SmartFusion cSoC: Using ACE with PDMA

Table of Contents

Introduction

The SmartFusion® customizable system-on-chip (cSoC) device contains a hard embedded microcontroller subsystem (MSS), programmable analog circuitry, and FPGA fabric consisting of logic tiles, static random access memory (SRAM), and phase-locked loops (PLLs). The MSS consists of a 100 MHz ARM® Cortex™-M3 processor, advanced high-performance bus (AHB) matrix, system registers, Ethernet MAC, DMA engine, real-time counter (RTC), embedded nonvolatile memory (eNVM), embedded SRAM (eSRAM), fabric interface controller (FIC), the Philips Inter-Integrated Circuit (I^2C), serial peripheral interface (SPI), and external memory controller (EMC).

The SmartFusion cSoC's programmable analog section contains the analog computing engine (ACE) and analog front-end (AFE) blocks. The SmartFusion cSoC devices have a versatile AFE that provides a very useful complement to the ARM Cortex-M3 microcontroller and general-purpose FPGA fabric. The SmartFusion cSoC devices have a sophisticated controller for the AFE called the ACE, which configures and sequences analog functions and post-processes the results, all independent of the Cortex-M3 processor, allowing it to work on other tasks. The SmartFusion cSoC devices have multiple analog-to-digital converters (ADCs) and digital-to-analog converters (DACs). The A2F200 has two of each, and the A2F500 has three of each. The SmartFusion DACs are the one-bit sigma-delta type. The ACE has a built-in first-order sigma-delta modulator that feeds each one-bit DAC. The DAC can be operated in either Voltage or Current Output mode. Refer to the *SmartFusion Programmable Analog User's Guide* for more details on programmable analog.

The peripheral DMA engine in MSS, which supports eight channels, facilitates data movement from ACE to an MSS memory location such as eSRAM. Alternatively, data can be moved to SRAM in the FPGA fabric. An APB master such as the Cortex-M3 processor, and the fabric master can always access the registers and SRAM in ACE (SSE and PPE) to move data themselves. However, this is very inefficient and wastes processor power. Instead, the dedicated PDMA engine should be used for moving data and the APB master should process only the moved PDMA data from the destination location. Refer to the *SmartFusion Microcontroller Subsystem (MSS) User's Guide* for more details on MSS Peripheral DMA.

In the ACE, RAW ADC data or PPE filtered data or PPE processed data is made available for the PDMA to transfer to other memory locations in the MSS or the FPGA fabric. This can be configured in the SmartDesign MSS configurator. Refer to the *MSS ACE Configuration User's Guide* for further details on the configuration. The PDMA data is available in the 32x32 dedicated FIFO in the ACE. This FIFO is written by PPE with RAW, filtered, processed data based on your selection. The ADC samples analog data at the pads, while the ADC datavalid is used as the write signal into the ADC FIFO (4 deep x 16 width). This ADC FIFO contains 12-bit raw ADC data and a 5-bit channel number. The PPE monitors the busy signals on ADC FIFOs in a round robin fashion, and starts processing as soon as it detects a non-empty ADC. The PPE microcode is responsible for pushing various data into the PDMA FIFO. If you have selected the check box in MSS ACE configurator, then the first thing the PPE does is to take the data from the ADC FIFO and puts it into the PDMA FIFO.

This application note and associated design files explain how to use PDMA to move ADC data from ACE to embedded SRAM. The Cortex-M3 processor is used as the APB master.

Design Example Overview

In the design example, PDMA is used to move ADC data from the ACE to MSS eSRAM. The Cortex-M3 processor acts as the APB master and processes the ADC data and extracts ADC results. The result is written to HyperTerminal interfaced with the UART0 peripheral. The application code resides in the eNVM.

Figure 1 shows the design example used with this application note to demonstrate using PDMA with ACE.

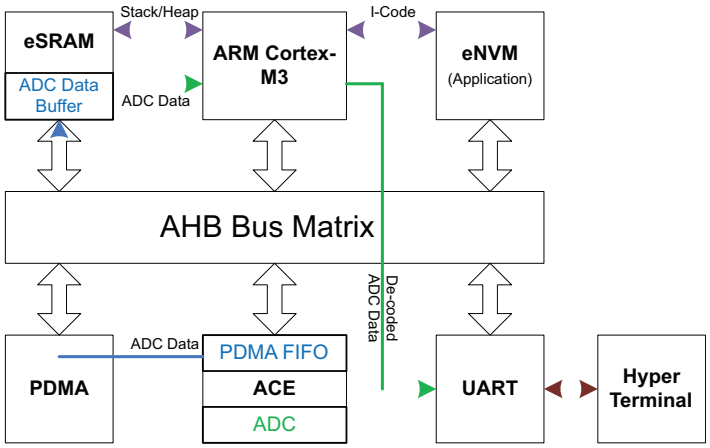

***Figure 1* • ACE PDMA Data Flow with MSS eSRAM**

This application note covers the following:

1. Configuring the hardware platform
2. Customizing software application
3. Configuring PDMA the channel for ADC data transfer
4. Processing of ADC data to extract ADC result

Configuring Hardware

The MSS ACE configurator generates analog configuration multiplexer (ACM), SSE, and PPE microcode specific to the application, based on your configuration of the analog services. The microcode is programmed into the eNVM along with system boot code when the SmartFusion cSoC device is programmed. On system start-up, the analog configuration is written into ACM (0x40020200-0x400203FC) and microcode is copied from the eNVM into SSE and PPE SRAMs by the Cortex-M3 processor as part of the system boot. This step customizes the hardware platform.

In the design example used with the application note, ACE is configured to monitor direct ADC input. ADC translates the analog input to digital values and stores it in ADC FIFO. In the MSS ACE configurator, select **send raw ADC result to DMA**. This results in PPE microcode, which moves data from ADC FIFO to PDMA FIFO.

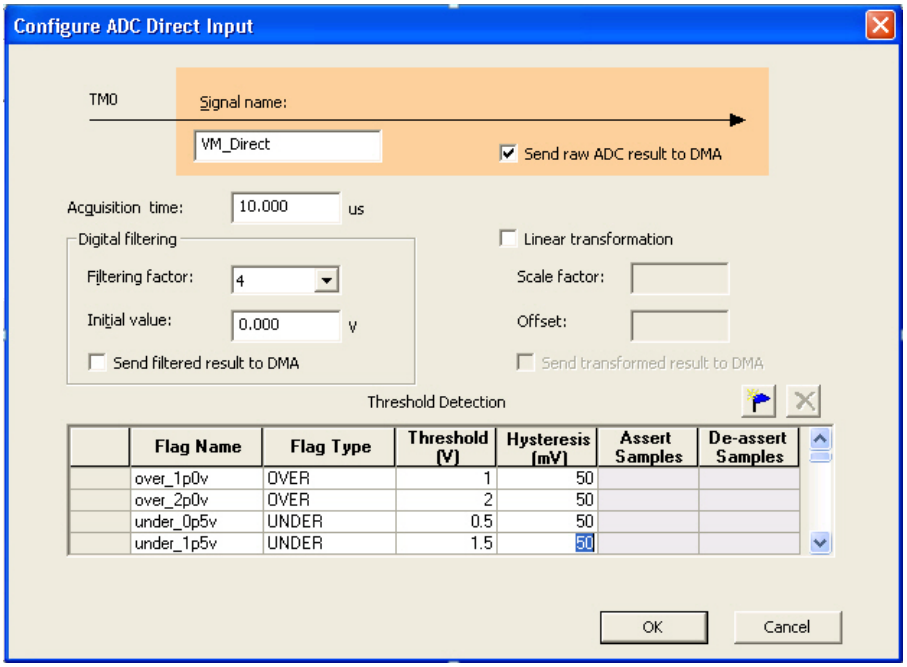

***Figure 2* • ACE Configuration with PDMA Option**

Design files provided with this application note have Verilog and VHDL projects. You must have STAPL files ready to program the SmartFusion cSoC devices.

Customizing Software

The ACE configuration information is written into customized header files and source files. These files reside in the folder <Libero_Project/Firmware/drivers_config/mss_ace>. These files contain the customization information you specified to the ACE, such as logical channel names, flag names, threshold values, and filtering. These files are referred to as driver configuration files.

The information contained in these files is used by the drivers to access and process data from the dedicated SRAMs, and appropriate registers in SSE and PPE. Refer to the following user's guides:

- *SmartFusion MSS ACE Driver User's Guide*
- *SmartFusion MSS PDMA Driver User's Guide*
- *MSS ACE Configuration User's Guide*

Design files provided with this application note contain a SoftConsole project built with drivers and driver configuration files corresponding to the Libero® System-on-Chip (SoC) software projects.

Note: It is very important that you copy the ACE driver configuration files into the software project and program the target hardware with the generated eNVM content; otherwise the hardware platform and software application would be out of synchronization, resulting in unexpected behavior.

Configuring PDMA Channel for ADC Data Transfer

Setting up PDMA involves the following steps. These are implemented in the c file main.c, which is part of the application project provided in the design files. The following sections provide an overview of these configuration steps.

PDMA Initialization

The PDMA engine must be initialized. Use the PDMA driver function PDMA_init() for this. This function resets PDMA and also clears any pending PDMA interrupts in the Cortex-M3 processor interrupt controller.

Usage:

```
PDMA_init():
```

Configuring one of the PDMA Channels with ACE as Source

The function PDMA_configure() is used for this purpose. In this case, PDMA channel 0 is configured with ACE as source for PDMA data transfer.

Usage:

```
  PDMA_configure(
  PDMA_CHANNEL_0,
      PDMA_FROM_ACE,
      PDMA_LOW_PRIORITY | PDMA_WORD_TRANSFER |PDMA_INC_DEST_FOUR_BYTES,
  PDMA_DEFAULT_WRITE_ADJ
      );
```

Managing PDMA Transfer

Actual PDMA transfer for the configured PDMA channel is managed using the function PDMA_start(). The source address for the PDMA is the ACE PDMA FIFO, which is a predefined source in the driver files.The destination address for the PDMA is a buffer in MSS eSRAM, defined in the application.

Usage:

```
PDMA_start
        (
        PDMA_CHANNEL_0,
         /* Source for PDMA */
        (uint32_t) PDMA_ACE_PPE_DATAOUT,
        /* Destination for PDMA */
        (uint32_t) g_samples_buffer[g_pdma_buffer_idx],
        SAMPLES_BUFFER_SIZE
       );
```

Setting up Interrupts

PDMA_set_irq_handler() function is used to associate the ISR function for interrupts generated on the completion of a transfer on the PDMA channel specified. This function enables the PDMA interrupt both in the PDMA controller and in the Cortex-M3 processor interrupt controller.

Usage:

```
PDMA_set_irq_handler(PDMA_CHANNEL_0, ace_pdma_handler);
```

Processing of ADC Data to Extract ADC Result

Once the data has been received by the PDMA in the destination memory location, the application must extract the ADC number, channel number, and the raw data from the 32-bit data received. The ACE driver function ACE_translate_pdma_value() is used to decode the ACE data received by the PDMA. Refer to the PDMA data format table in the *MSS ACE Configuration User's Guide* for details on the data format.

Usage:

```
volatile uint16_t raw_value;
raw_value =  ACE_translate_pdma_value(g_samples_buffer[proc_buffer_idx][i], 0);
```

Running the Design

Board Settings

The design example works on the SmartFusion Development Kit Board and the SmartFusion Evaluation Kit Board with default board settings. Refer to the following user's guides for default board settings:

- *SmartFusion Development Kit User's Guide*
- *SmartFusion Evaluation Kit User's Guide*

Program the Design

Program the SmartFusion Evaluation Kit Board (A2F-EVAL-KIT) or the SmartFusion Development Kit Board (A2F-DEV-KIT) with the provided STAPL file (refer to"Appendix A – Design Files" on page 7) using FlashPro and then power cycle the board.

The direct ADC input is assigned to pin W8, which is connected to the POT on the A2F-EVAL-KIT and A2F-DEV-KIT. Rotating the POT changes the input voltage applied at the ADC input. Refer to the *SmartFusion Evaluation Kit User's Guide* or the *SmartFusion Development Kit User's Guide* for schematics and pin mapping.

Running the Application

Invoke the SoftConsole IDE from the Libero SoC project (refer to "Appendix A – Design Files" on page 7) and launch the debugger. Start a HyperTerminal with 57600 baud rate, 8 data bits, 1 stop bit, no parity, and no flow control. If your PC does not have the HyperTerminal program, use any free serial terminal emulation program like PuTTY or Tera Term. Refer to the *Configuring Serial Terminal Emulation Programs* tutorial for configuring the HyperTerminal, Tera Term, and PuTTY.

When you run the debugger in SoftConsole, the HyperTerminal window displays a menu that allows controlling ADC result display (Figure 3).

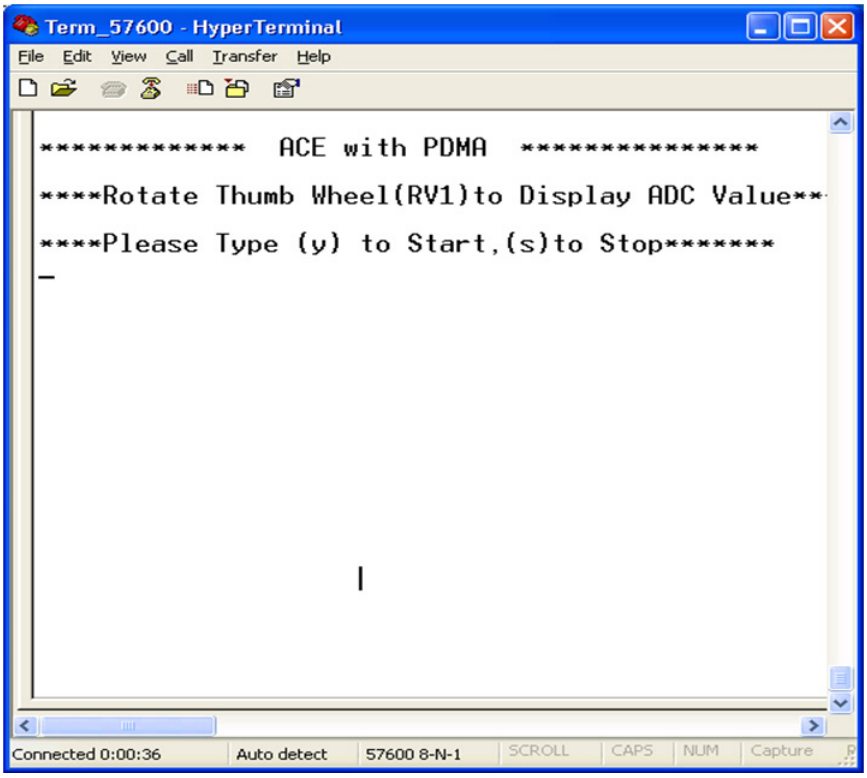

***Figure 3* • Menu Selection of ADC Result Display**

Type **y** to start the display of the ADC result. Note that as the POT is rotated, the display varies. You can stop the display by pressing **s** and resume the display by pressing **y**.

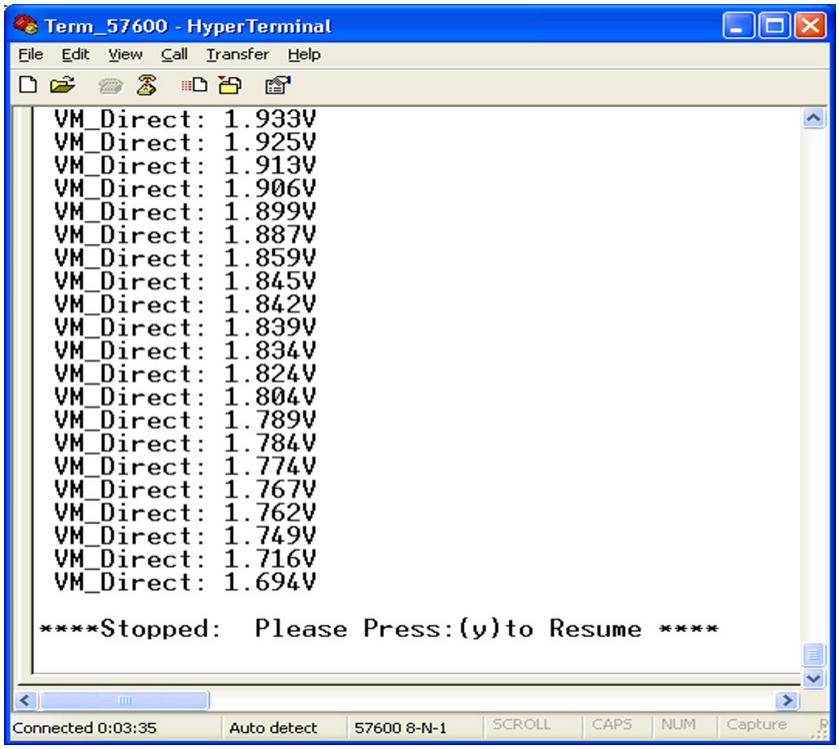

***Figure 4* • ADC Result Display**

Release Mode

The release mode programming file (STAPL) is also provided. Refer to the Readme.txt file included in the programming file for more information. Refer to the *Building Executable Image in Release Mode and Loading into eNVM tutorial* for more information on building an application in release mode.

Conclusion

This application note and design example demonstrate usage of PDMA with ACE to showcase the offload capability in SmartFusion cSoC devices.

Appendix A – Design Files

You can download the design files from the Microsemi SoC Products Group website:

www.microsemi.com/soc/download/rsc/?f=A2F_AC352_DF.

The design file consists of Libero SoC projects and programming file (*.stp) for A2F500 and A2F200. Refer to the Readme.txt file included in the design file for directory structure and description.

You can download the programming files (*.stp) in release mode from the Microsemi SoC Products Group website: www.microsemi.com/soc/download/rsc/?f=A2F_AC352_PF.

The programming file consists of STAPL programming files (*.stp) for A2F500 and A2F200, and a Readme.txt file.

List of Changes

The following table lists critical changes that were made in each revision of the document.

Revision*	Changes	Page
Revision 4 (January 2013)	Added "Board Settings" section (SAR 43469).	5
Revision 3 (February 2012)	Removed ".zip" extension in the Design files link (SAR 36763).	7
Revision 2 (January 2012)	Modified the "Configuring Hardware" section (SAR 35790).	2
	Modified the "Customizing Software" section (SAR 35790).	3
	Modified the "Program the Design" section (SAR 35790).	5
	Modified the "Running the Application" section (SAR 35790).	5
	Added section called "Release Mode" (SAR 35790).	7
	Modified the "Appendix A – Design Files" section (SAR 35790).	7
Revision 1 (May 2011)	Modified "Conclusion" and "Appendix A – Design Files" sections.	7

*Note: *The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.*

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.

51900220-4/01.13