DE1-SoC
ADC Controller

For Quartus II 14.1

1 Core Overview

The DE1-SoC ADC Controller IP Core provides an interface between a processor and the AD7928 Analog-to-Digital Converter present on the DE1-SoC board.

2 Functional Description

The DE1-SoC ADC Controller IP Core provides access to all 8 input channels of the AD7928 Analog-to-Digital Converter (ADC). It controls all required digital signals both to and from the ADC, and provides the user with a memory mapped register interface to read converted values. The core configures the ADC to operate in the 0 V - 5 V range.

3 Instantiating the Core in Qsys

The DE1-SoC ADC Controller IP core can be instantiated in a system using Qsys or as a standalone component from the IP Catalog within the Quartus II software. Designers use the core's configuration wizard to specify the number of channels to be read by the ADC Controller as shown in Figure 1. The ADC controller will only read channels 0 to *n*-1, where *n* is the number of channels specified in the wizard. Designers also specify the frequency of the clock that will drive the AD7928 chip. The acceptable clock frequency range is 0.01 MHz to 20 MHz.

4 Software Programming Model

4.1 Register Map

The DE1-SoC ADC IP Core provides eight registers for reading and two for writing, as shown in Table 1. The eight readable registers contain the outputs from the ADC for the eight analog inputs. The two writable registers are used to control the ADC. Writing to the *Update* register triggers an update of the stored conversions, and writing to *Auto-Update* enables or disables the automatic update feature.

4.1.1 Channel Registers

These eight registers hold the 12-bit outputs from the eight ADC channels. They are refreshed upon completion of an update operation. An update operation can be triggered manually by writing to the *Update* register, or automatically by enabling the Auto-Update mode via the *Auto-Update* register.

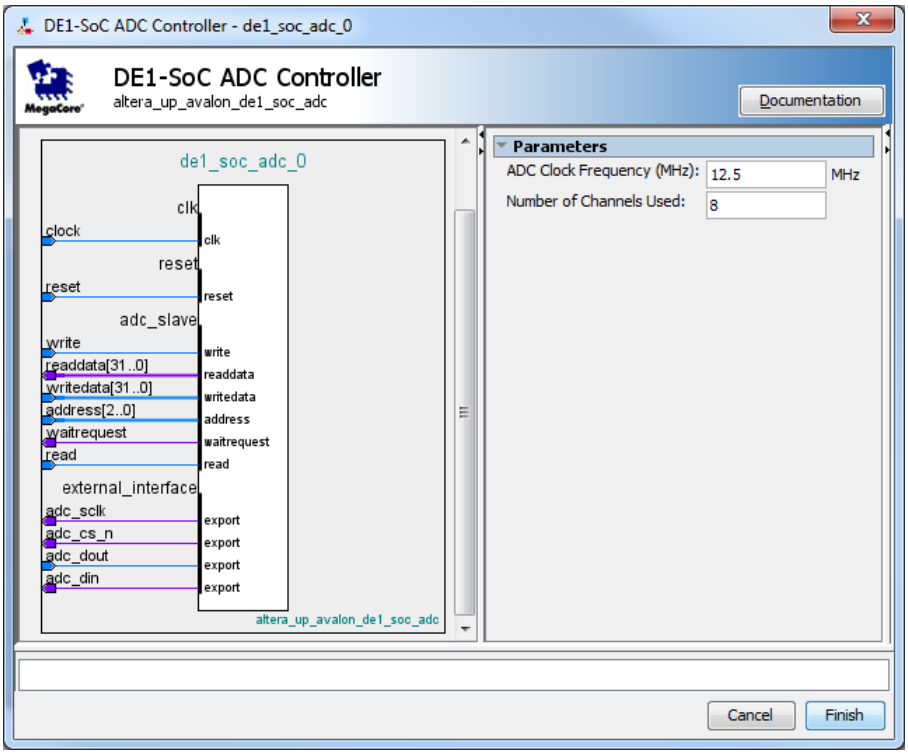

Figure 1. ADC Controller's Configuration Wizard.

Table 1. DE1-SoC ADC Controller register map			
Offset in bytes	**Register name**	**Read/Write**	**Purpose**
0	CH_0	R	Converted value of channel 0
	Update	W	Update the converted values
4	CH_1	R	Converted value of channel 1
	Auto-Update	W	Enables or disables auto-updating
8	CH_2	R	Converted value of channel 2
12	CH_3	R	Converted value of channel 3
16	CH_4	R	Converted value of channel 4
20	CH_5	R	Converted value of channel 5
24	CH_6	R	Converted value of channel 6
28	CH_7	R	Converted value of channel 7

4.1.2 *Update* Register

Writing any value to the *Update* register begins a conversion cycle on the ADC. During this time, all desired channels (as specified in the Qsys configuration wizard) are sampled. The new values become available in the Channel registers once the entire update operation has finished. If reads to the channel registers are attempted during the conversion cycle, the *wait_request* signal will be raised, causing the processor to stall until the update has finished.

4.1.3 *Auto-Update* Register

On system startup, this register will be loaded with a zero value. Writing a '1' to this register will enable auto-updating, while writing a '0' will disable it.

When auto-update is enabled, the system will automatically begin another update operation after the previous one finishes. Additionally, if reads to the channel registers are attempted during an update operation, the stored values from the previous update operation will be read without waiting for the latest update to finish. This is in contrast to a read during an update operation triggered by the *Update* register, where the *wait_request* signal would be asserted until the current update operation finishes.

4.2 Programming with the ADC Controller

The DE1-SoC ADC Controller core is packaged with C-language functions accessible through the hardware abstraction layer (HAL). These functions implement basic operations for the ADC Controller.

To use the functions, the C code must include the statement:

```
#include "altera_up_avalon_de1_soc_adc.h"
```

4.2.1 alt_up_de1_soc_adc_open_dev

Prototype: `alt_up_de1_soc_adc_dev* alt_up_de1_soc_adc_open_dev(const char *name)`
Include: `<altera_up_avalon_de1_soc_adc.h>`
Parameters: `name` – the ADC Controller name. For example, if the ADC controller name in Qsys is "ADC", then *name* should be "/dev/ADC"
Returns: The corresponding device structure, or NULL if the device is not found.
Description: Open the ADC controller device specified by *name* .

4.2.2 alt_up_de1_soc_adc_read

Prototype: `unsigned int alt_up_de1_soc_adc_read ( alt_up_de1_soc_adc_dev *adc, unsigned channel)`
Include: `<altera_up_avalon_de1_soc_adc.h>`
Parameters: `adc` – struct for the ADC controller device .
`channel` – the channel to be read, from 0 to 7.
Returns: data – The converted value from the desired channel.
Description: Read from a channel of the ADC.

4.2.3 alt_up_de1_soc_adc_update

Prototype: `void alt_up_de1_soc_adc_update(alt_up_de1_soc_adc_dev *adc)`
Include: `<altera_up_avalon_de1_soc_adc.h>`
Parameters: `adc` – struct for the ADC controller device .
Description: Trigger the controller to convert all channels and store the values.

4.2.4 alt_up_de1_soc_adc_auto_enable

Prototype: `void alt_up_de1_soc_adc_auto_enable(alt_up_de1_soc_adc_dev *adc)`
Include: `<altera_up_avalon_de1_soc_adc.h>`
Parameters: `adc` – struct for the ADC controller device .
Description: Enable automatic converting of channels.

4.2.5 alt_up_de1_soc_adc_auto_disable

Prototype: `void alt_up_de1_soc_adc_auto_disable(alt_up_de1_soc_adc_dev *adc)`
Include: `<altera_up_avalon_de1_soc_adc.h>`
Parameters: `adc` – struct for the ADC controller device .
Description: Disable automatic converting of channels.