP port interface. The Cortex-A9 processor is used to configure the XADC for user-specific configuration parameters.

---

1. ARM® Advanced eXtensible Interface 4 (AXI4)

A web server based GUI interface is used to configure the XADC and display the collected samples.

## Hardware Design Overview

The hardware design is intended to provide connectivity between the XADC block in the PL and the Processing System-7 embedded block. The XADC embedded block has a DRP interface for reading and writing data into the XADC DRP addressable register map. The XADC wizard converts the AXI4-Lite transactions into the DRP address map. The XADC wizard instantiates the AXI interrupt controller that converts the alarm output from the XADC embedded block into interrupt events for the processor.

Figure 1 shows the hardware block diagram.

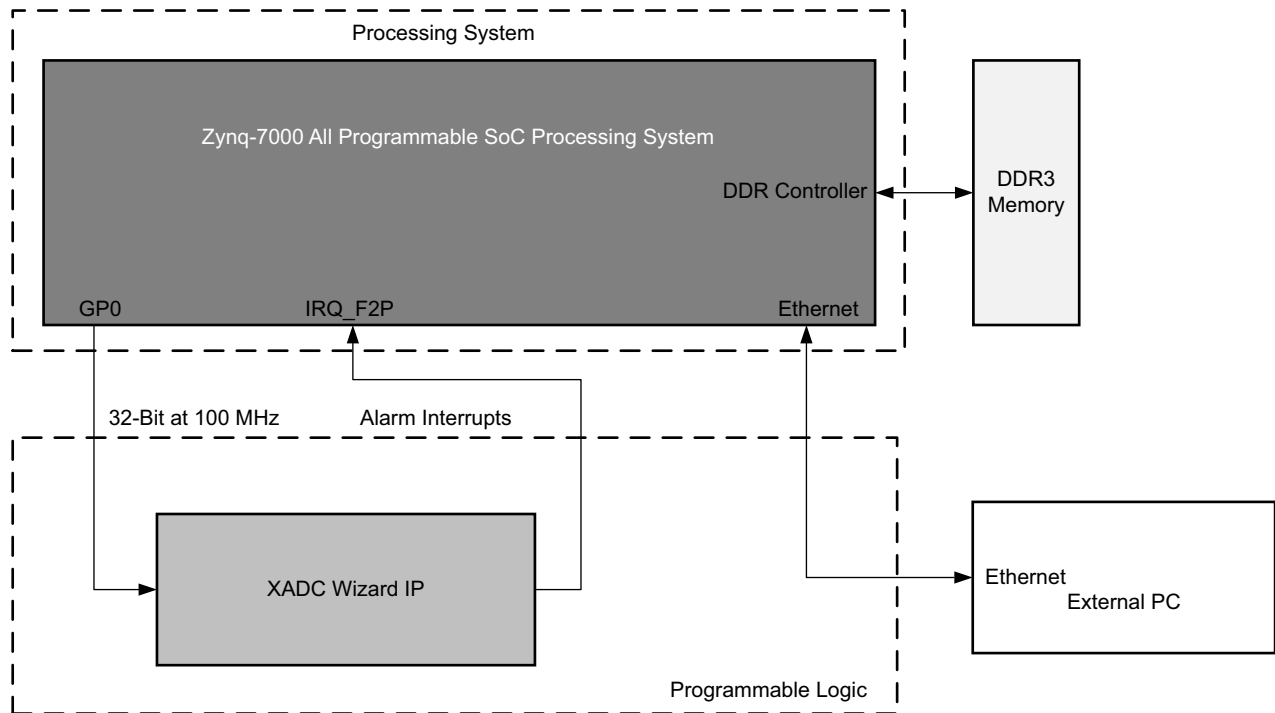


Figure 1: Hardware Block Diagram

The following sequence describes the data flow:

1. The Linux driver for the XADC initializes the XADC Wizard IP.
2. The Linux driver sets up the alarm registers with user-defined threshold values.
3. The alarm output from the XADC is set when the measured voltage or temperature value crosses the preset alarm threshold.
4. The XADC Wizard IP raises interrupts through the Fabric-to-PS interrupt pin.
5. The processor serves the interrupt and reports to the user interface about the alarm event.
6. The alarm event stays high until the alarm condition is valid.

The interrupt port from XADC wizard is connected to IRQ\_F2P interrupt port 0 with ID 91.

The design uses the Vivado® Design Suite IP Integrator flow for creation of the block design. After the block design is created, a hardware wrapper is generated that instantiates the IP-specific wrapper files.

Table 1 shows the Xilinx IPs used in the hardware design.

Table 1: IPs Used in the Hardware Design

IP Name	Description	Configuration
Processing System-7	Generates a wrapper that instantiates the Processing System-7 embedded block	Configured with ZC702 default preset from the Processing System-7 IP wizard
AXI Interconnect IP	AXI Interconnect IP that converts AXI3 transactions from Processing System-7 IP to AXI4 transactions for XADC Wizard IP	Configured for one master and one slave interface
XADC Wizard IP	XADC Wizard IP instantiates the XADC embedded block and converts AXI4-Lite transactions to DRP transactions required by the embedded block	Configured to support continuous sampling mode
Proc Sys Reset	Applies reset to peripherals and AXI Interconnect IP	Configured to apply reset to AXI Interconnect and XADC Wizard IP

Table 2 shows the address map of the memory mapped peripheral in the PL.

Table 2: Address Map of Memory-Mapped Peripheral in the PL

Peripheral in PL	Address map (Hex)
XADC Wizard IP	43C00000-43C0FFFF

## Software Architecture

The software application used in this application note is based on the Industrial Input/Output (IIO) framework driver, which is part of the Linux `git` tree.

The Linux IIO subsystem is the standard framework used for providing support for devices that fall into ADC/DAC categories.

The subsystem provides the following facilities to the user space:

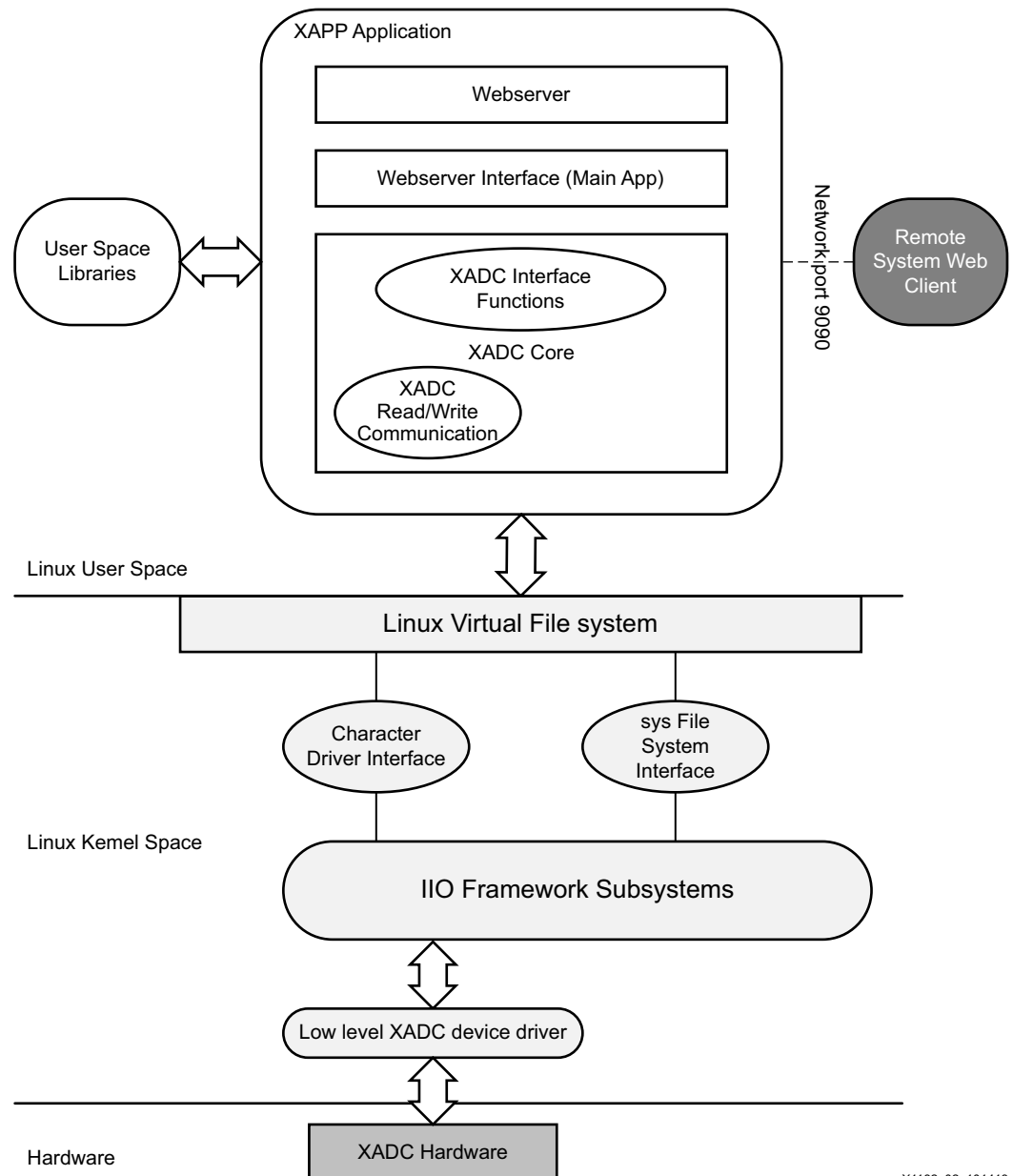
- Sysfs interface for communicating with the devices
- Character driver interface for receiving the event information

The software application used in this application note can be divided into four logical sections:

- XADC core
  - This part of the application is like a library that is solely responsible for communicating with the IIO system for hardware configuration, retrieving sensor values, and event handling.
- Web server
  - The web server listens to the connection request from the remote web clients. Web clients running on a remote computer system can acquire sensor data and event notifications through the web server interface. The server gets sensor data and event notifications through the XADC core.
  - It also acquires threshold values from the web client and passes them over to the XADC core to configure the hardware for events/alarms generation.
- Web interface
  - This section is responsible for stitching the web server to the XADC core. It uses the API exposed by XADC core to get the sensor alarm values and to set the thresholds. It passes these values to and from web server.

- Web client
  - The web client runs on the remote system in a web browser. The client communicates with the web server on http port 9090.
  - The client runs a GUI application you can use as a front end. The GUI displays the captured sensor data along with graphs. It also provides you an interface to program threshold values for the sensor events.

The application normally runs as a daemon process in the background on a Zynq-7000 All Programmable SoC device. [Figure 2](#) shows the top-level view of the application and IIO framework.



X1182\_02\_101413

Figure 2: **Software Block Diagram**

## Application XADC Core APIs

The XADC core section of the application is responsible for all communication with the XADC device through the IIO subsystem.

The implementation for this section is available in the `xadc_core.h` and `xadc_core.c` files.

Other applications can call the APIs provided by this section in the `xadc_core_if.h` file.

The following are descriptions for the enumerations, structures, and APIs declared in the `xadc_core_if.h` file:

### Enumerations

- **XADC\_Parm**—This enumeration defines the available parameters that can be queried for the statistic.
- **XADC\_Alarm**—This enumeration defines the available alarms on the system which can be programmed and queried for status.

### Structures

- **xadc\_callback**—This structure contains the call-back information for the alarm. It has a function pointer and an argument pointer that should be passed for the function.

### Function APIs

- **int xadc\_core\_init(void);**  
This function should be called once in the beginning of the program. It is responsible for finding the XADC device nodes and initializing global variables.  
Argument: N/A  
Return Value: [Integer] 0: For success, -1: Device node not found
- **int xadc\_core\_deinit(void);**  
This function should be called at the end of program. It is responsible for releasing resources.  
Argument: N/A  
Return Value: [Integer] 0: Success
- **void xadc\_update\_stat(void);**  
This function updates the global caches for all statistic parameters. This should be called before the `xadc_get_value()` function, which reads the statistic from the cache.  
Argument: N/A  
Return Value: N/A
- **float xadc\_get\_value(enum XADC\_Param parameter);**  
This function returns the cached statistics value of a given parameter. For voltage, the return value is in millivolts, and for temperature, it is in degrees Celsius.  
Argument: parameter: enumeration value for the parameter whose value is needed  
Return Value: [float] cached statistic of the given parameter [mV/degree C].
- **float xadc\_touch(enum XADC\_Param parameter);**  
This function updates the global cache statistic for the given parameter. It returns the realtime value of the given parameter, unlike `xadc_get_value`.  
Argument: parameter: enumeration value for the parameter whose value is needed.  
Return Value: [float] realtime value of the given parameter[mV/degree C].

- `int xadc_set_threshold(enum XADC_Alarm alarm, float threshold_low, float threshold_high, struct Xadc_callback *callback);`

This function sets the threshold values for the given alarm.

Argument:

- alarm: enumeration value for the alarm for which the threshold is set.
- threshold\_low: Low threshold value for the alarm [mV/degree C].
- threshold\_high: High threshold value for the alarm [mV/degree C]
- callback: callback information for the event on the given alarm. If NULL is passed in this argument, no callback is registered, otherwise, on event occurrence, callback > func(arg) is called.

Return Value: [Integer] 0: Success, Non-zero: Error

- `bool xadc_get_alarm_status(enum XADC_Alarm alarm);`

This function returns the current status of the event for the given alarm, valid only after setting the threshold. It is useful in cases where one does not need `callback`, but just wants to get the status of the event at some stage.

Argument: alarm: enumeration value for the alarm for which event status is needed.

Return Value: [bool] 1 = Event Active, 0 = Event Inactive.

## Linux Driver

The IIO framework (subsystem) is implemented in kernel space. It provides IIO device driver registration for the low-level device drivers. The device drivers have to implement the required functions as specified by the IIO framework and register them as part the framework. These functions get called for all the hardware-specific operations.

This driver replaces the functionality for hwmon based ADC driver. Therefore, to use this subsystem for XADC, hwmon xadc should be disabled in the kernel build.

## Sysfs Interface

If there is an XADC entry in the `.dts` file, the IIO subsystem populates the sysfs interface for the XADC. The sysfs entries can be found in the `/sys/bus/iio/devices/<populated-device>` file.

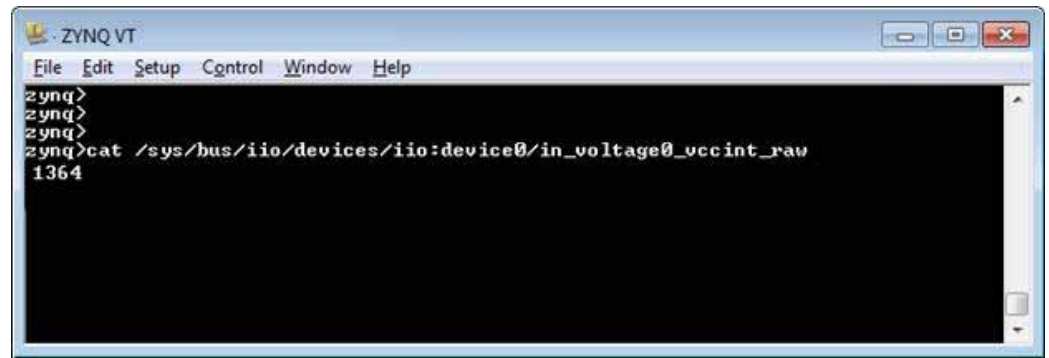
The path `/sys/bus/iio/devices/<populated-device>` contains the file nodes for different values of various parameters.

You can confirm the correct device by reading the `/sys/bus/iio/devices/<populated-device>/name` file (it is read as "xadc" in our case).

For example, to get the sensor raw code value of  $V_{CCINT}$ , read the following file:

```
/sys/bus/iio/devices/<populated-device>/in_voltage0_vccint_raw
```

Figure 3 shows a command line example.



```

ZYNQ VT
File Edit Setup Control Window Help
zynq>
zynq>
zynq>
zynq>cat /sys/bus/iio/devices/iio:device0/in_voltage0_vccint_raw
1364

```

X1182\_03\_101113

Figure 3: Figure 3: Command Line Example

## Command Line Execution

This application note provides a command line interface for monitoring the internal  $V_{CCINT}$ ,  $V_{CCAUX}$ , and  $V_{CCBRAM}$  channels. The commands can be executed in the command line shell of the Linux OS. These commands are installed in the system while the Linux OS boots up.

The commands in [Table 3](#) are available.

Table 3: System Monitoring Commands

Command	Explanation	Example
<code>xadc_get_value_vccint</code>	This command returns the monitored $V_{CCINT}$ value.	Type this command in the console: <b>\$ xadc_get_value_vccint</b> to return the $V_{CCINT}$ value in millivolts.
<code>xadc_get_value_vccaux</code>	This command returns the monitored $V_{CCAUX}$ value.	Type this command in the console: <b>\$ xadc_get_value_vccaux</b> to return the $V_{CCAUX}$ value in millivolts.
<code>xadc_get_value_vccbram</code>	This command returns the monitored $V_{CCBRAM}$ value.	Type this command in the console: <b>\$ xadc_get_value_vccbram</b> to return the $V_{CCBRAM}$ value in millivolts.
<code>xadc_get_value_temp</code>	This command returns the monitored temperature value.	Type this command in the console: <b>\$ xadc_get_value_temp</b> to return the temperature value in degrees Celsius

## Event Notifications

The event notification can be enabled by writing to the files under the `/sys/bus/iio/devices/<populated-device>/events/` directory.

For example, to set the high and low threshold of  $V_{CCINT}$ , write the raw code of the value to the following files respectively.

```

/sys/bus/iio/devices/<populated-device>/events/in_voltage0_vccint_thresh_rising_value
/sys/bus/iio/devices/<populated-device>/events/in_voltage0_vccint_thresh_falling_value

```

And, to enable the  $V_{CCINT}$  event for both high and low, write `1` to following files:

```

/sys/bus/iio/devices/<populated-device>/events/in_voltage0_vccint_thresh_rising_en
/sys/bus/iio/devices/<populated-device>/events/in_voltage0_vccint_thresh_falling_en

```

After setting the threshold, to get an event there is a character driver interface.

To receive an event, follow these steps after setting the threshold:

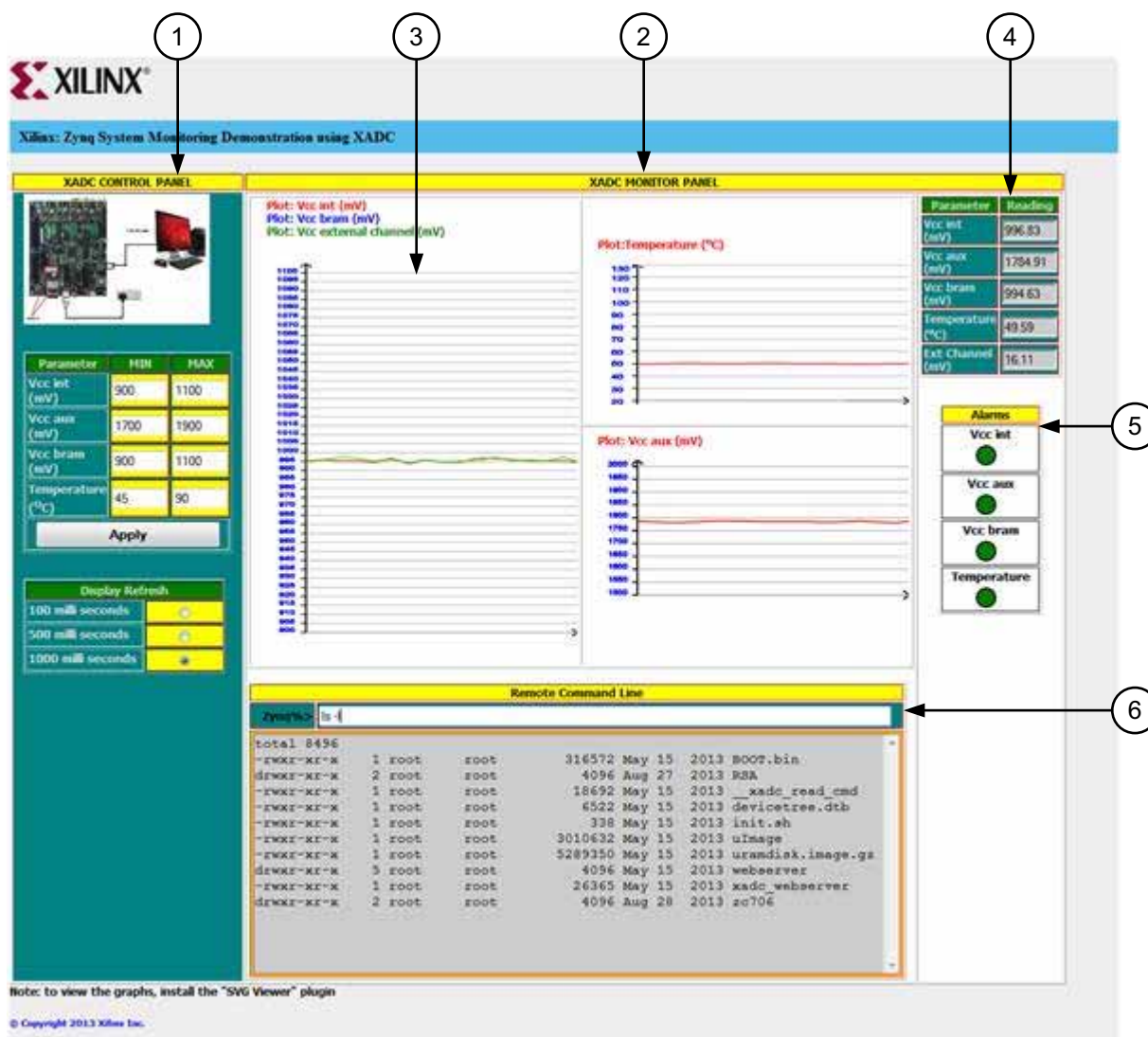
1. Include `<linux/iio/events.h>` for the events and ioctl definitions.  
`#include <linux/iio/events.h>`
  2. Use the `<populate-device>` directory name and open the device file  
`"/dev/<populate-device>".`
  3. Use this `fd` (of step 2) to get the event file descriptor:  
`ioctl(fd, IIO_GET_EVENT_FD_IOCTL, &event_fd)`
  4. Call `read` on this `event_fd`.  
`read(event_fd, &event, sizeof(event));`  
Here "event" is of type "struct iio\_event\_data".  
Decipher this event as described in `events.h` file to locate the exact event.
  5. This `read` is a blocking call, which is released when any event has occurred.
- 

## GUI Interface

The GUI for AXI-XADC is based on Webserver. This is done using HTML, JavaScript, and TCP-HTTP protocol handlers. The GUI screen is shown in [Figure 4](#). The GUI screen is divided into two panels:

- XADC Control Panel
- XADC Monitor Panel





X1182\_04\_101113

Figure 4: Webserver Based GUI Application

Notes relevant to Figure 4:

1. XADC Control Panel
2. XADC Monitor Panel
3. Graphs
4. Readings
5. Alarms
6. Remote Command Line

### XADC Control Panel

This panel is located on the left side of the screen (numbered 1 in Figure 4). Use this panel to modify the minimum and maximum threshold values for  $V_{CCINT}$ ,  $V_{CCBRAM}$ ,  $V_{CCAUX}$ , and temperature. When you click the **Apply** button, these values are programmed in the appropriate Zynq-7000 All Programmable SoC device register. This causes the AXI-XADC Alarm system to get adjusted to the new threshold values you specified.

## XADC Monitor Panel

This panel is shown with number 2 in [Figure 4](#). The web client collects readings from the AXI XADC interface periodically (once every second) and populates the readings on the XADC Monitor Panel.

The XADC Monitor Panel has four parts, showing various readings and graphs.

### Graphs

The graphs are shown with number 3 in [Figure 4](#). There are four linear graphs plotted by the web page.

- $V_{CCINT}$  graph is plotted with a scale of 0–1200 millivolts.
- $V_{CCBRAM}$  graph is plotted with a scale of 0–1200 millivolts.
- $V_{CCAUX}$  graph is plotted with a scale of 0–1000 millivolts.
- External voltage graph is plotted with a scale of 0–1000 millivolts.

Each of these graphs is plotted with readings collected every one second. Also, the graphs maintain a history of the last 10 readings. As time moves, the graphs scroll from right-to-left and plot the latest values on the right side.

### Readings

The various readings are shown with number 4 in [Figure 4](#). WebClient probes the XADC every second and gets the latest readings. These values are also plotted on the graphs.

### Alarms

The alarm indications are shown with number 5 in [Figure 4](#). The Zynq-7000 device's AXI XADC interface can monitor whether or not the current value of a parameter is out of bounds (minimum, maximum threshold values). If a value is out of bounds, a corresponding alarm is raised.

The web page tracks and updates the alarm status for four parameters ( $V_{CCINT}$ ,  $V_{CCAUX}$ ,  $V_{CCBRAM}$ , and temperature). The color of each alarm turns to green (within bounds) or red (out of bounds) in real time.

You can program the MIN or MAX threshold values of a parameter and test the alarms. The alarm turns red as soon as values go beyond the programmed threshold.

### Remote Command Line

The Remote Command Line window is shown with number 6 in [Figure 4](#). This is an extended Linux prompt onto the GUI. This control acts as shell prompt for the Zynq-7000 device. You can treat it as a Zynq-7000 AP SoC shell, and enter any valid Linux command. The text is carried to the Zynq-7000 device and an attempt is made to execute it as a Linux command. After executing the command, the Zynq-7000 device sends the results to the GUI and displays results on the Remote Command Line console.

The command entered here is executed in a temporary process that gets killed after completing the execution. That means the results of a command are not persistent.

For example, if the command executed is `cd /usr`, the shell returns to the home folder after it returns (it does not stay in new changed folder).

## External Channel Measurement

For measuring the voltage level in an external channel, the IIO framework provides an external channel measurement option. You have to enable the external channel, intended to be monitored in the device tree source (`.dts`) file. In the Zynq-7000 All Programmable SoC ZC702 Evaluation Kit, the AMS101 daughter card is shipped and can be inserted into the

XADC header pin. For more details on connecting an external signal source to the AMS101 card, see *AMS101 Evaluation Card User Guide* (UG886) [Ref 2].

## Hardware Requirements

The design can be tested using the ZC702 evaluation platform. To collect external data using the AMS101 evaluator card, you need to follow instructions listed in the *AMS101 Evaluation Card User Guide* (UG886) [Ref 2] to connect the AMS101 card to the ZC702 evaluation board. An external signal can be applied to the auxiliary channel of the AMS101 card and the collected samples can be read through the AXI XADC interface.

## Experimental Results

The maximum external signal bandwidth that can be supported with the AXI4-Lite interface of XADC Wizard IP to AXI GP interface in the Zynq-7000 family has been measured and found to be 2.12 MHz. The measurement is taken in ideal conditions with the CPU running only the XADC application. No other applications are run. The measurement is taken with alarm events enabled in the driver.

The time taken from issuing a read request to the reception of completion data has been experimentally measured. The completion data is read when End Of Conversion (EOC) interrupt is asserted. Table 4 summarizes the latency value under different operating conditions.

Table 4: Latency Values

Command Type	Worst Case Latency <sup>(1)</sup>	Average Latency <sup>(1)</sup>
Memory read	0,72 $\mu$ s	0.4771 $\mu$ s

1. The values are measured with AXI4-Lite clock frequency set to 100 MHz.

## Conclusion

This design provides the platform for using the AXI4-Lite interface of XADC for system monitoring. The design also explores the possibility of using an external auxiliary channel through the AXI4-Lite interface of XADC and characterizes the maximum signal frequency that can be monitored using the interface.

The latency number provided in this application note is the best that can be achieved through this interface. It can vary depending on CPU load conditions.

## Reference Design

The reference design ZIP file can be downloaded from the following URL:

<https://secure.xilinx.com/webreg/clickthrough.do?cid=346167>

Follow the instructions provided in the `readme` file for building hardware and software code. Table 5 shows the reference design checklist.

See the [Zynq AXI XADC App Note Wiki page](#) for rebuilding the design.

Table 5: Reference Design Checklist

Parameter	Description
<b>General</b>	
Developer name	Xilinx
Target devices	Zynq-7000 All Programmable SoC
Source code provided	Yes
Source code format	C
Design uses code and IP from existing Xilinx application note and reference designs, CORE Generator™ technology, or third parties	Yes

Table 5: Reference Design Checklist (Cont'd)

Parameter	Description
<b>Simulation</b>	
Functional simulation performed	No
Timing simulation performed	No
Test bench used for functional and timing simulations	No
Test bench format	Not applicable
Simulator software/version used	Not applicable
SPICE/IBIS simulations	Not applicable
<b>Implementation</b>	
Synthesis software tools/version used	Vivado Design Suite 2013.3
Implementation software tools/versions used	Vivado Design Suite 2013.3
Static timing analysis performed	Vivado Design Suite 2013.3
<b>Hardware Verification</b>	
Hardware verified	Yes
Hardware platform used for verification	ZC702

## Device Tree

Here is the snippet of XADC entry in the Xilinx Linux device tree:

```
xadc@43c00000 {
    compatible = "xlnx,axi-xadc-1.00.a";
    reg = <0x43c00000 0x10000>;
    interrupts = <0 59 4>;
    interrupt-parent = <&gic>;
    clocks = <&ps_clk>;
    xlnx,channels {
        #address-cells = <1>;
        #size-cells = <0>;
        channel@0 {
            reg = <0>;
        };
    };
};
```

For more information on possible XADC entries, please see the Linux kernel documentation:

<Documentation/devicetree/bindings/iio/xilinx-xadc.txt>

## References

The following references are cited or useful with this application note:

1. *7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide* ([UG480](#))
2. *AMS101 Evaluation Card User Guide* ([UG886](#))
3. *Zynq-7000 All Programmable SoC Technical Reference Manual* ([UG585](#))
4. Analog Devices Wiki, Linux Industrial I/O Subsystem, IIO Overview: <https://wiki.analog.com/software/linux/docs/iio/iio>
5. [Zynq AXI XADC App Note Wiki](#)

## Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
11/18/2013	1.0	Initial Xilinx release.

## Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at [www.xilinx.com/legal.htm#tos](http://www.xilinx.com/legal.htm#tos); IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at [www.xilinx.com/legal.htm#tos](http://www.xilinx.com/legal.htm#tos).