



XAPP1172 (v1.0.1) March 18, 2014

Using the Zynq-7000 Processing System (PS) to Xilinx Analog to Digital Converter (XADC) Dedicated Interface to Implement System Monitoring and External Channel Measurements

Authors: Pallav Joshi, Srinivasa Attili and Mrinal J. Sarmah

Summary

The XADC to PS interface is a dedicated interface present in Zynq™-7000 family All Programmable SoCs that enable connectivity between the PS and the XADC block without requiring building of the HW bitstream. The interface can be used for system monitoring applications where CPU needs to monitor the on chip voltage and temperature along with external voltage or current channels.

The XADC provides alarm events which are set based on user pre-configured values. The alarms can be used to trigger system level actions such as shutting down the system in case of over temperature. This application note explores the system monitoring aspects of the dedicated XADC to PS interface along with event programming and monitoring.

Introduction

The Zynq-7000 family is based on the Xilinx® All Programmable SoC (AP SoC) architecture. These products integrate a feature-rich dual-core ARM® Cortex™-A9 MPCore™ based processing system (PS) and Xilinx programmable logic (PL) in a single device. The ARM Cortex-A9 MPCore CPUs are the heart of the PS which also includes on-chip memory, external memory interfaces, and a rich set of I/O peripherals.

The Zynq-7000 PS has a dedicated interface to the XADC, two 12-bit, 1 MSPS analog to digital converters supporting on chip voltage and temperature measurements and 17 external channels. The XADC is a hard block offered in all Zynq-7000 AP SoCs. The XADC to PS interface enables a reliable system monitoring capability in the Zynq-7000 AP SoC, without requiring the need to enable the Programmable Logic. This interface allows monitoring of both internal (voltage sensors and temperature) and external signals such as on-board voltages, currents or other environmental sensors. Alarms and warning signals can be set for conditions such as over-temperature and over/under-voltage conditions, giving users a more complete perspective of the conditions surrounding the Zynq-7000 AP SoC.

The communication between the PS and the XADC happens over a serial interface which is similar to Joint Test Action Group (JTAG) interface. The command from the PS is stored in a command FIFO using the APB interface and then serialized using a parallel-to-serial converter. The serialized data is then sent over the TDO pin (port) of the JTAG interface. The JTAG clock is generated in the Zynq-7000 PS subsystem. The incoming serial data from the XADC is converted to parallel data using a serial-to-parallel converter. The parallel data is stored in a data FIFO and then read out by the PS using the APB interface. For more information on the Zynq-7000 XADC to PS interface, please refer to the *Zynq-7000 All Programmable SoC Technical Reference Manual* ([UG585](#)).

Note that while the XADC to PS interface is in use, the user will not be able to use the external JTAG interface to access the XADC registers.

For more details on XADC please see chapter 30 of [UG585](#).

The focus of this application note is to showcase the system monitoring feature of Zynq-7000 AP SoC devices implemented using dedicated XADC to PS interface.

This application note:

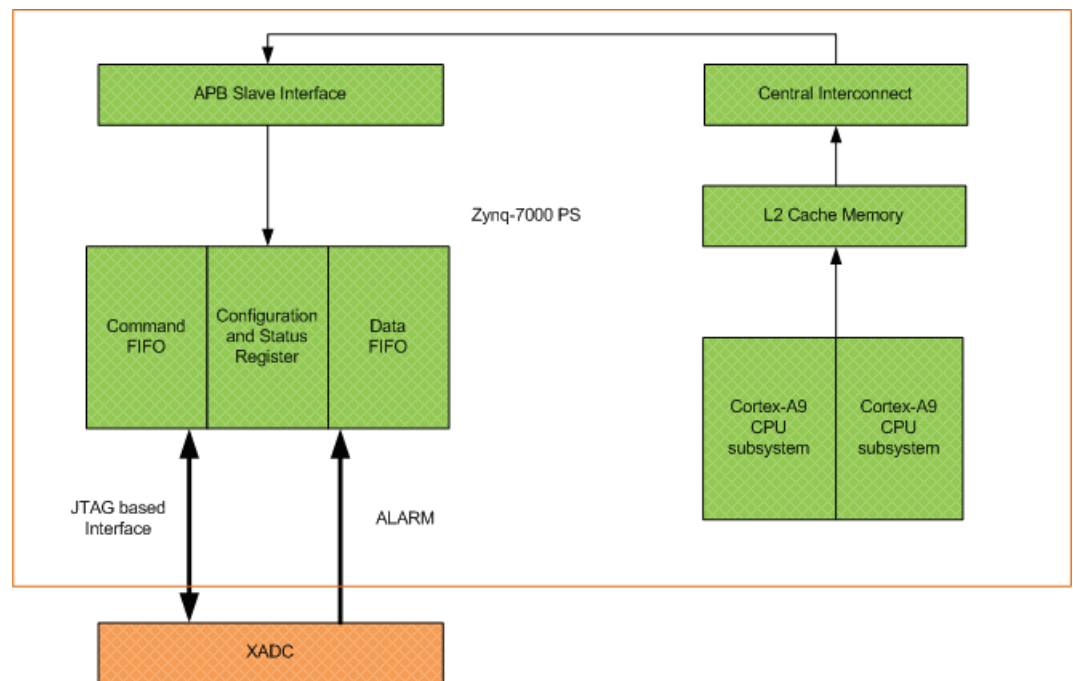
1. Showcases how to quickly establish an interface between the Zynq-7000 PS and the XADC
2. Presents an example code that covers the typical system monitoring use cases including alarms
3. Publishes the performance metric of the XADC to PS interface

The application note targets the Xilinx Zynq-7000 ZC702 evaluation board.

The XADC to PS interface cannot read data at the speed of 1 MHz (please refer to Experimental Results for more details on maximum signal bandwidth). For high speed signal processing application it is encouraged to use the PL interface of XADC. However the PL interface requires a bitstream to be created and the Linux driver interface to be modified. The dedicated XADC to PS interface should be used primarily for system monitoring applications.

Design Overview

This section details the design used in this application note. The block diagram of the design is shown in [Figure 1](#).



XAPP1172_01_042513

Figure 1: Design Block Diagram

The XADC to PS interface is normally controlled by software executing in the Application Processing Unit (APU). The software writes 32-bit commands followed by No-Operation (NOOP) commands to the command FIFO. The command FIFO output is serialized and sent to the XADC using the JTAG interface. There is a programmable idle gap time between packets to enable the XADC to load data read in response to the previous command. For every word shifted out of the command FIFO, a corresponding word is shifted in to the data FIFO. In the case of a DRP read command, as data is shifted out of the command FIFO, the old content of the XADC_DRP's DR register is shifted out. After IGAP time, the result of the current DRP read is available in the XADC DRP's DR register. When the next command from TXFIFO is shifted out, the result of the current read which is in the DR register is shifted into the RDFIFO.

Note: The NOOP command is required to read out the result from the Data FIFO present in the Zynq-7000 PS. For a series of read commands, the user can write the commands back to back in the command FIFO and read out the results with a single NOOP command (call this as bursty read mode).

The Experimental Results section provides a comparative data of single read mode vs the bursty read mode.

Software Architecture

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

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

The subsystem provides the following facilities to the user space:

- Sysfs interface for communicating with the devices from user space.
- Character driver interface for receiving the event information

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

1. XADC core

This is a static library which is solely responsible for communicating with the IIO system for hardware configuration, retrieving sensor values, and event handling.

2. Webserver

The Webserver listens to the connection request from the remote web clients. The web clients running on 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 web client and passes it over to the XADC core; so as to configure the hardware for event/alarm generation.

3. Web interface

This section is responsible to stitch the Webserver with the XADC core. It uses the API exposed by the XADC core to get the sensor and alarm values, and to set the thresholds. It passes these values to and from the Webserver.

4. Web Client

The web client runs on the remote system in the web browser. The client communicates with the web server on http port 9090.

The client runs a GUI application used as front end by the user. The GUI displays the captured sensor data along with graphs. It also provides an interface for the user to program threshold values for the sensor events.

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

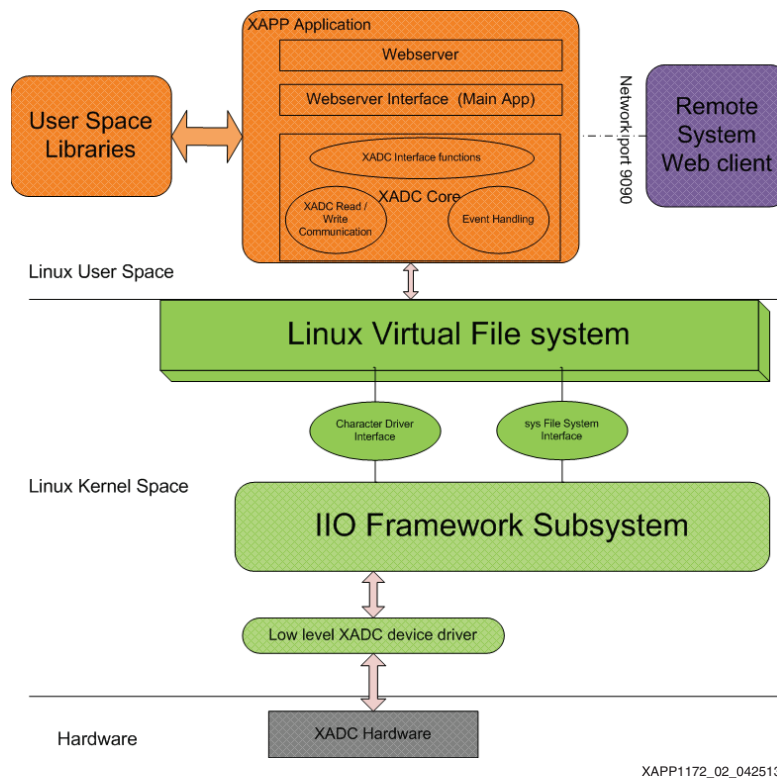


Figure 2: Software Block Diagram

XADC Core APIs Used in this Application

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.

Descriptions for the enums, structures and APIs declared in the `xadc_core_if.h` file follows.

Enumerations

- `XADC_Parm`
 - This enum defines the available parameters that can be queried for the statistic.
- `XADC_Alarm`
 - This enum defines the available Alarms on the system, which can be programmed and queried for the status.
- `XADC_Init_Type`
 - This enum defines the type of initialization for the xadc core. The valid values are `READ_ONLY` and `FULL`, where later allows to initialize the event handling thread.

Structures

- `Xadc_callback`
 - This structure contains the call back information for the alarm. It has function pointer and argument pointer that should be passed for the function.

Function APIs

int xadc_core_init(enum XADC_Init_Type type);;

This function should be called once in the beginning of the program. It is responsible for finding the XADC device nodes and to initialize global variables. If called with init type FULL, it also spawns the event handling thread which monitors the alarms and handles the particular event.

Argument: type: enum value for the type of initialization. READ_ONLY type, if event handling is not needed. FULL type, if event handling is also required.

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 the resources.

Argument: N/A

Return Value: [Integer] 0: Success

void xadc_update_stat(void);

This function just updates the global caches for all the statistic parameters. This should be called before 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 the given parameter. For voltage, the return value is in mV and for temperature in degree Celsius.

Argument: Parameter: enum value for the parameter for which 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: enum value for the parameter for which 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: enum value for the alarm for which threshold are set.
- threshold_low: Low threshold value for the alarm [mV /degreeC]
- threshold_high: High threshold value for the alarm [mV /degreeC]
- callback: callback information for the event on the given alarm. If NULL is passed in this arg, no call back will be registered otherwise on event occurrence callback→ func(arg) will be 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 case, one don't need callback but just get status of the event at some stage.

Argument: Alarm: enum 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 hardware-specific operations.

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

Sysfs Interface

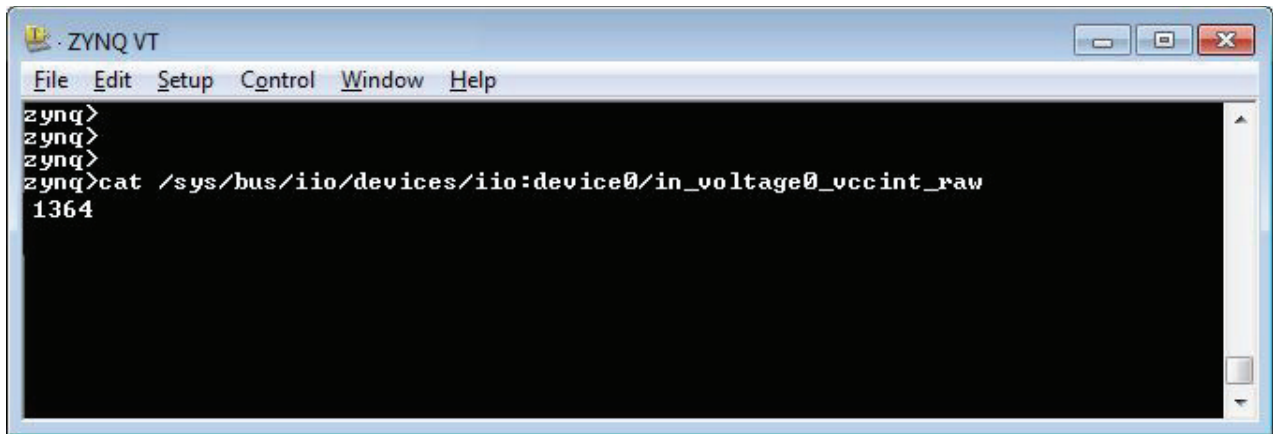
If there is an XADC entry in the `dtb` file, the IIO subsystem populates the sysfs interface for the XADC. The sysfs entries can be found in the `/sys/bus/iio/devices/<populated-device>`.

The correct drive can be confirmed by reading `/sys/bus/iio/devices/<populated-device>/name` file (it is read as `xadc` in this case).

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

For example, to get the sensor raw code value of `vccint`, (the user can use the “cat” command to read different values from sysfs interface):

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



```

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

```

XAPP1172_03_042913

Figure 3: Command Line Example

Command Line Execution

The application note provides command line interface for monitoring the internal VCCINT, VCCAUX and VCCBRAM channels. The commands can be executed from the command line shell of the Linux OS. These commands are installed in the system while the Linux OS boots up.

The following commands are available:

- Command: `xadc_get_value_vccint`
 - Explanation: This command returns the monitored VCCINT value.
 - Example: `$ xadc_get_value_vccint`, which returns the VCCINT value in mV
- Command: `xadc_get_value_vccaux`
 - Explanation: This command returns the monitored VCCAUX value.
 - Example: `$ xadc_get_value_vccaux`, which returns the VCCAUX value in mV
- Command: `xadc_get_value_vccbram`
 - Explanation: This command returns the monitored VCCBRAM value.
 - Example: `$ xadc_get_value_vccbram`, which returns the VCCBRAM value in mV
- Command: `xadc_get_value_temp`
 - Explanation: This command returns the monitored temperature value.
 - Example: `$ xadc_get_value_temp`, which returns the temperature value in °C.

Event Notification

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 vccint, write the raw code of the value to the following files respectively (the user can set a value in the sysfs interface using the “echo” command):

```
/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
```

To enable the vccint 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, there is char driver interface to get an event:

Proceed as follows (after setting the threshold) to receive the event:

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 the file descriptor (fd) retrieved by opening device file (in step 2) to get the event file descriptor:
 - `ioctl(fd, IIO_GET_EVENT_FD_IOCTL, &event_fd)`
4. Call read on the event_fd (from step 3):
 - a. `read(event_fd, &event, sizeof(event));`
 - b. Here *event* is of type struct `iio_event_data`.
 - c. Decipher this event as described in the `events.h` file to know the exact event.
5. This read is a blocking call, which is released when any event has occurred.

GUI Interface

The GUI for PS-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 panes:

- Control Pane
- Monitor Pane

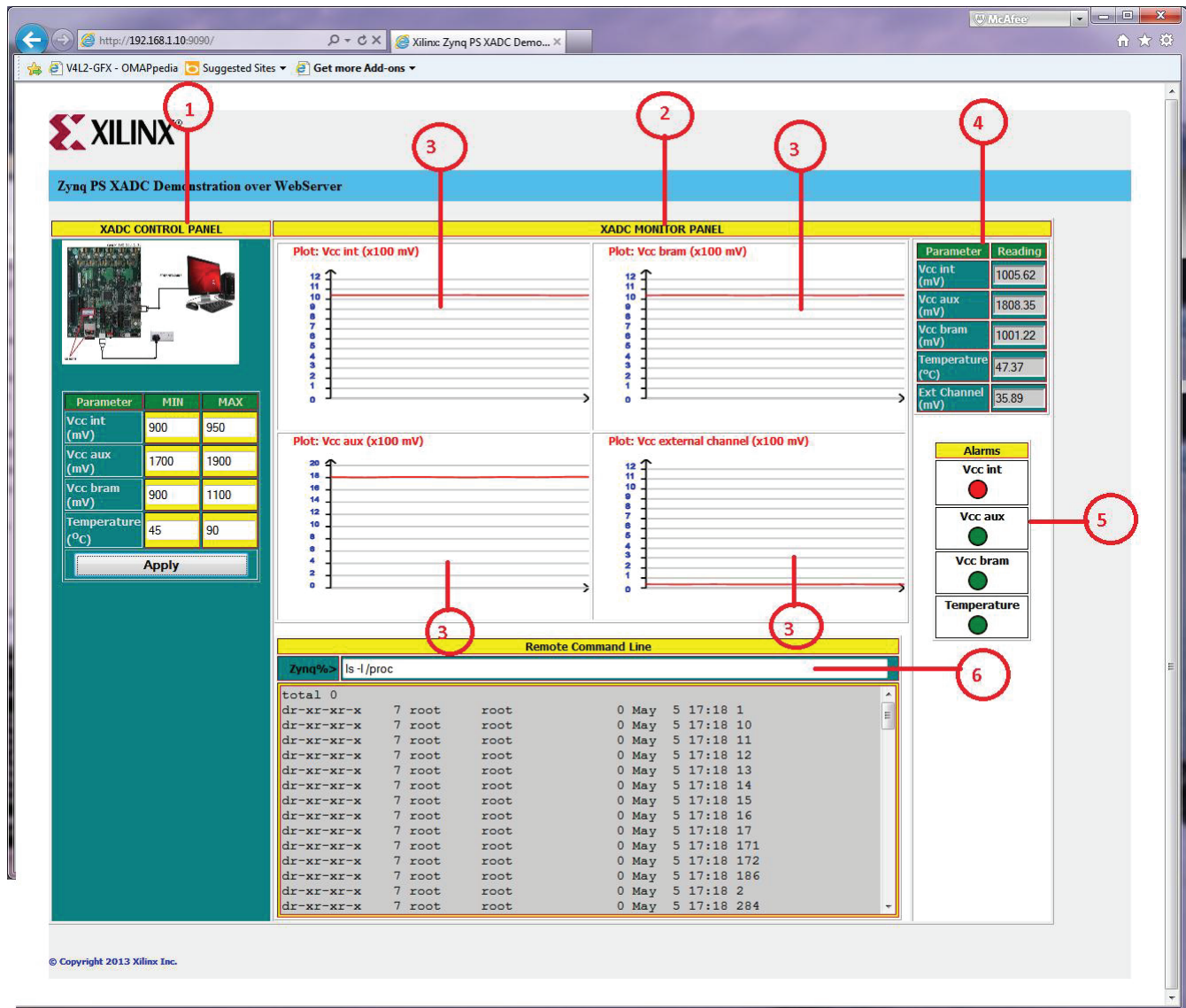
Control Pane

This pane is located on the left side of the screen ([Figure 4](#), no.1). The user can use this pane to modify the minimum and maximum threshold values for Vccint, Vccbram, Vccaux, and Temperature. When the user clicks the **Apply** button, these values are programmed in appropriate Zynq-7000 AP SoC register. This causes the PS-XADC alarm system to get adjusted to the new threshold values specified by the user.

Monitor Pane

The Web-client collects readings from XADC to PS periodically (once every second) and populates the readings on the Monitor Pane ([Figure 4](#), no.2).

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



XAPP1172_04_042913

Figure 4: Web Server Based GUI Application

Graphs

There are four linear graphs (Figure 4, no.3) plotted by the Web page:

- Vccint graph is plotted with a scale of 0–1,200 mV
- Vccbram graph is plotted with a scale of 0–1,200 mV
- Vccaux graph is plotted with a scale of 0–2,000 mV
- External voltage graph is plotted with a scale of 0–1,200 mV

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

Readings

The latest readings (Figure 4, no.4) from XADC to PS interface are displayed on the web page. Web-client probes the Zynq-7000 AP SoC XADC to PS every second and gets the latest readings. These values are plotted on the graphs.

Alarms

The Zynq-7000 XADC to PS interface has the capability to monitor whether the current value of a parameter is out of bounds (minimum, maximum threshold values). If so, a corresponding alarm is raised (Figure 4, no.5).

The web page tracks and update the alarm status for four parameters (Vccint, Vccaux, Vccbram, and Temperature). The color of each alarm is turned to green (within bounds) or red (out of bounds) in real time.

A user can program the MIN, MAX threshold values of a parameter and test working of alarms. The alarm turns red as soon as the values go beyond the programmed threshold.

Remote Command Line

The remote command window (Figure 4, no.6) is an extended Linux prompt onto the Web page. This control acts as shell prompt for the Zynq-7000 AP SoC board. The user can treat it as a Zynq shell, and enter any valid Linux command. The text is carried to Zynq-7000 AP SoC board and attempts to execute as a Linux command. After executing the command, the Zynq-7000 AP SoC board sends the results to the Web page and is displayed on the console.

Note that, the command entered here is executed in a temporary process and the process 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 (it does not stay in the new changed folder).

External Channel Measurement

For measuring the voltage level in the external channel, the IIO framework provides external channel measurement option. The user must enable the external channel intended to be monitored in the device tree source (dts) file. In the ZC702 evaluation kit, an AMS101 daughter card is included which inserts into the XADC header pin. For more details on connecting an external signal source to the AMS101 card, refer to the *AMS101 Evaluation user Guide*, (UG886).

Note: While using the external channel in the XADC, the user does not have to create a bitstream with the appropriate user constraint directive applied for external channels.

Hardware Requirement

The design can be tested using the ZC702 evaluation platform. To collect the external data using the AMS101 evaluation card, the user needs to follow the instructions in UG886 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 XADC to PS interface.

Experimental Results

The data from the 32-bit command FIFO is serialized using the clock generated by dividing the PCAP clock by the TCLK_RATE parameter (see *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585) for further details). With TCLK_RATE = 4 and PCAP clock set to 200 MHz, the frequency of the generated clock is $200 \text{ MHz} / 4 = 50 \text{ MHz}$. Serializing 32-bit command takes $32 \times 20 \text{ ns} = 0.64 \mu\text{s}$ time.

IGAP value of 20 imparts a delay of $20 \times 20 \text{ ns} = 0.4 \mu\text{s}$ between two commands. When a NOOP command is written after writing an XADC read command, the serialization of the command takes 0.64 μs . Thus, one write command takes $0.64 + 0.4 + 0.64 = 1.68 \mu\text{s}$ theoretically.

Linux OS causes additional delay in the data acquisition from the XADC using the XADC to PS interface.

The maximum external signal bandwidth that can be supported with the XADC to PS interface in Zynq-7000 family has been measured and found to be 100 kHz. The measurement is taken in ideal conditions with the CPU running only the XADC to PS application. No other applications are run. The measurement is taken with alarm events enabled in the driver. The limitation on the maximum signal bandwidth is due to the serial nature of the XADC to PS interface. The full XADC bandwidth of 1 MSPS can be explored by using the DRP interface of the XADC.

The latency numbers for single and bursty read commands have been experimentally found. [Table 1](#) summarizes these values.

Table 1: Latency Numbers for Single and Bursty Read Commands

Command Type	IGAP	TCLK_RATE	PCAP Clock Frequency (MHz)	Worst Case Latency (μ s)	Average Latency (μ s)
Single Command	20	4	200	8.78	3.92
Bursty Read (for 15 read commands)	20	4	200	24.96	13.15

Conclusion

This design provides the platform required to use the dedicated XADC to PS interface for system monitoring. The design also explores the possibility of using an external auxiliary channel through the XADC to PS interface and characterizes the maximum signal frequency which can be monitored using the interface. The external auxiliary channel can be used to monitor external voltage and current which is not captured by internal sensor.

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

Reference Design

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

<http://secure.xilinx.com/webreg/clickthrough.do?cid=343190>

Follow the instructions provided in the readme file to build hardware and software code. Refer to the following wiki page for the application note build and run instructions:

<http://www.wiki.xilinx.com/zynq-xadc-ps-xapp>

Table 2: Reference Design Matrix

Parameter	Description
General	
Developer Name	Xilinx
Target Devices	Zynq-7000 AP SOC
Source Code Provided	Yes
Source Code Format	C
Design uses code and IP from existing Xilinx application note and reference designs, CORE Generator software, or third part	Yes
Simulation	
Functional Simulation Performed	No
Timing Simulation Performed	No
Test bench used for functional and timing Simulations	No

Table 2: Reference Design Matrix (Cont'd)

Parameter	Description
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulation	N/A
Implementation	
Synthesis software tools/version used	N/A
Implementation software tools/versions used	N/A
Static timing analysis performed	No
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	ZC702

Appendix A: Device Tree

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

```
xadc@f8007100 {
    compatible = "xlnx,ps7-xadc-1.00.a";
    reg = <0xf8007100 0x20>;
    interrupts = <0 7 4>;
    interrupt-parent = <&gic>;
    clocks = <&ps_clk>;
    xlnx,channels {
        #address-cells = <1>;
        #size-cells = <0>;
        channel@0 {
            reg = <0>;
        };
    };
};
```

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

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

References

1. *Zynq-7000 All Programmable SoC Technical Reference Manual* ([UG585](#))
2. *7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide* ([UG480](#))
3. IIO framework: <https://wiki.analog.com/software/linux/docs/iio/iio>
4. *AMS101 Evaluation Card User Guide* ([UG886](#))

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
07/03/2013	1.0	Initial Xilinx release.
03/18/2014	1.0.1	Corrected “comments” to “commands” in Table 1 and “Statistic” to “Static” in Table 2 .

Notice of 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 the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; 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 Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.