XAPP1203 (v1.0) April 22, 2014

Implementation of Signal Processing IP on Zynq-7000 AP SoC to Post-Process XADC Samples

Authors: Mrinal J. Sarmah and Cathal Murphy

Summary

This application note is a follow up and companion to the white paper *Efficient Implementation of Analog Signal Processing Functions in Xilinx All Programmable Devices* (WP442) [Ref 1]. The white paper proposes a simple and easy design flow for implementing analog signal processing functions in Xilinx FPGAs and All Programmable SoCs (AP SoCs), leveraging Xilinx All Programmable Abstractions. This application note describes in detail how to leverage the concepts outlined in the white paper to build signal processing IP cores and a complete mixed signal system easily on the Zynq-7000 AP SoC.

Introduction

Most systems need to be able to interact with the real world for the purpose of monitoring and controlling it. To allow this to happen, systems contain sensors that translate real-world stimuli such as light, heat, and sound into analog electrical signals. The analog outputs of these sensors need to be processed and digitized so that the appropriate information can be presented to the digital controller or processor. The analog processing and conditioning of the sensor output is done in many different ways, depending on factors such as the accuracy requirement and the type of sensor used. Common analog functions include:

- Level translation (bipolar to single-ended)
- Gain/attenuation
- Bandwidth limiting/filtering/noise reduction
- Gain and offset error cancellation
- Linearization

As most systems process digital information, an analog-to-digital converter (ADC) is required to digitize the processed sensor output. These ADCs come in many forms and are typically defined by their resolution and speed. ADCs are now also integrated into microcontrollers and FPGAs such as the XADC, which can be found in all 7 series FPGAs and AP SoCs. ADC speeds have increased significantly over the years due to process scaling. It is now relatively inexpensive to buy ADCs with sample rates in the range of 1 MSPS and greater. This increase in speed, together with digital signal processing, can be leveraged to enhance the performance of the solution.

The white paper *Efficient Implementation of Analog Signal Processing Functions in Xilinx All Programmable Devices* (WP442) introduced the concept of using Xilinx All Programmable Abstractions to implement common analog signal processing functions easily and efficiently in Xilinx FPGAs and AP SoCs. It showed that in doing so, the resulting subsystems were more accurate, flexible, cheaper, and faster to design.

This application note describes exactly how to apply these concepts to Xilinx FPGAs or AP SoCs for specific analog signal processing using high-level synthesis (HLS). The functions include:

- Low-pass filtering
- Decimation
- Linearization

This application note describes in detail how a complete mixed signal subsystem can be assembled in the Vivado® Design Suite IP Integrator (IPI), providing a complete reference design for the digital subsystem shown in Figure 1.

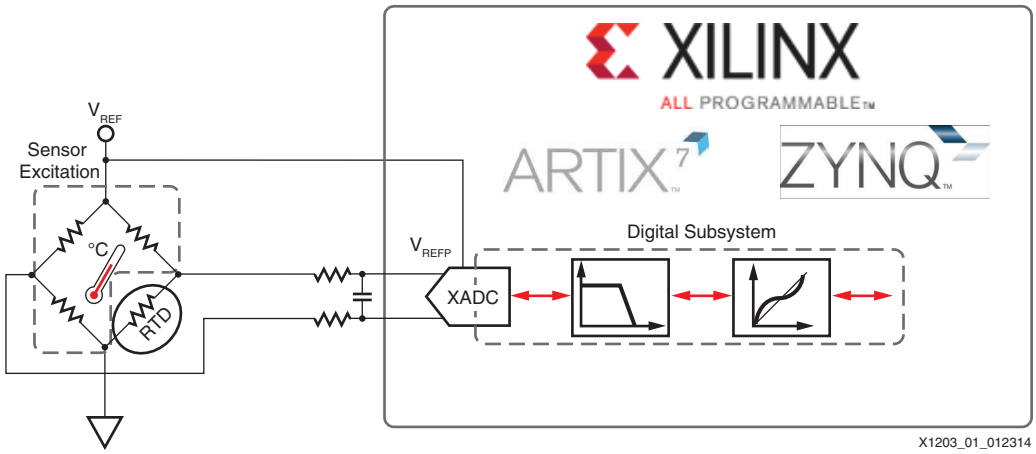

Figure 1: **System Block Diagram**

A UART-based application (HyperTerminal or TeraTerm) can be used to evaluate the complete subsystem or the individual IP cores. The user can refer to the steps mentioned in the Zynq AMS Post Processing in PL wiki page for detailed implementation steps.

Hardware Design Overview

FPGAs contain arrays of configurable logic blocks (CLBs) that can be configured to perform a specific digital algorithm in a cost-effective manner. The programmable logic portion of Zynq-7000 AP SoCs has arrays of CLBs to perform digital algorithms written in C or C++ or Register Transfer Level (RTL) code. The Vivado Design Suite assists users to implement the desired algorithm in a more reusable and IP-centric manner.

The Vivado High Level Synthesis (HLS) tool is capable of generating RTL code from an algorithm written in C or C++. The tool uses a feature-rich library to convert functions in C/C++ to RTL. The user can use appropriate HLS directives to interpret AXI4-Stream or AXI4-Lite interface options so as to enable reusability of the generated RTL across multiple system-level designs.

The Vivado IP Integrator (IPI) is another tool that enables instantiation of an IP in a design that has schematic-based connections. IPI has Connection Automation and Block Automation features to enable automation of instantiation for appropriate AXI4 interconnect and connections to appropriate master and slave IP cores in the design. The user can package the generated RTL from the Vivado HLS, define appropriate interfaces, instantiate the IP in the user block design, and use the Connection Automation tool to connect the interfaces.

In the example, the Sensor Linearizer algorithm has been implemented in C. A code snippet is provided below:

```
void axi_sensor_lin(ap_axiu<32,5,1,1> *input,
          ap_axiu<32,5,1,1> *output,
          unsigned int interp_matrix[1024], unsigned char gain, unsigned char offset, unsigned
char control)
 {
    AP_BUS_AXI_STREAMD(input, BUS_A);
    AP_BUS_AXI_STREAMD(output, BUS_B);

    unsigned short step_height, step_width, index, tmp_value, input_data;
    unsigned short interp_data, next_interp_data;

    input_data = (unsigned short)((input->data)&0xFFFF)>>4;
    step_width = (input_data % 4);
    index = input_data / 4;

    interp_data = (unsigned short)(interp_matrix[index]&0xFFFF);
    next_interp_data = (unsigned short)(interp_matrix[index+1]&0xFFFF);
```

```
        step_height = (-interp_data+next_interp_data);
        tmp_value = interp_data + step_height * step_width / 4;
        output->data = (unsigned int)((tmp_value+offset)*gain/128)&0xFFFF;
    }
```

The function axi_sensor_lin takes non-linear AXI4-Stream data, the linearizer coefficients, and gain and offset as input and gives out the linearized output data. The function calculates the output data based on linearizer coefficients and a distance metric of the current and the next values of the linearizer coefficient matrix. The AP_BUS_AXI_STREAMD directive is used to infer the AXI4-Stream interface for the input and the output bus. The Vivado HLS generates the RTL code with the AXI4-Stream interface for input and output data, which can be packaged using IP Packager in the Vivado tools.

A block diagram of the design is shown in Figure 2. The design comprises a control processor and the analog post-processing IP implemented in the programmable logic (PL) of the Zynq-7000 AP SoC. The analog post-processing IP cores use the DSP blocks available in the PL.

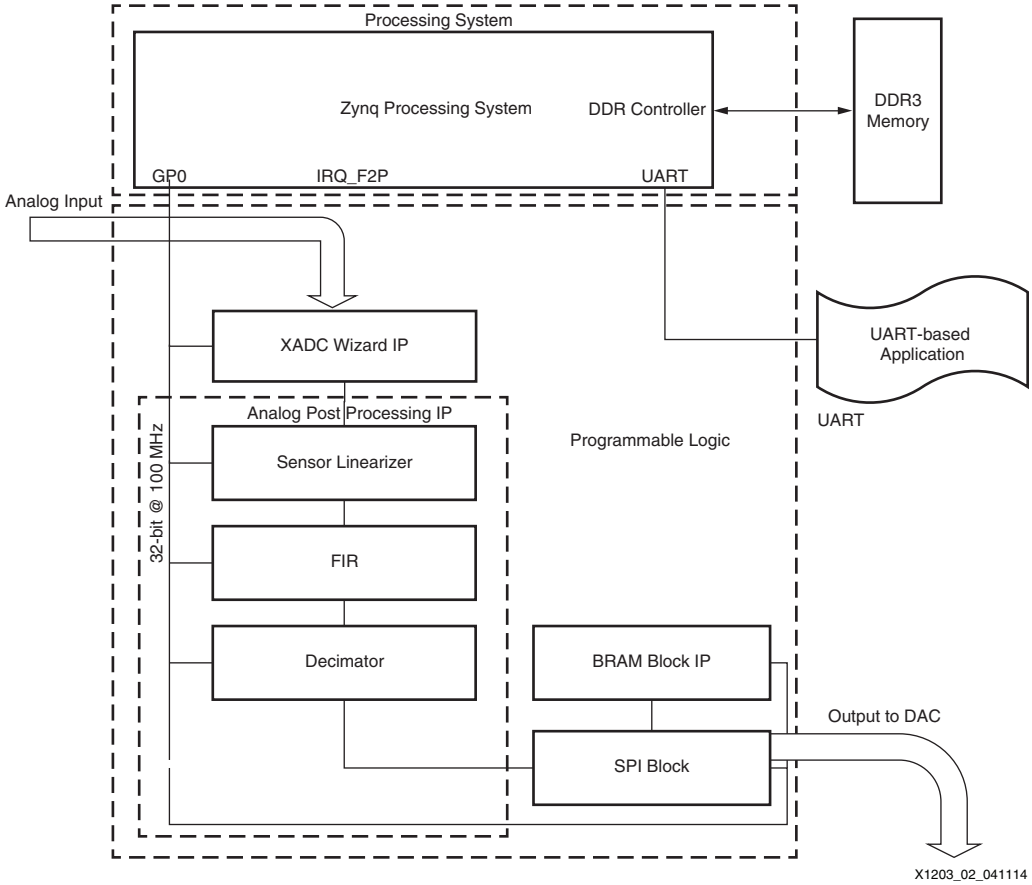

Figure 2: **Hardware Block Diagram**

The dual-core ARM® Cortex™-A9 based processing system (PS) of the Zynq-7000 AP SoC acts as a control processor and initializes and configures the different peripheral blocks used in the design.

The control path is established over an AXI4-Lite interface originating in the general purpose (GP) port of the Zynq-7000 processing system-7 IP. The data path in the design establishes an AXI4-Stream interface between the interconnecting blocks.

The following sequence describes the data flow:

1. The control software initializes the XADC Wizard IP, and the Sensor Linearizer, Finite Impulse Response (FIR) filter, and Decimator blocks.
2. The analog input stimulus is sampled at the XADC V_P/V_N interface pins.
3. The XADC converts the analog samples to 16-bit digital code, and the samples are made available in the AXI4-Stream interface of the XADC Wizard IP.
4. The Linearizer block processes the samples if the block is enabled by the control software.
5. The linearized output goes to the FIR filter block, and the block generates a filtered output if the FIR filter is enabled by the control software.
6. The Decimator block processes the filtered output and decimates the samples by a programmable decimation factor.
7. The decimator output goes to the SPI core, which directs the samples to the BRAM Block IP. The SPI core also sends the samples to the AD5065 DAC that provides a stimulus to the XADC block.
8. The control software reads the samples from the BRAM Block IP using the AXI4-Lite interface.

XADC Wizard IP

The XADC Wizard IP is fully configurable using the AXI4-Lite interface. All the XADC registers are directly mapped onto the AXI4 interface in the same order. Any read or write into the memory space of the IP core is performed into the IP core itself.

Sensor Linearizer

This IP performs a linear interpolation look-up table (LUT). The algorithm finds the greatest difference between the ideal and actual result. The difference is added to or subtracted from the actual non-linear response to find the linear response. The initial values of the non-linear response have the greatest variation from the ideal response because the input varies very little for a big output change.

Figure 3 shows a block diagram of the Sensor Linearizer block used in the application note. The block has two primary interfaces: the AXI4-Stream interface used in the data path of the samples and the AXI4-Lite interface that is used as a control interface. The AXI4-Lite interface updates the BRAM block with the list of coefficients generated by the user. The Linearizer block's coefficients are determined by the nature of the non-linear function of the sensor generating the analog stimulus. The AXI4-Lite interface can run at a clock frequency that is different from the AXI4-Stream interface's clock frequency. The samples received in the slave AXI4-Stream interface are corrected using the linearizer coefficients to exhibit a linear behavior.

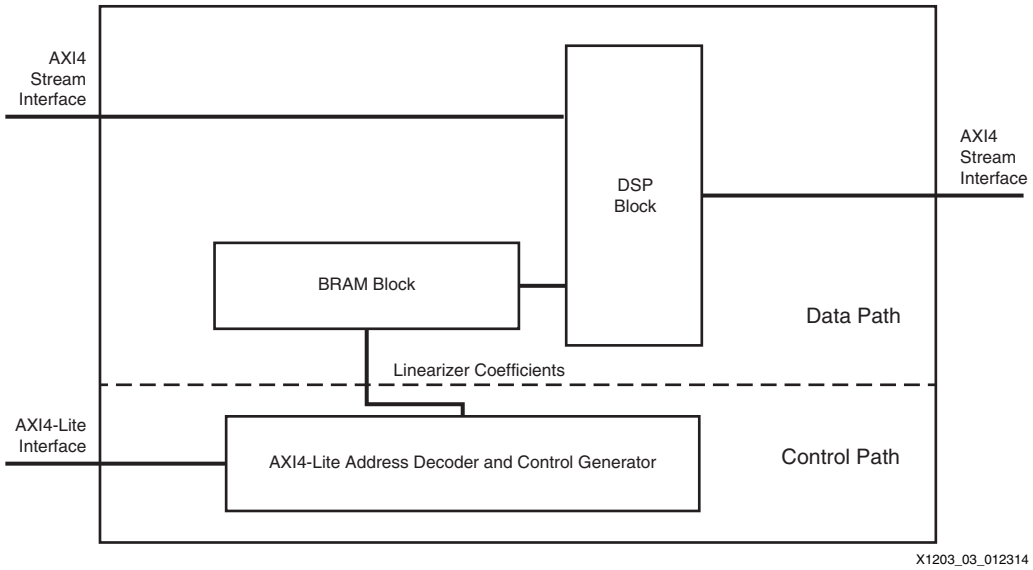

Figure 3: **Linearizer Block Diagram**

FIR Filter

The FIR filter IP core must be configured with the desired filter coefficients. A write into the 0x4 register writes a value into the coefficient's shift register. The lower 6 bits of the 0x0 register must be written with the number of coefficients of the filter. Bit position 16 is the software reset bit and position 17 is the bypass bit. A block diagram of the FIR filter is shown in Figure 4.

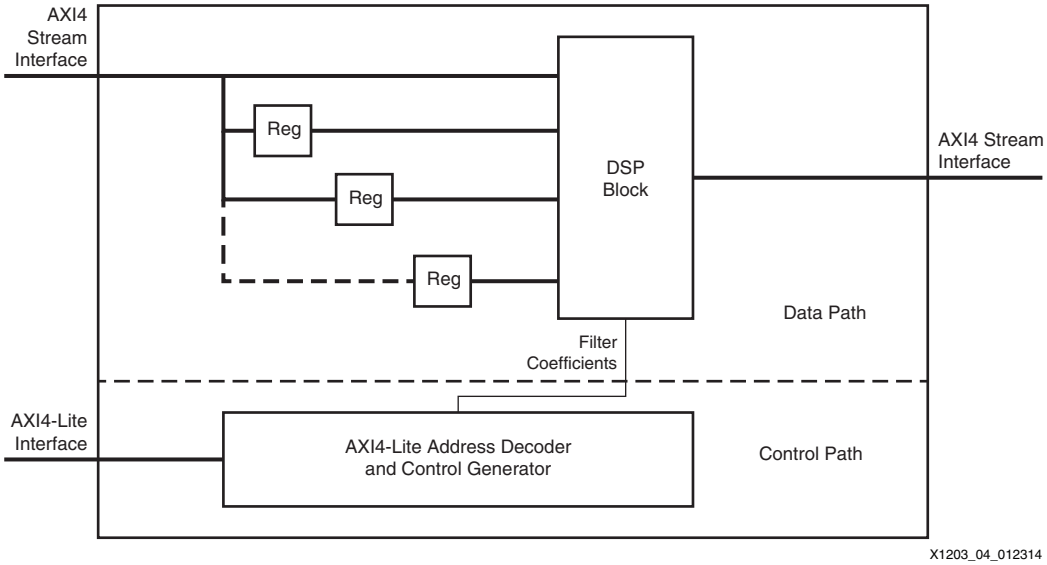

Figure 4: **FIR Filter Block Diagram**

The FIR filter block has two primary interfaces: the AXI4-Stream interface is used in the data path and the AXI4-Lite interface is used in the control path. The FIR filter coefficients generated by the user based on the FIR cutoff frequency requirement can be programmed using the AXI4-Lite interface. The samples arriving at the slave AXI4-Stream interface are delayed and multiplied by the coefficients to determine the output samples.

Decimator Block

This IP has one counter for every channel and outputs a value only when the counter reaches 0. The initial value of the counter is programmable through the AXI-Lite interface. The decimator block needs to work in conjunction with the FIR filter block. The output samples are decimated based on the programmed decimation value.

The blocks described above can be bypassed using the AXI4-Lite interface. Each of the individual blocks can be reset using a software-controlled reset that is applied via the AXI4-Lite interface associated with each of the blocks.

The reference design uses the Vivado IPI 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.

Figure 5 shows the IPI design used in the application note.

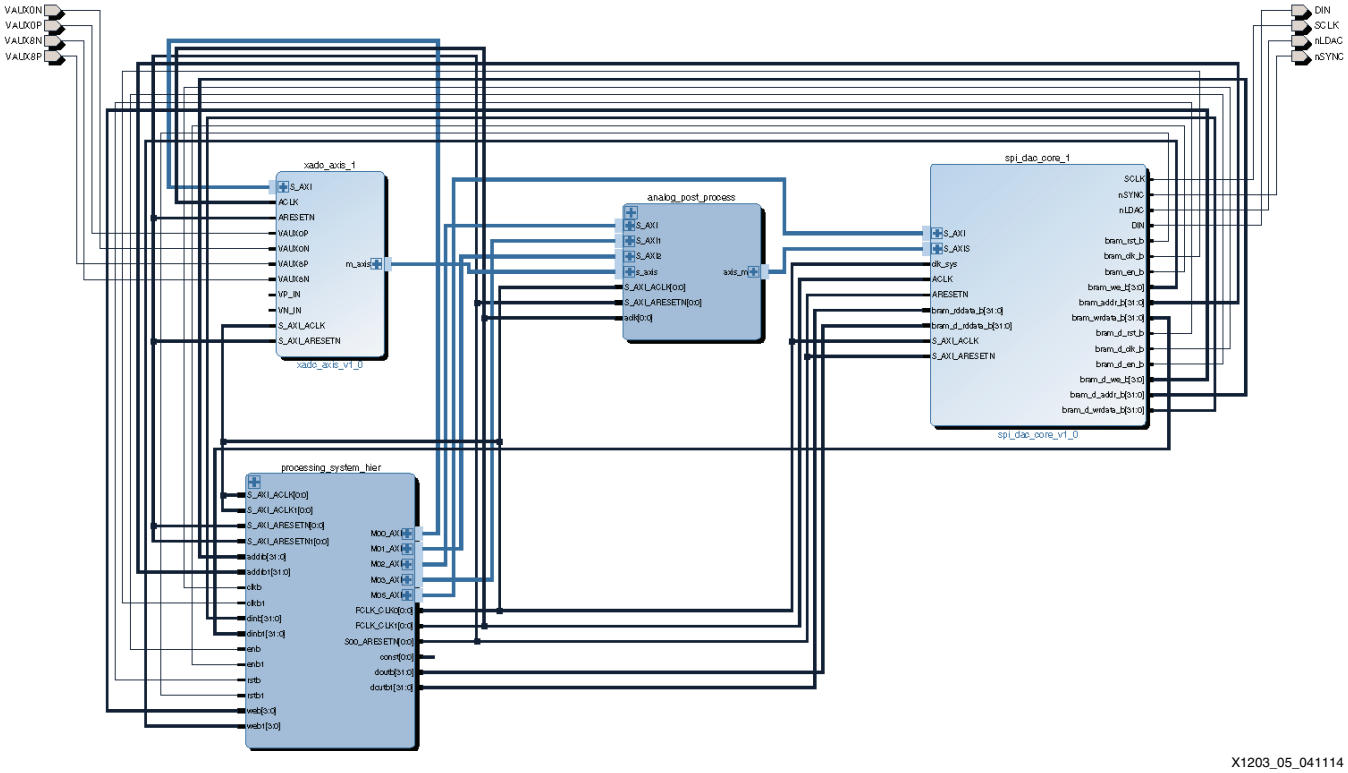

Figure 5: **IPI Block Design**

Figure 6 shows an expanded view of the analog post-processing block.

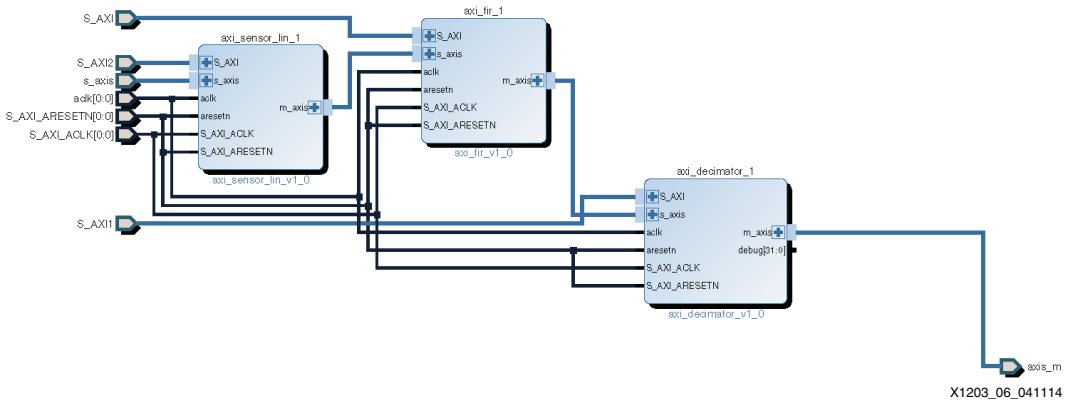

Figure 6: **Expanded View of Analog Post-Process Block**

Table 1 shows the IP cores used in the hardware design.

Table 1: **IP Cores Used in the Hardware Design**

IP Name	Description	Configuration
Processing System-7	Generates a wrapper that instantiates the Processing System-7 block.	Configured with the 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.
Proc Sys Reset	Applies reset to peripherals and AXI Interconnect IP.	Configured to apply reset to AXI Interconnect and XADC Wizard IP.
AXI BRAM Controller IP	Collects post processed samples from the signal processing blocks and stores them in the block RAM. The stored samples are read by the Processing System-7 IP.	Configured for true dual-port configuration.

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

Table 2: **Address Map of the Peripheral**

Peripheral in PL	Address Map (Hex)
XADC Wizard IP	70000000-7000FFFF
FIR Filter block	70600000-7060FFFF
Sensor Linearizer block	60000000-6000FFFF
Decimator block	70A00000-70A0FFFF
SPI core	70100000-7010FFFF
BRAM Control IP 1	40000000-4001FFFF
BRAM Control IP 2	42000000-4201FFFF

Software Architecture

The software application used in the application note is based on the stand-alone operating system available in the Xilinx Software Development Kit. The software application controls the configuration parameters of the design blocks and captures data from the AXI BRAM Controller in a periodic interval.

The following steps describe the data flow in the design. The software program:

1. Initializes the peripherals used in the application note
2. Sets default coefficients for the FIR filter
3. Loads the default Linearizer block coefficients
4. Configures the Decimator block with default decimation value
5. Waits on a while loop for commands entered from the UART-based application
6. Decodes the commands from the UART application and executes the command

The UART application configures the filter coefficients used in the design, sets the linearizer coefficients, and configures the decimation factor value. The UART application commands follow the format shown in Table 3.

Table 3: **UART Application Command Format**

Command	Description	Format
Read value from design	Reads a value on a particular address	R `AAAA` `AAAA` denotes the address in 16-bit hexadecimal format.
Write value to the design	Writes a value to a particular address	W `AAAA DDDD` `AAAA` denotes the address in 16-bit hexadecimal format. `DDDD` denotes the data to be written in 16-bit hexadecimal format.

The function APIs used in the application note are described below. All APIs are defined in the file `ams_xilinx_demo_hls.c`.

void config_fir (int_base_addr)

This API configures the FIR filter coefficients.

void config_xadc (void)

This API configures the XADC core registers. The core directly maps the XADC registers into the processor memory.

void decimation_config (XDecimator_XDecInstancePtr, int dec_value, int num_channel)

This API changes a specific channel of a specific decimator instance with the given decimation value.

int decimation_initialization (void)

This API configures both decimation cores with the same value. The 32-bit words pack four channel decimation values: (MSB) 3-2-1-0 (LSB).

void fill_memory_ramp (void)

This API fills the SPI to DAC memory with a ramp. The number of ramps in memory is programmable in this API.

void fill_memory_sine (int f)

This API fills the SPI to DAC memory with a sinusoidal of frequency *f*. The data is scaled to be positive and full swing.

int linearization_initialization (void)

This API configures the linearization coefficients. The LUT is directly mapped into system memory.

If the input signal is x_x, the linearization LUT must have sqrt(y) → sqrt(x_x) = x. The LUT has 1,024 positions. The input data has 2^{12}. This means that the LUT contains four data spaced values: 4096/1024 = 4.

int main ()

This is the main function of the reference design. The example design configures the XADC and FIR cores and provides a basic shell to modify several parameters using a serial port.

void toggle_bypass_decimator ()

This API toggles the bypass mode of the Decimation core.

void toggle_bypass_fir (unsigned int base_addr)

This API toggles the bypass mode of the FIR core.

void toggle_bypass_lin ()

This API toggles the bypass mode of the Sensor Linearization core.

void toggle_reset_fir (unsigned int base_addr)

This API toggles the reset mode of the FIR core.

Note: The user can use the TeraTerm application to configure the hardware design. Refer to the readme file in the ZIP for a list of supported UART commands.

Hardware Requirement

The design can be tested using the ZC702 evaluation platform. To collect external data by connecting the AMS101 card to the ZC702 evaluation board, follow the instructions described in *AMS101 Evaluation Card User Guide* (UG886) [Ref 2]. An external signal can be applied to the dedicated V_P/V_N channel of the XADC block, and collected samples can be read back using the UART application.

Comparative Analysis

This section summarizes the analog vs. digital implementation of the FIR filter and Sensor Linearizer IP. The analysis is based on the analog and digital implementation of the FIR filter and the Sensor Linearizer in MATLAB, and the results might vary slightly in the actual hardware implementation.

FIR Filter IP

The digital implementation of the FIR filter IP enables the user to implement a very high order filter IP that is more difficult to implement using analog components. As an example, it is possible to implement a 63 order FIR filter on an FPGA without much complexity and configurability, while the analog implementation would almost be very complex. The digital filter has better rolloff and stop-band attenuation.

Figure 7 shows the magnitude response of a 63 order digital filter and second order analog Sallen-Key filter. As observed in the figure, the digital filter has better rolloff and stop-band attenuation.

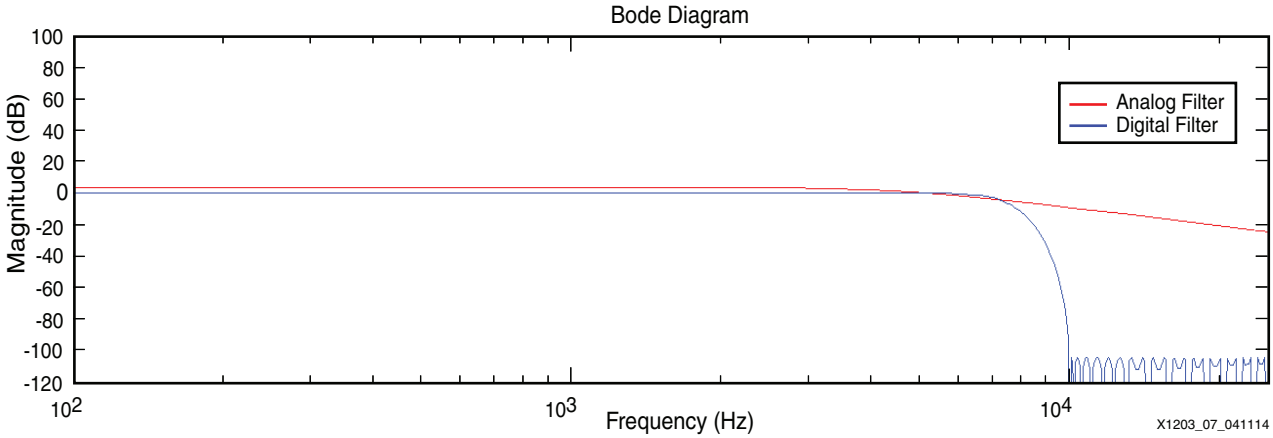

Figure 7: **Digital and Analog Filter Magnitude Response**

Sensor Linearizer IP

In the digital domain, linearization can be done very efficiently and accurately. The advantage of implementing the Sensor Linearizer IP in the digital domain is low resource requirements to implement the algorithm and less interpolation error, and thus better accuracy. In addition to correcting for any non-linearity, the digital approach can be leveraged to remove gain and offset errors.

Figure 8 shows the analog input before linearization, the ideal response, and the inverse linearization function that is used to generate the linearizer coefficients.

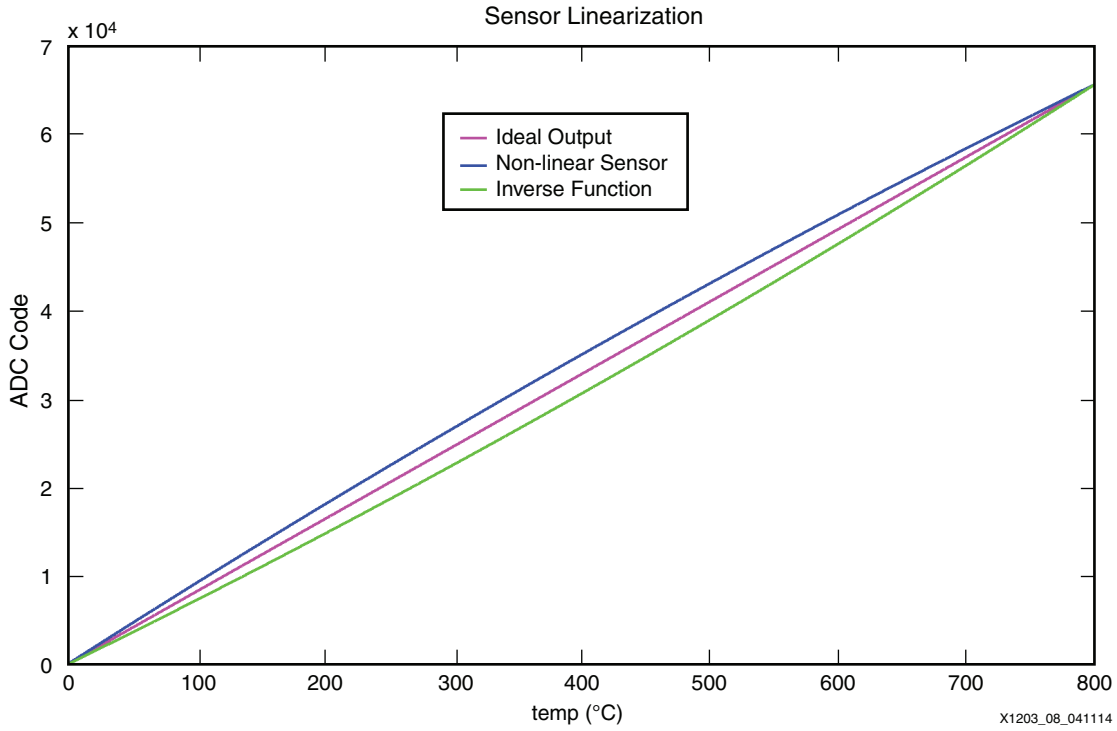

Figure 8: **Sensor Linearization Function**

Figure 9 shows the error plot of the digital linearization implementation that is computed based on the difference metric of the ideal and the interpolated linear function.

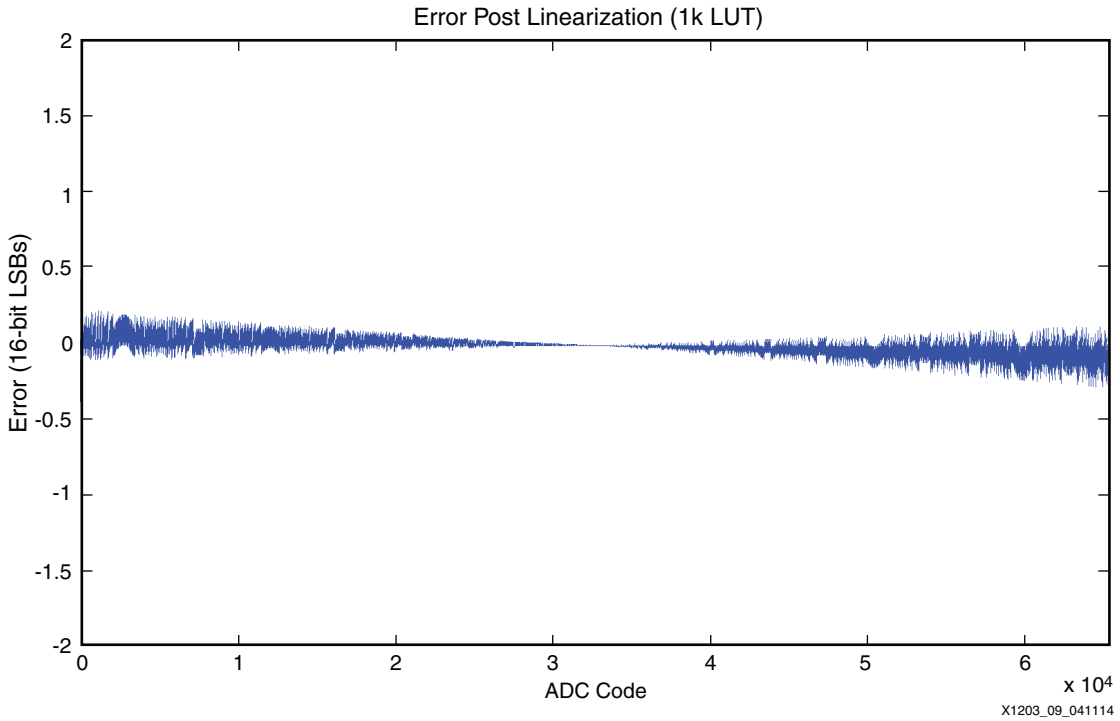

Figure 9: **Error Metric of the Digital Sensor Linearization IP**

The polynomial in Equation 1 is used for the computation.

$$RRTD = R0[1 + AT + BT2 + C(T - 100)T3] \quad \textit{Equation 1}$$

Where:

R0 is a 100Ω resistance at 0°C (Pt100)

A = 3.9083×10^{-3}

B = -5.775×10^{-7}

C = 0 for T > 0°C, or C = -4.23225×10^{-12} for T < 0°C

The error is calculated to be 0.00003%, as compared to 0.11% as shown in *Analog linearization of resistance temperature detectors* [Ref 7].

Experimental Results

The XADC has been configured for a sampling frequency value of 961.54 KSPS, which means the maximum signal bandwidth that can be acquired is 480 kHz. In the experimental setup, multiple frequency tones have been generated using an external sinusoid generator, and the resultant signal quality metric has been calculated.

The FIR filter IP has been verified with the filter coefficients generated for a 63 order FIR filter with cutoff frequency set to 10 kHz. With the use of the FIR filter in the design, a noticeable (~17 dB) improvement in signal-to-noise ratio (SNR) has been observed for frequencies lying within the cutoff frequency. The Sensor Linearizer IP has been verified to correct a second-order non-linear signal that is generated from the AD5065 DAC present on the AMS101 evaluator card.

Table 4 summarizes the resource utilization of the design. The resource numbers also include resources used for the AXI4-Lite Interconnect IP implementation logic.

Table 4: **Resource Utilization**

Design Block	Flip-Flops	LUTs	DSP Slices	Block RAM
FIR filter	324 (0.3%)	523 (0.9%)	1 (0.5%)	0
Linearizer block	442 (0.4%)	293 (0.5%)	1 (0.5%)	1 (0.7%)
Decimator	688 (0.6%)	823 (1.5%)	0	0
XADC block	31 (0.02%)	47 (0.08%)	0	1 (0.7%)

Reference Design

The reference design files for this application note can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=356125

Follow the instructions provided in the readme file for building the hardware and software code. Refer to the Zynq AMS Post Processing in PL wiki page for detailed implementation steps.

Table 5 shows the reference design matrix.

Table 5: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Xilinx
Target devices (stepping level, ES, production, speed grades)	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 party	Yes
Simulation	
Functional simulation performed	No
Timing simulation performed	No
Test bench used for functional and timing simulations	No
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/version used	Vivado Design Suite 2013.4
Implementation software tools/versions used	Vivado Design Suite 2013.4
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	ZC702 evaluation board

Conclusion

This application note demonstrates how samples from an analog-to-digital converter can be post processed to filter out environmental noise in a cost effective way. The design blocks used in the application note are light-weight solutions based on DSP blocks complying with standard AXI interfaces. The IP cores can be reused in the user design as a means to post process the XADC samples. The Vivado IPI-based flow enables ease of reuse in a schematic-based environment in which the designer does not have to work with the underlying RTL.

Appendix: LabView Serial Interface

Table 6 defines the register map for various UART application actions. The transmission format over the UART is the address first followed by the data byte. The address limit is 16 bits, and the data value is 16 bits.

For writes (16 bits of address and 16 bits of write data), an OK acknowledgement is provided.

For reads, a UART command of R AAAA is followed by the read data.

Table 6: **Register Map**

UART Application Commands	UART Transmission		UART Reception
	Address	Data	
Connection Establishment: Design Version Register (Read Only)	0x0000		Read-only design version register. Returns 16 bits of data. Design version (D11-8).(D7-0) UART Command: R 0000
Time Domain/Frequency Domain Tab: Collect Data (Read Only)	0x0001	-	The FPGA sends back 4,096 samples (8 KB) of raw data for further processing by the UART application. For simultaneous sampling mode, dual display is needed, and a fast Fourier transform (FFT) returns 8,192 samples with even samples corresponding to the VAUX[0] channel and odd samples corresponding to the VAUX[8] channel. UART Command: R 0001

References

This document uses the following references:

1. *Efficient Implementation of Analog Signal Processing Functions in Xilinx All Programmable Devices* (WP442)
2. *AMS101 Evaluation Card User Guide* (UG886)
3. AMS101 Evaluation Card Documentation www.xilinx.com/support/index.html/content/xilinx/en/supportNav/boards_and_kits/accessory_boards/ams101-evaluation-card.html
4. *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585)
5. *7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide* (UG480)
6. *7 Series FPGA AMS Targeted Reference Design User Guide* (UG960)
7. Analog linearization of resistance temperature detectors www.ti.com/lit/an/slyt442/slyt442.pdf

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
04/22/2014	1.0	Initial Xilinx release.

Notice of Disclaimer

Automotive Applications Disclaimer