

Using the Analog Front End in the SAM V7/E7/S7 MCUs

APPLICATION NOTE

Description

This application note explains the use of the Analog Front-End Controller (AFEC) implemented in the Atmel® | SMART™ SAMV71/V70/E70/S70 microcontrollers. It gives a brief description of the characteristics of the AFEC module, and provides an example that gives a general overview of the AFEC.

Features

- 12-bit resolution, with up to 16-bit resolution by digital averaging
- 2 MHz Conversion Rate
- Wide Range of Power Supply Operation
- Selectable Single-ended or Differential Input Voltage
- Selectable Single or Dual Sample-and-Hold Mode
- Programmable Gain for Maximum Full-Scale Input Range 0–VDD
- Programmable Offset Per Channel
- Automatic correction of offset and gain errors
- Integrated Multiplexers Offering Up to 12 Independent Analog Inputs
- Hardware or Software Trigger
 - External trigger pin
 - Timer counter outputs (corresponding TIOA trigger)
 - PWM event line
- DMA Support
- Possibility of AFE Timings Configuration
- Two Sleep Modes and Conversion Sequencer
 - Automatic wake-up on trigger and back to sleep mode after conversions of all enabled channels
 - Possibility of customized channel sequence
- Standby Mode for Fast Wake-up Time Response
 - Power-down capability
- Automatic Window Comparison of Converted Values

Table of Contents

Description.....	1
Features.....	1
1. Prerequisites.....	3
1.1. SAM V71 Xplained Ultra Evaluation Kit.....	3
1.2. SAM V71 Xplained Ultra Software Package.....	4
1.3. Atmel Studio 6.....	4
2. AFEC Software Package and User Interface.....	5
2.1. AFEC Software Package Library Introduction.....	5
2.2. Add the AFEC Library Files to the Project.....	9
3. AFEC Module Introduction.....	12
3.1. Functional Description.....	12
3.2. Getting Started with the AFEC.....	13
4. How Use the AFEC.....	19
4.1. Programming Sequence.....	19
5. Application Example.....	25
5.1. AFEC Main Configuration.....	26
5.2. Terminal Window Configuration.....	26
5.3. Bench Setup.....	26
5.4. Results.....	27
6. Reference Documents.....	33
7. Appendix: Example Application Source Code.....	34
8. Revision History.....	42

1. Prerequisites

1.1. SAM V71 Xplained Ultra Evaluation Kit

The Atmel SAM V71 Xplained Ultra evaluation kit is ideal for evaluating and prototyping with the Atmel SAM V71, SAM V70, SAM S70 and SAM E70 ARM® Cortex®-M7 based microcontrollers. The ATSAMV71-XULT evaluation kit does not include extension boards; these can be purchased individually.

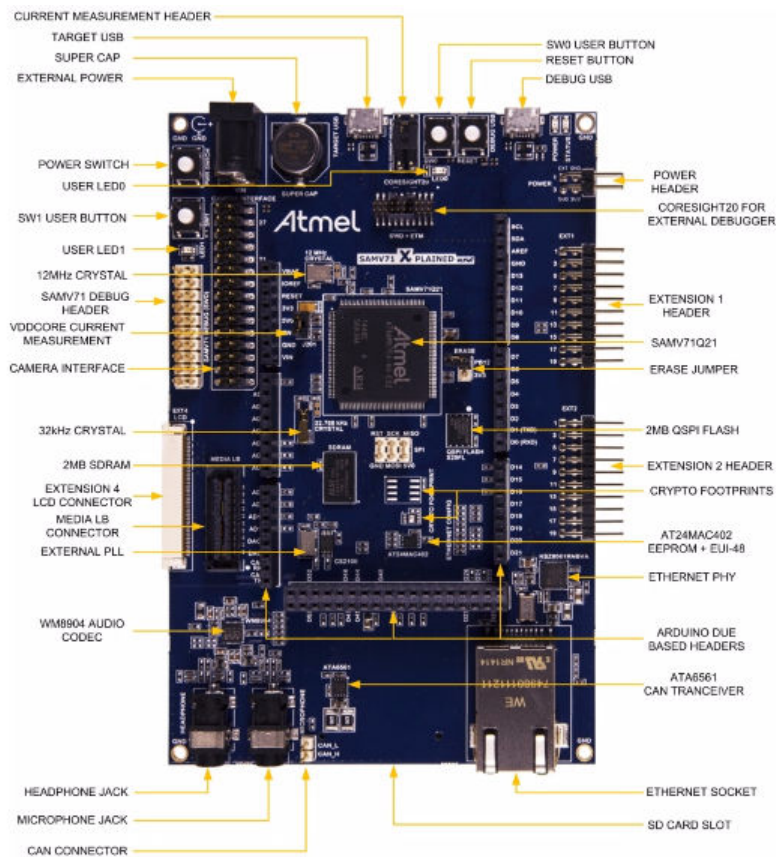


Important: For more information, visit the dedicated web page for the Atmel SAM V71 Xplained Ultra Evaluation Kit: <http://www.atmel.com/tools/ATSAMV71-XULT.aspx?tab=overview>



Important: Download the related schematics used in this application note: http://www.atmel.com/images/Atmel-42408-SAMV71-Xplained-Ultra_User-Guide.zip

Figure 1-1 SAM V71 Xplained Ultra Evaluation Kit



1.2. SAM V71 Xplained Ultra Software Package

The SAM V71 Xplained Ultra software package provides basic drivers, software services and libraries for Atmel SAMV71/V70/E70/S70 microcontrollers. It contains full source code, usage examples, documentation, and ready-to-use projects for Atmel Studio, GNU, IAR EWARM, and Keil MDK.



Important: In this application note, we use the Atmel Studio Software Package available here: <http://www.atmel.com/System/BaseForm.aspx?target=tcm:26-66028>

The AFEC software library is described later in this application note.

1.3. Atmel Studio 6



Atmel Studio 6 is the integrated development platform (IDP) for developing and debugging Atmel ARM Cortex-M processor-based and Atmel AVR[®] microcontroller applications. The Atmel Studio 6 IDP gives you a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code. Atmel Studio 6 supports all 8- and 32-bit AVR, the new SoC wireless family, SAM3 and SAM4 microcontrollers, and connects seamlessly to Atmel debuggers and development kits.

Download link: http://www.atmel.com/microsite/atmel_studio6/

Additionally, the IDP now includes two new features designed to further enhance your productivity: Atmel Gallery and Atmel Spaces.

- Atmel Gallery is an online apps store built in to Studio 6, allowing you to purchase both in-house and third-party development tools and embedded software.
Link: <http://gallery.atmel.com/>
- Atmel Spaces is a collaborative workspace where you can securely share embedded design and track progress of projects with your peers.
Link: <http://spaces.atmel.com/gf/>



Important: The Atmel Studio version used in this application note is Atmel Studio 6.2 with Service Pack 2.

2. AFEC Software Package and User Interface

2.1. AFEC Software Package Library Introduction

The software package provides a complete library for use with the AFEC.



Important: The Software Packages for Atmel Studio/IAR/Keil/GNU are available here: <http://www.atmel.com/tools/samv71-samv70-same70-sams70-software-package.aspx>

To use the AFEC, six related files are required:

- ...\\samv71_Xplained_Ultra_softpack_1.3\\libraries\\libchip_samv7\\include\\afec.h
- ...\\samv71_Xplained_Ultra_softpack_1.3\\libraries\\libchip_samv7\\source\\afec.c
- ...\\samv71_Xplained_Ultra_softpack_1.3\\Atmel\\samv71_Xplained_Ultra\\libraries\\libchip_samv7\\include\\samv7\\component\\component_afec.h
- ...\\samv71_Xplained_Ultra_softpack_1.3\\Atmel\\Atmel\\samv71_Xplained_Ultra\\libraries\\libboard_samv7-ek\\board.h
- ...\\samv71_Xplained_Ultra_softpack_1.3\\libraries\\libchip_samv7\\source\\afe_dma.c
- ...\\samv71_Xplained_Ultra_softpack_1.3\\libraries\\libchip_samv7\\include\\afe_dma.h

There are two examples available for the AFEC:

- ...\\samv71_Xplained_Ultra_softpack_1.3\\examples\\afe12_dma
- ...\\samv71_Xplained_Ultra_softpack_1.3\\examples\\afe_temp_sensor



Tip: In this application note, Atmel Studio 6 will be used as the Integrated Development Environment and the “getting_started” example is used as a starting point to make the full implementation from scratch, but any example can be used.



Caution: The getting-started example is used as a basis to implement a user program. It is advised to clear the main.c file by removing all the functions used for the getting-started implementation in order to restart from a clean project.

2.1.1. Overview of AFEC Functions

The table below lists the main functions used by the AFEC.

Function Name	Description	Location	Impacted Registers
void AFEC_Initialize arguments: Afec* pAFEC; uint32_t dwId	Initialize the Power Management Controller to enable the AFEC clock	afec.c	PMC_PCSR0 (ID 29), PMC_PCSR1 (ID 40), AFEC_CR, AFEC_MR
uint32_t AFEC_SetClock arguments: Afec* pAFEC; uint32_t dwPres; uint32_t dwMck	Configure the PRESCAL value of the AFEC_MR register to get the required AFEC frequency	afec.c	AFEC_MR
void AFEC_SetTiming arguments: Afec* pAFEC; uint32_t dwStartup; uint32_t dwTracking; uint32_t wSettling	Configure the TRANSFER, STARTUP and TRACKTIM bits of the AFEC_MR register	afec.c	AFEC_MR
void AFEC_SetTrigger arguments: Afec* pAFEC; uint32_t dwTrgSel	Configure the trigger source	afec.c	AFEC_MR
void AFEC_SetAnalogChange arguments: Afec* pAFEC; uint8_t bEnDis	If enabled, it allows different analog settings for each channel, Otherwise, DIFF0, GAIN0 and OFF0 are used for all channels.	afec.c	
void AFEC_SetSleepMode arguments: Afec* pAFEC; uint8_t bEnDis	Enable/Disable Sleep mode	afec.c	AFEC_MR
void AFEC_SetFastWakeup arguments: Afec* pAFEC; uint8_t bEnDis	Enable/Disable Fast Wake-up mode	afec.c	AFEC_MR
void AFEC_SetSequenceMode arguments: Afec* pAFEC; uint8_t bEnDis	Enable/Disable Sequence mode	afec.c	AFEC_MR
void AFEC_SetSequence arguments: Afec* pAFEC; uint32_t dwSEQ1 ; uint32_t dwSEQ2	Set channel sequence.	afec.c	AFEC_SEQ1R AFEC_SEQ2R
void AFEC_SetSequenceByList arguments: Afec* pAFEC; uint8_t ucChList[] ; uint8_t ucNumCh	Set sequence by list	afec.c	AFEC_SEQ1R AFEC_SEQ2R
void AFEC_SetTagEnable arguments: Afec* pAFEC; uint8_t bEnDis	Set "TAG" mode, show channel number in last data or not.	afec.c	AFEC_EMR

Function Name	Description	Location	Impacted Registers
void AFEC_SetCompareChannel arguments: Afec* pAFEC; uint32_t dwChannel	Set compare channel	afec.c	AFEC_EMR
void AFEC_SetCompareMode arguments: Afec* pAFEC; uint32_t dwMode	Enable/Disable Compare mode	afec.c	AFEC_EMR
void AFEC_SetComparisonWindow arguments: Afec* pAFEC; uint32_t dwHi	Set comparison values	afec.c	AFEC_CWR
uint32_t AFEC_GetConvertedData arguments: Afec* pAFEC; uint32_t dwChannel	Get the last converted data	afec.c	AFEC_CSELR
void AFEC_SetStartupTime arguments: Afec* pAFEC; uint32_t dwUs	Set the start-up time	afec.c	AFEC_MR
void AFEC_SetTrackingTime arguments: Afec* pAFEC; uint32_t dwNs	Set the tracking time	afec.c	AFEC_MR
void AFEC_SetAnalogOffset arguments: Afec* pAFEC; uint32_t dwChannel ; uint32_t aoffset	Set the analog offset	afec.c	AFEC_CSELR AFEC_COCR
void AFEC_SetAnalogControl arguments: Afec* pAFEC ; uint32_t control	Set analog offset to be used for channel CSEL	afec.c	AFEC_ACR
uint32_t Afe_ConfigureDma arguments: AfeDma *pAfed ; Afec *pAfeHw ; uint8_t Afeld; sXdmad *pXdmad	Configure the DMA to work with AFEC	afe_dma.c	
uint32_t Afe_SendData arguments: AfeDma *pAfed ; AfeCmd *pCommand	Send AFEC data with the DMA	afe_dma.c	

2.1.2. Software Package AFEC Macro Definitions

The AFEC macro definitions from the software package are provided below.

```

/
*-----
----
*           Macros function of register access
*-----
----*/

```

```

#define AFEC_GetModeReg( pAFEC )                ((pAFEC)->AFEC_MR)
#define AFEC_SetModeReg( pAFEC, mode )          ((pAFEC)->AFEC_MR = mode)

#define AFEC_GetExtModeReg( pAFEC )             ((pAFEC)->AFEC_EMR)
#define AFEC_SetExtModeReg( pAFEC, mode )       ((pAFEC)->AFEC_EMR = mode)

#define AFEC_StartConversion( pAFEC )           ((pAFEC)->AFEC_CR =
AFEC_CR_START)

#define AFEC_EnableChannel( pAFEC, dwChannel )  {\
    (pAFEC)->AFEC_CHER = (1 << (dwChannel));\
}

#define AFEC_DisableChannel(pAFEC, dwChannel) {\
    (pAFEC)->AFEC_CHDR = (1 << (dwChannel));\
}

#define AFEC_EnableIt(pAFEC, dwMode)           {\
    (pAFEC)->AFEC_IER = (dwMode);\
}

#define AFEC_DisableIt(pAFEC, dwMode)          {\
    (pAFEC)->AFEC_IDR = (dwMode);\
}

#define AFEC_SetChannelGain(pAFEC,dwMode)      {\#define
AFEC_SetChannelGain(pAFEC,dwMode)              {\
}

#define AFEC_GetModeReg( pAFEC )                ((pAFEC)->AFEC_MR)
#define AFEC_SetModeReg( pAFEC, mode )          ((pAFEC)->AFEC_MR = mode)

#define AFEC_GetExtModeReg( pAFEC )             ((pAFEC)->AFEC_EMR)
#define AFEC_SetExtModeReg( pAFEC, mode )       ((pAFEC)->AFEC_EMR = mode)

#define AFEC_StartConversion( pAFEC )           ((pAFEC)->AFEC_CR =
AFEC_CR_START)

#define AFEC_EnableChannel( pAFEC, dwChannel )  {\
    (pAFEC)->AFEC_CHER = (1 << (dwChannel));\
}

#define AFEC_DisableChannel(pAFEC, dwChannel) {\
    (pAFEC)->AFEC_CHDR = (1 << (dwChannel));\
}

#define AFEC_EnableIt(pAFEC, dwMode)           {\
    (pAFEC)->AFEC_IER = (dwMode);\
}

#define AFEC_DisableIt(pAFEC, dwMode)          {\
    (pAFEC)->AFEC_IDR = (dwMode);\
}

#define AFEC_SetChannelGain(pAFEC,dwMode)      {\
    (pAFEC)->AFEC_CGR = dwMode;\
}

#define AFEC_EnableDataReadyIt(pAFEC)          ((pAFEC)->AFEC_IER =
AFEC_IER_DRDY)

```

```
#define AFEC_GetStatus(pAFEC) ((pAFEC)->AFEC_ISR)

#define AFEC_GetCompareMode(pAFEC) ((pAFEC)->AFEC_EMR) &
(AFEC_EMR_CMPMODE_Msk)

#define AFEC_GetChannelStatus(pAFEC) ((pAFEC)->AFEC_CHSR)

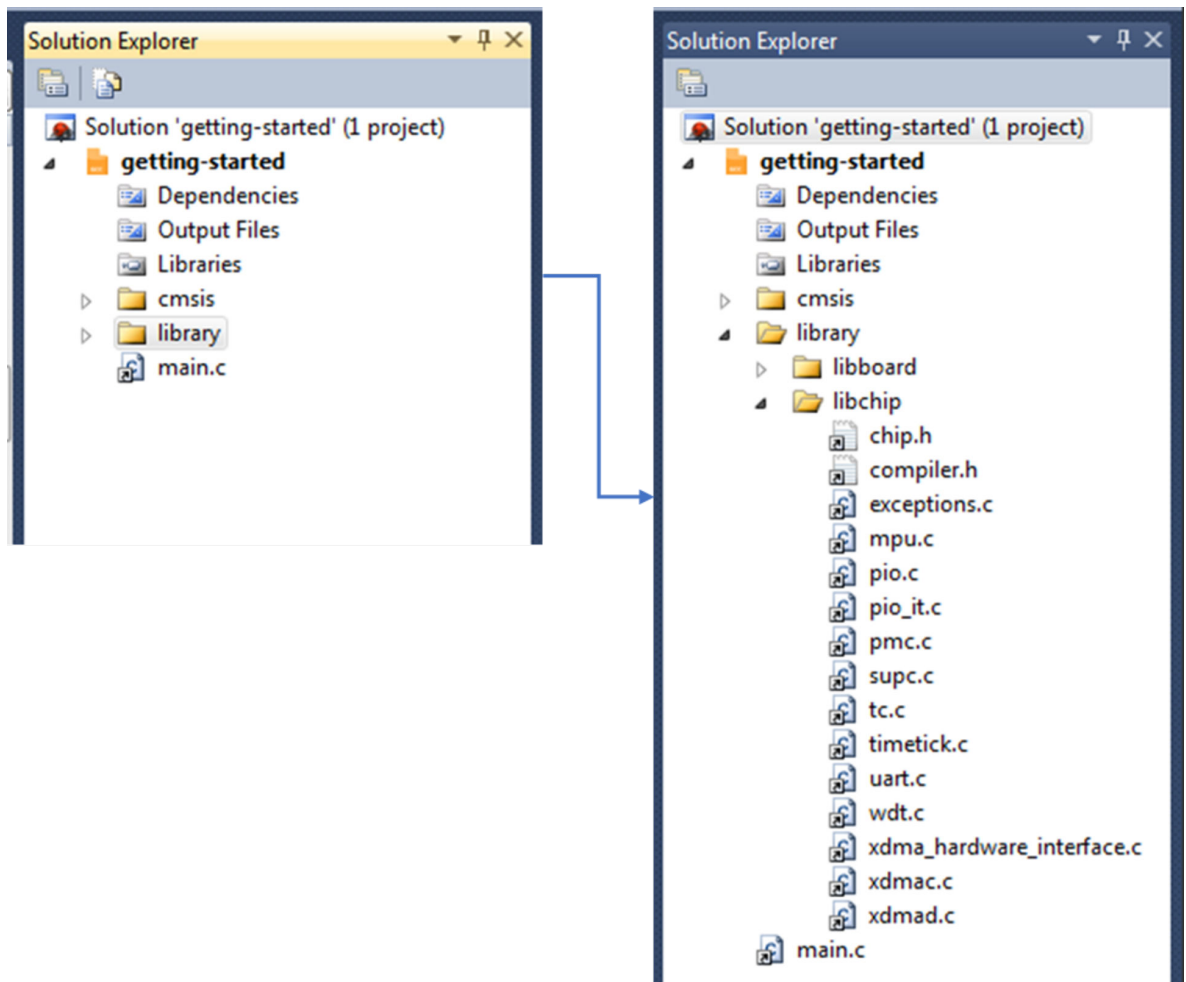
#define AFEC_GetInterruptMaskStatus(pAFEC) ((pAFEC)->AFEC_IMR)

#define AFEC_GetLastConvertedData(pAFEC) ((pAFEC)->AFEC_LCDR)
```

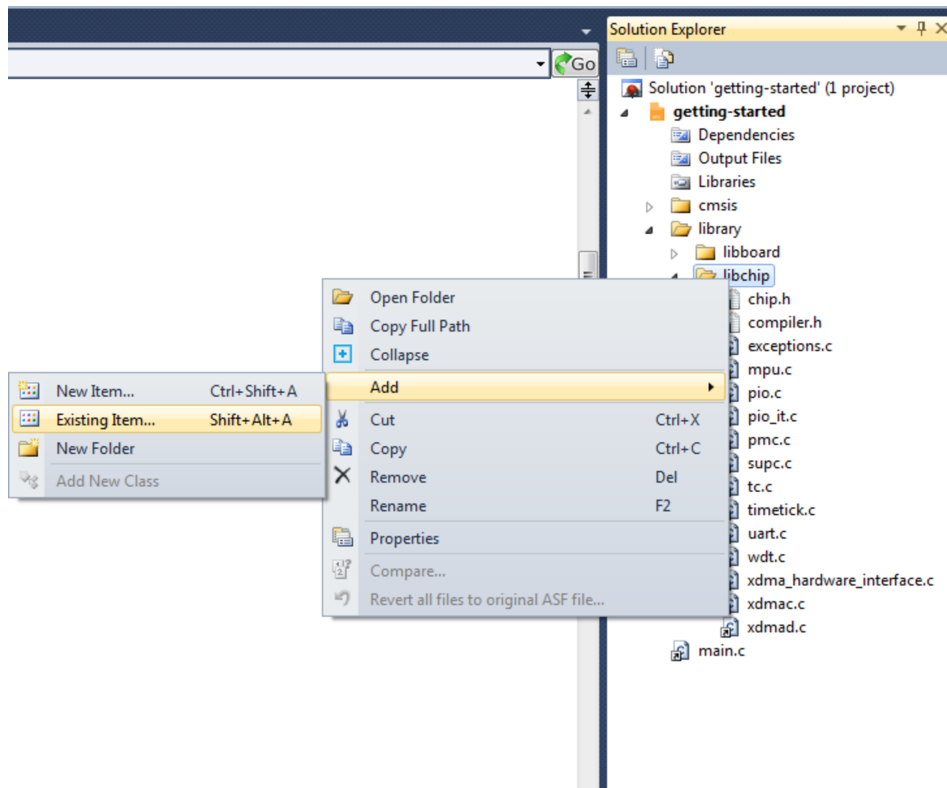
2.2. Add the AFEC Library Files to the Project

The `afec.c` and `afec_dma.c` files must be added to the project in order to use the getting-started example as a basis for the implementation. To add these files, follow the steps below:

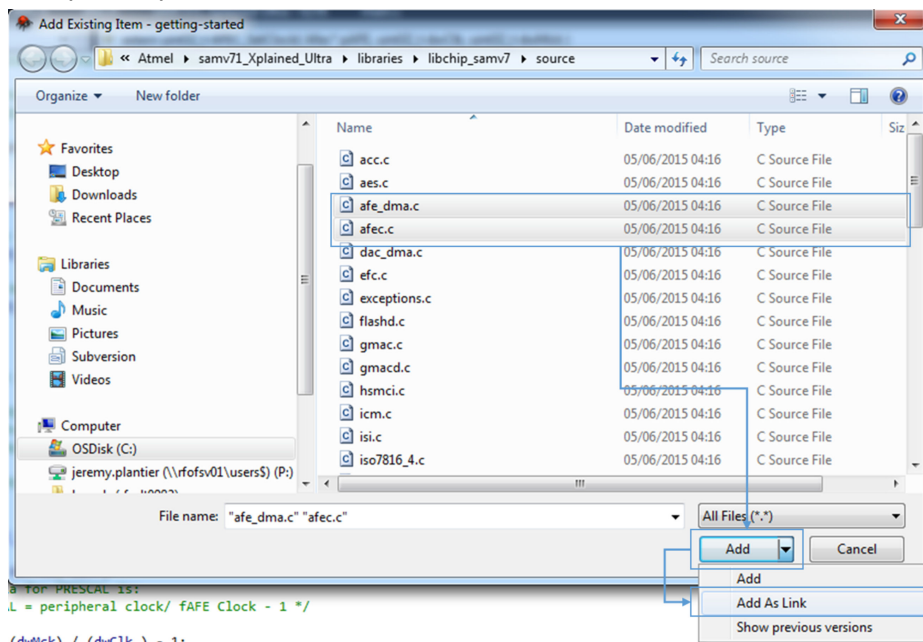
1. Open the getting-started example by opening the `getting-started.atsln` file located in the ..
`\SAMV71_softpack_1.3_for_gnu_arm\Atmel\samv71_Xplained_Ultra\examples\getting-started\build\studio` directory. Atmel Studio opens and the getting-started example is launched.
2. From the Solution Explorer window, deploy the libchip library directory:



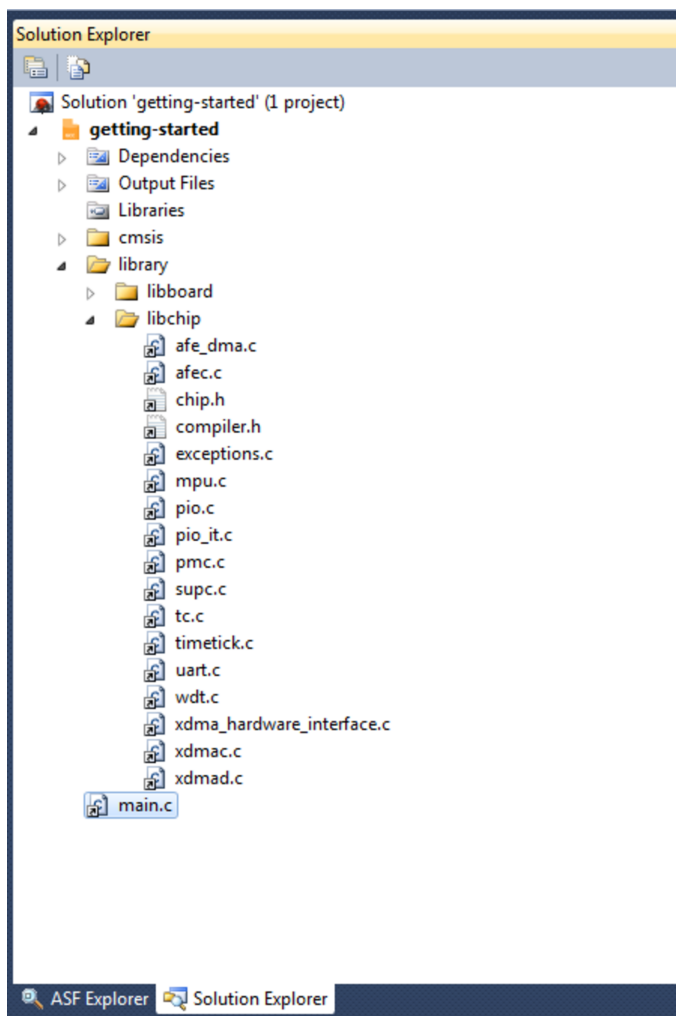
3. Add an existing item to the project by right-clicking on the libchip directory and selecting Add>Existing Item...



4. Select the **afec.c** and the **afe_dma.c** files in `\SAMV71_softpack_1.3_for_gnu_arm\Atmel\samv71_Xplained_Ultra\libraries\libchip_samv7\include` and choose the option "Add as Link" from the open drop-down menu:



5. The files have now been added and are visible from the Solution Explorer window:



3. AFEC Module Introduction

3.1. Functional Description

The Analog Front-End Controller (AFEC) is based on an Analog Front-End cell (AFE) integrating a 12-bit Analog-to-Digital Converter (ADC), a Programmable Gain Amplifier (PGA), a Digital-to-Analog Converter (DAC) and a 12-to-1 analog multiplexer, making possible the analog-to-digital conversions of 12 analog lines. two 6-to-1 analog multiplexers, making possible the analog-to-digital conversions of 12 analog lines (in single Sample-and-Hold mode) or two simultaneous conversions of 6 analog lines (in dual Sample-and-Hold mode). The conversions extend from 0V to ADVREFP. The AFEC supports an 10-bit or 12-bit resolution mode which can be extended up to a 16-bit resolution by digital averaging.

Conversion results are reported in a common register for all channels, as well as in a channel-dedicated register.

Software trigger, external trigger on rising edge of the AFE_ADTRG pin or internal triggers from Timer Counter output(s) are configurable.

The comparison circuitry allows automatic detection of values below a threshold, higher than a threshold, in a given range or outside the range. Thresholds and ranges are fully configurable.

The AFEC internal fault output is directly connected to PWM Fault input. This input can be asserted by means of comparison circuitry in order to immediately put the PWM outputs in a safe state (pure combinational path).

The AFEC also integrates a Sleep mode and a conversion sequencer and connects with a DMA channel. These features reduce both power consumption and processor intervention.

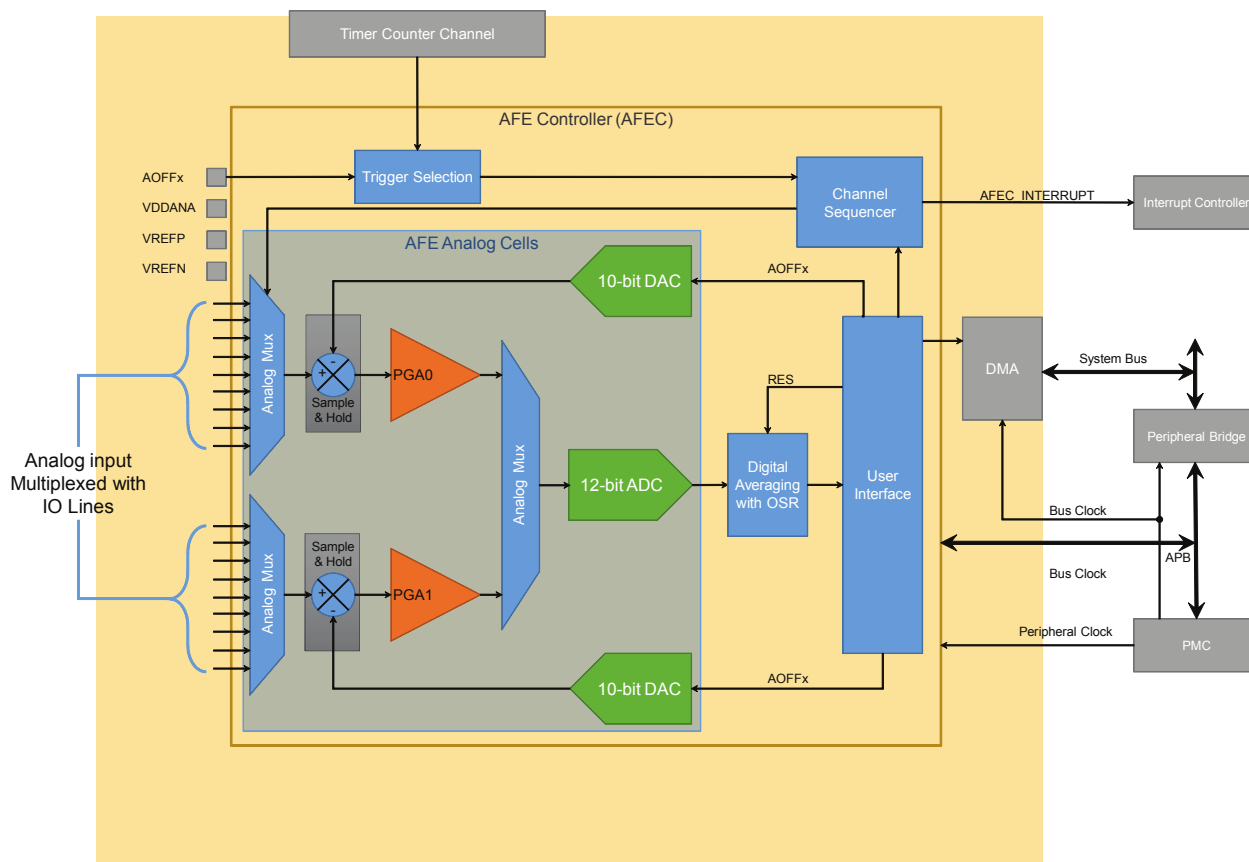
The AFEC has a selectable single-ended or fully differential input and benefits from a 2-bit programmable gain. A set of reference voltages is generated internally from a single external reference voltage node that may be equal to the analog supply voltage.

The AFEC is powered by the analog power domain (VDDANA) and VREFP is the positive reference of the AFE. The VREFN pin must be connected to ground.

DAC1 and DAC0 provide an analog output voltage (VDAC) in the range [0 : VREFP] with an accuracy equal to 10 bits. The DAC output voltage is single-ended and is used as a reference node by the sampling stage S/H0 and S/H1 (Sample-and-Hold PGA), relative to the single-ended input signal being sampled on the selected channel.

As a consequence, programming the DAC output voltage offers a capability to compensate for a DC offset on the input signal being sampled. DC offset compensation is effective in single-ended operation and is not effective in fully differential operation.

Figure 3-1 AFEC Block Diagram



3.2. Getting Started with the AFEC

To get started with the AFEC, some registers must be configured. Only some of the AFEC registers are described in this section. For descriptions of all AFEC registers, refer to the product datasheets.

3.2.1. Enabling the AFEC

3.2.1.1. PMC and AFEC Control Register

The AFEC Control register (AFEC_CR) enables the AFEC at the cell level. Before programming the AFEC_CR to enable the AFEC, the AFEC must be enabled from the Power Management Controller (PMC).

To enable the AFEC, the user program must perform the following steps:

1. Enable the AFEC in the PMC Peripheral Clock Enable register (PMC_PCER0) using the AFEC PID (29).
2. Perform an AFEC reset by setting the SWRST bit of the AFEC_CR.
3. Clear the AFEC Mode register (AFEC_MR).

Code example for the AFEC0:

```
/* Enabling the AFEC in the PMC */
PMC->PMC_PCER0 = 1 << AFEC_ID

/* Reset the controller */
AFEC0->AFEC_CR = AFEC_CR_SWRST;
```

```

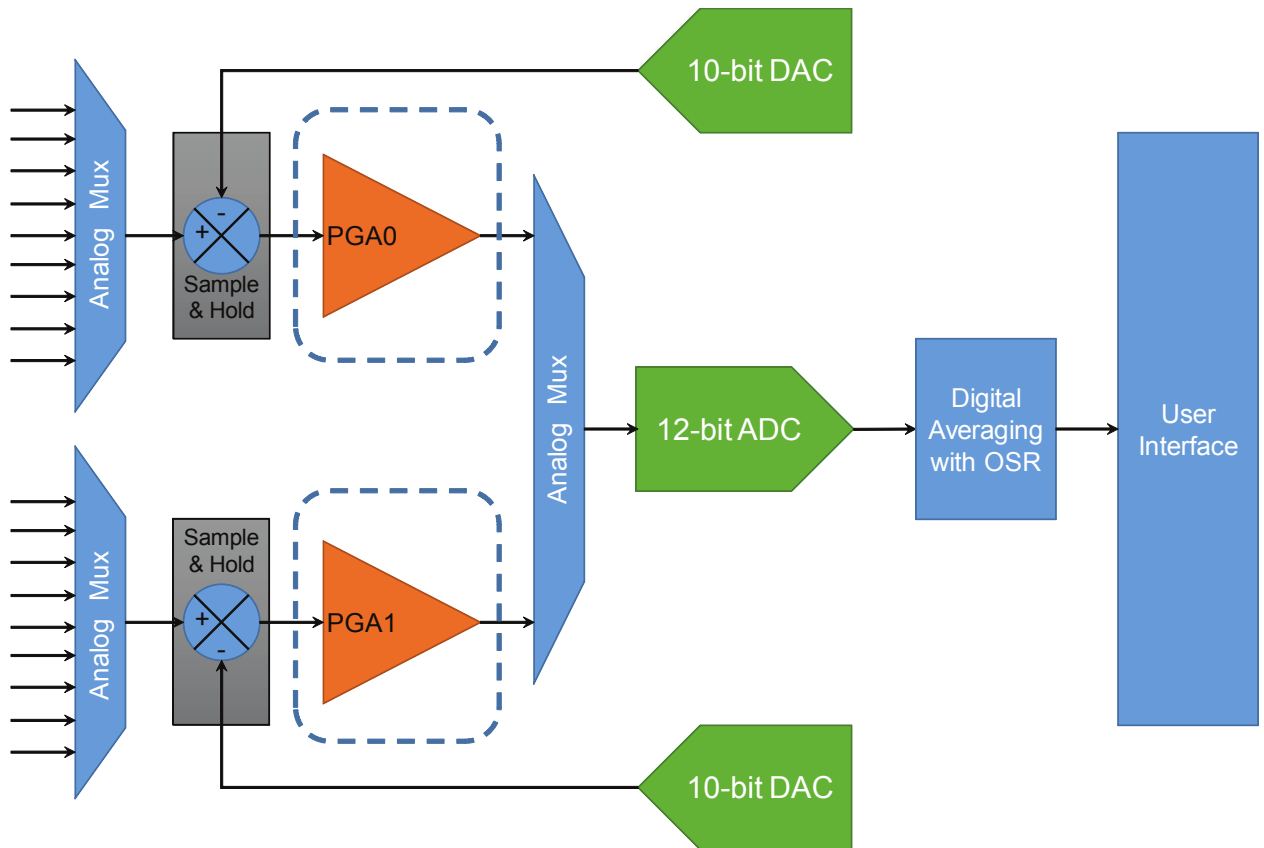
/* Reset Mode Register */
AFEC0->AFEC_MR = 0;

```

3.2.1.2. Enable the Programmable Gain Amplifier

The AFE embeds programmable gain amplifiers that must be enabled prior to any conversion. The bits PGA0EN and PGA1EN in the Analog Control register (AFEC_ACR) must be set.

Figure 3-2 AFE Analog Cell Block Diagram



3.2.2. Configuring the AFEC Mode Register

To ensure correct functionality of the AFEC, the AFEC Mode register (AFEC_MR) must be configured as described in the following sections.

Figure 3-3 Detailed View of the AFEC_MR



3.2.2.1. Tracking, Transfer and Startup Timings Value

The AFE uses the AFE clock to perform conversions. Converting a single analog value to a 12-bit digital data requires tracking clock cycles as defined in the field TRACKTIM, and transfer clock cycles as defined in the field TRANSFER of the AFEC_MR. The tracking phase starts during the conversion of the previous channel.

The AFE conversion time (t_{AFE_conv}) is applicable for all modes and depends on the values of AFEC_MR.TRACKTIM and ADC_MR.TRANSFER. If TRACKTIM > 14, the conversion time is calculated as:

$$t_{AFE_conv} = (TRANSFER + 6) + (TRACKTIM + 1) \times t_{AFE \text{ Clock}}$$

If TRACKTIM ≤ 14, the conversion time is calculated as:

$$t_{AFE_conv} = ((TRANSFER + 6) + 15) \times t_{AFE \text{ Clock}}$$

If the tracking time is longer than the conversion time of the 12-bit AD converter (t_{AFE_CONV}), the tracking phase is extended to the end of the previous conversion.

The timing diagrams below provide information on the timings of the AFEC:

Figure 3-4 Sequence of AFE Conversions when Tracking Time > Conversion Time

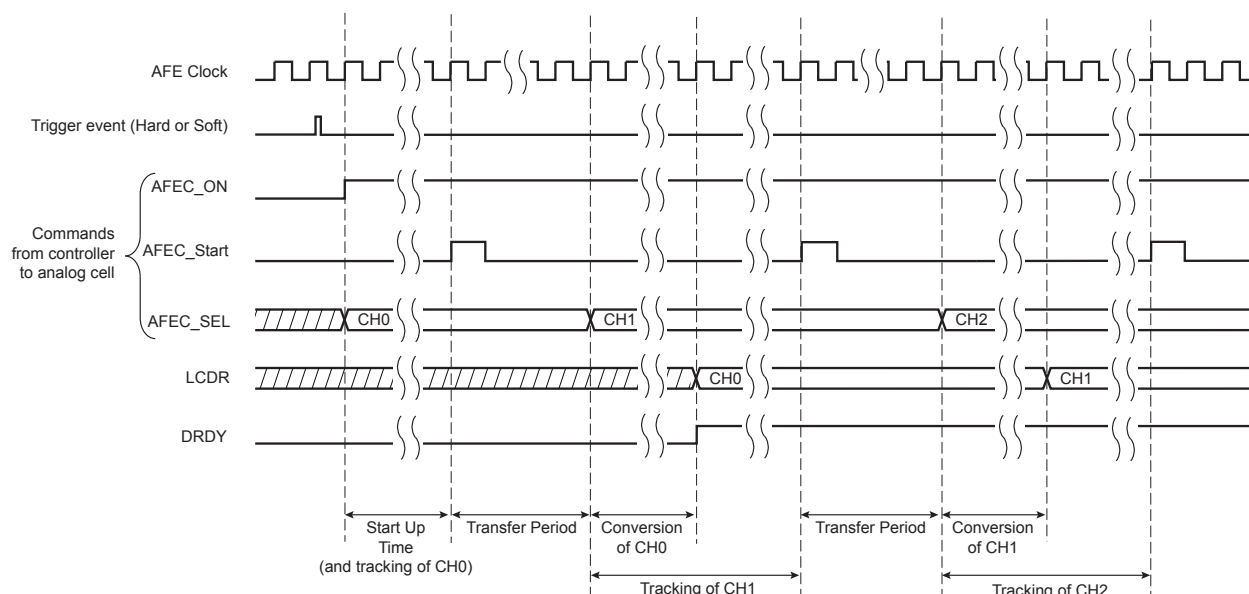
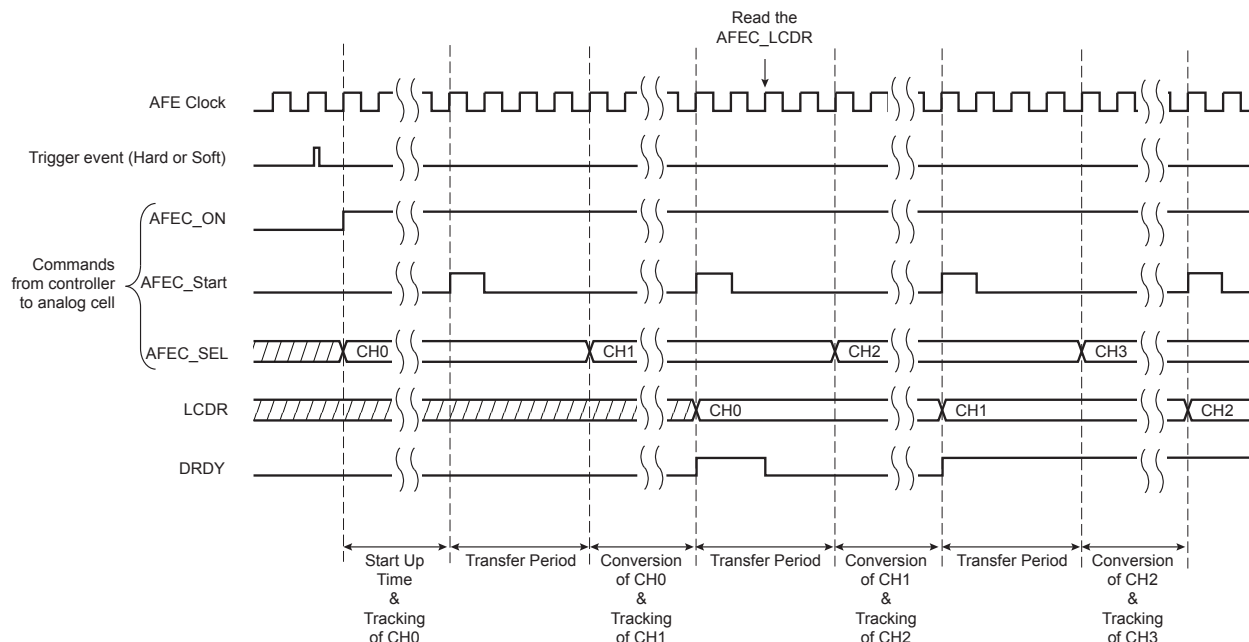


Figure 3-5 Sequence of AFE Conversions when Tracking Time < Conversion Time



3.2.2.2. AFEC Clock and PRESCAL

The AFE clock frequency is selected in the PRESCAL field of the AFEC_MR.

PRESCAL must be programmed to provide the AFE clock frequency. As given in the section “Electrical Characteristics” of the product datasheet, the AFE is able to run at up to 40 MHz for maximum of 2 MSps bit-rate.

The AFE clock frequency ranges from $f_{\text{peripheral clock}}/2$ if PRESCAL is 1, and $f_{\text{peripheral clock}}/512$ if PRESCAL is set to 255 (0xFF).

- PRESCAL: Prescaler Rate Selection**

$$\text{PRESCAL} = (f_{\text{peripheral clock}} / f_{\text{AFE Clock}}) - 1$$

When PRESCAL is cleared, no conversion is performed.

Code example to set the PRESCAL value:

```
/**
 * \brief Set AFE clock.
 *
 * \param pAFE Pointer to an AFE instance.
 * \param dwPres prescale value
 * \param dwMck Board MCK (Hz)
 *
 * \return AFE clock
 */

extern uint32_t AFEC_SetClock( Afec* pAFE, uint32_t dwClk, uint32_t dwMck )
{
    uint32_t dwPres, dwMr;
    /* Formula for PRESCAL is: PRESCAL = peripheral clock/ fAFE Clock - 1
    */

    dwPres = (dwMck) / (dwClk ) - 1;
    dwMr = AFEC_MR_PRESCAL(dwPres);
    if (dwMr == 0) return 0;

    dwMr |= (pAFE->AFEC_MR & ~AFEC_MR_PRESCAL_Msk);
    pAFE->AFEC_MR = dwMr;
    dwAFEClock = dwMck / (dwPres + 1);
    return dwAFEClock;
}
```

3.2.2.3. Additional AFEC_MR Bits

The following bits must be programmed in AFEC_MR:

- USEQ: User Sequence Enable
 - 0: NUM_ORDER: Normal mode: The controller converts channels in a simple numeric order.
 - 1: REG_ORDER: User Sequence mode: The sequence respects what is defined in AFEC_SEQ1R and AFEC_SEQ1R.
- ONE: This bit must be written to 1.
- FREERUN: Free Run Mode Selection
 - 0: OFF: Normal mode
 - 1: ON: Free Run mode: Never wait for any trigger. In Free Run mode, the sampling frequency (f_s) is calculated as $1/t_{AFE_conv}$.
- FWUP: Fast Wake-up Selection
 - 0: OFF: Normal Sleep mode: The sleep mode is defined by the SLEEP bit.
 - 1: ON: Fast wake-up Sleep mode: The voltage reference is ON between conversions and AFE is OFF.
- SLEEP: Sleep Mode Selection
 - 0: Normal mode: The AFE and reference voltage circuitry are kept ON between conversions.
 - 1: Sleep mode: The AFE and reference voltage circuitry are OFF between conversions.
- TRGEN: Hardware triggers are enabled or disabled using this bit.
 - 0: Starting a conversion is only possible by software.
 - 1: Hardware trigger selected by TRGSEL field is enabled.
- TRGSEL: Trigger Source Selection (ADTRG, Timer Counter, PWM, Analog Comparator)

Code example to configure AFEC_MR:

```
/* Step 3: AFEC Mode Register Configuration */
AFEC_SetModeReg(
AFEC0, AFEC_MR_FREERUN_OFF | AFEC_MR_TRANSFER(1) | AFEC_MR_TRACKTIM(2) |
AFEC_MR_ONEN | AFEC_MR_STARTUP_SUT512);
```

3.2.3. Starting a Conversion

The type of trigger selected with AFEC_MR.TRGEN determines when a conversion starts.

If a hardware (external) trigger is selected (AFEC_MR.TRGEN = 1), the conversion starts automatically after the trigger event programmed in AFEC_MR.TRGSEL.

Code example to select the trigger source:

1. Set AFEC_MR.TRGEN = 1:

```
/* Step 3: AFEC Mode Register Configuration */
AFEC_SetModeReg(
AFEC0, AFEC_MR_FREERUN_OFF | AFEC_MR_TRANSFER(1) | AFEC_MR_TRACKTIM(2) |
AFEC_MR_ONEN | AFEC_MR_STARTUP_SUT512 | AFEC_MR_TRGEN_EN);
```

2. Select the trigger source:

```
/**
 * \brief Set AFE trigger.
 *
 * \param pAFE Pointer to an AFE instance.
 * \param dwTrgSel Trigger selection
 */
extern void AFEC_SetTrigger( Afec* pAFE, uint32_t dwTrgSel )
{
    uint32_t dwMr;

    dwMr = pAFE->AFEC_MR;
    dwMr &= ~AFEC_MR_TRGSEL_Msk;
    dwMr |= dwTrgSel;
    pAFE->AFEC_MR |= dwMr;
}
```

If a software trigger is selected, (AFEC_MR.TRGEN = 0), the conversion starts immediately after setting the START bit of the AFEC Control Register (AFEC_CR).

Code example to start a software conversion:

```
/* Start Conversion by software trigger */
AFEC_StartConversion(AFEC0);
```

4. How Use the AFEC

The steps to configure the AFEC are listed below. They are given in more detail in the following section.

1. Configure the GPIO for AFEC. (If not already done by default).
2. Initialize the AFEC with AFEC_Initialize(). (PMC configuration).
3. Configure the AFEC Mode Register.
4. Set AFEC clock with AFEC_SetClock().
5. Select the active channel using AFEC_EnableChannel().
6. Adjust the Channel Offset Correction to get a good single-ended result form.
7. Enable the PGA0 and PGA1 of the AFEC0 and adjust the performances by tuning the AFE Bias Current Control.
8. Configure the Extended Mode Register (AFEC_EMR) by enabling/disabling averaging, enabling the channel tag and using the Single Trigger mode.
9. Start the conversion with AFEC_StartConversion().
10. Wait for the end of the conversion by polling the status with AFEC_GetStatus().
11. Get the converted data using AFEC_GetConvertedData() or AFEC_GetLastConvertedData().

The implementation can directly start from the **main.c** file just before the main function.

The first function to implement is the AFEC initialization function:

```
/**
 * \brief Initialize AFE.
 */
static void _afe_initialization(void)
{

}

/
*-----
--
*           Exported functions
*-----
--*/
/**
 * \brief getting-started Application entry point.
 * \return Unused (ANSI-C compatibility).
 */
extern int main( void )
{

    /* Disable watchdog */
    WDT_Disable( WDT ) ;
    ...
}
```

4.1. Programming Sequence

1. Configure the pins for the AFEC.
By default, Channel 5 of the AFEC0 is configured as analog input and, as a result, does not require further configuration.

For a better readability, a local definition related to the channel used can be added as follows:

```

/
*-----
-----
*           Local definitions
*-----
-----*/

/** First AFE Channel used*/
#define TEST_CHANNEL    5

```

2. Initialize the AFEC with AFEC_Initialize().

```

/**
 * \brief Initialize AFE.
 *
 */
static void _afe_initialization(void)
{
    /* Step 1: Configure the pins for AFEC. (If not already done by
    default) */
    /* Step 2: AFEC init */
    AFEC_Initialize( AFEC0, ID_AFEC0 );
}

```

3. Configure the AFEC_MR.

By using the macro "AFEC_SetExtModeReg(pAFEC, mode)" defined in the afec.h file, configure the AFEC_MR as follows:

- Free Run mode disabled
- Transfer period = 1, where TRANSFER = (Transfer period / AFE clock periods) – 6
- Tracking time = 2, where TRACKTIM = (Tracking time / AFE clock periods) – 1
- Conversion time = $t_{AFE_CONV} = (6 + 1 + 15) \times t_{AFE_Clock}$
- AFEC_MR.ONE = 1
- Start-up time = 64 (64 periods of AFE clock)

In this example, we chose to set the tracking time to the maximum value (15). For more details, refer to the AFE Timings in the Electrical Characteristics section of the product datasheet.

```

/**
 * \brief Initialize AFE.
 *
 */
static void _afe_initialization(void)
{
    /* Step 1: Configure the pins for AFEC. (If not already done by
    default) */
    /* Step 2: AFEC init */
    AFEC_Initialize( AFEC0, ID_AFEC0 );

    /*Step 3: AFEC Mode Register Configuration */
    AFEC_SetModeReg(AFEC0, AFEC_MR_FREERUN_OFF | AFEC_MR_TRANSFER(1) |
    AFEC_MR_TRACKTIM(2) | AFEC_MR_ONE | AFEC_MR_STARTUP_SUT64);
}

```

4. Set the AFEC clock with AFEC_SetClock().

The AFEC_SetClock function requires three arguments:

- Afec* pAFEC (pointer to the AFE instance)
- uint32_t dwPres (Prescale rate selection) used as follow: $PRESCAL = f_{\text{peripheral clock}} / f_{\text{AFE Clock}} - 1$
- uint32_t dwMck (main board crystal frequency) this value is already defined in the libboard \board.h file (BOARD_MCK).

A local definition related to the AFE Clock value used can be added as follows:

```

/
*-----
-----
*          Local definitions
*-----
-----*/

/** IRQ priority for PIO (The lower the value, the greater the
priority) */
#define IRQ_PRIOR_PIO      0

/** LED0 blink time, LED1 blink half this time, in ms */
#define BLINK_PERIOD      1000

/** First AFE Channel used*/
#define TEST_CHANNEL      5

/* AFE Clock Value */
#define AFE_CLK            2200000

/**
 * \brief Initialize AFE.
 *
 */
static void _afe_initialization(void)
{
    /* Step 1: Configure the pins for AFEC. (If not already done by
    default) */
    /* Step 2: AFEC init */
    AFEC_Initialize( AFEC0, ID_AFEC0 );

    /*Step 3: AFEC Mode Register Configuration */
    AFEC_SetModeReg(AFEC0,AFEC_MR_FREERUN_ON|AFEC_MR_TRANSFER(1) |
    AFEC_MR_TRACKTIM(2) | AFEC_MR_ONE | AFEC_MR_STARTUP_SUT64);

    /* Step 4      Set AFEC clock AFEC_SetClock() */
    AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK ) ;
}

```

5. Select the active channel using the macro definition AFEC_EnableChannel().

```

/**
 * \brief Initialize AFE.
 *
 */
static void _afe_initialization(void) {

    /* Step 1: Configure the pins for AFEC. (If not already done by
    default) */
    /* Step 2: AFEC init */
    AFEC_Initialize( AFEC0, ID_AFEC0 );

```

```

    /*Step 3: AFEC Mode Register Configuration */
    AFEC_SetModeReg(AFEC0,AFEC_MR_FREERUN_ON|AFEC_MR_TRANSFER(1) |
AFEC_MR_TRACKTIM(2) | AFEC_MR_ONE | AFEC_MR_STARTUP_SUT64);

    /* Step 4      Set AFEC clock AFEC_SetClock() */
    AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK );

    /* Step 5      Select the channel using the Macro definition
AFEC_EnableChannel(). */
    AFEC_EnableChannel(AFEC0, TEST_CHANNEL);
}

```

6. Adjust the channel offset correction to get a good single-ended result form.
An offset value set in AFEC_COCR.AOFF defines the analog offset to be used for channel CSEL (configured in the AFEC_CSELR). This value is used as an input value for the DAC included in the AFE.

A local definition related to the CHANNEL OFFSET value used can be added as follows:

```

/
*-----
-----
*          Local definitions
*-----
-----*/

/** IRQ priority for PIO (The lower the value, the greater the
priority) */
#define IRQ_PRIOR_PIO      0

/** LED0 blink time, LED1 blink half this time, in ms */
#define BLINK_PERIOD      1000

/** First AFE Channel used*/
#define TEST_CHANNEL      5

/** AFE Channel DAC Offset */
#define CHANNEL_OFFSET    0x200

/**
 * \brief Initialize AFE.
 *
 */
static void _afe_initialization(void)
{

    /* Step 1: Configure the pins for AFEC. (If not already done by
default) */
    /* Step 2: AFEC init */
    AFEC_Initialize( AFEC0, ID_AFEC0 );

    /*Step 3: AFEC Mode Register Configuration */
    AFEC_SetModeReg(AFEC0,AFEC_MR_FREERUN_ON|AFEC_MR_TRANSFER(1) |
AFEC_MR_TRACKTIM(2) | AFEC_MR_ONE | AFEC_MR_STARTUP_SUT64);

    /* Step 4      Set AFEC clock AFEC_SetClock() */
    AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK );

    /* Step 5      Select the active channel using the Macro definition

```

```

AFEC_EnableChannel(). */
AFEC_EnableChannel(AFEC0, TEST_CHANNEL);

/* Step 6 Adjust the channel level offset */
AFEC_SetAnalogOffset(AFEC0, TEST_CHANNEL, CHANNEL_OFFSET);
}

```

7. Enable the PGA0 and PGA1 of the AFEC0 and adjust the performances by tuning the AFE Bias Current Control.

```

/**
 * \brief Initialize AFE.
 *
 */
static void _afe_initialization(void)
{
    /* Step 1: Configure the pins for AFEC. (If not already done by
    default) */
    /* Step 2: AFEC init */
    AFEC_Initialize( AFEC0, ID_AFEC0 );

    /*Step 3: AFEC Mode Register Configuration */
    AFEC_SetModeReg(AFEC0,AFEC_MR_FREERUN_ON|AFEC_MR_TRANSFER(1)|
    AFEC_MR_TRACKTIM(2)| AFEC_MR_ONE| AFEC_MR_STARTUP_SUT64);

    /* Step 4 Set AFEC clock AFEC_SetClock() */
    AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK );

    /* Step 5 Select the active channel using the Macro definition
    AFEC_Enable Channel(). */
    AFEC_EnableChannel(AFEC0, TEST_CHANNEL);

    /* Step 6 Adjust the channel level offset */
    AFEC_SetAnalogOffset(AFEC0, TEST_CHANNEL, CHANNEL_OFFSET);

    /* Step 7 Enable the PGA 0 and 1 of the AFEC0 and adjust the
    performances by tuning the AFE Bias Current Control */
    AFEC_SetAnalogControl(AFEC0, AFEC_ACR_IBCTL(1) | AFEC_ACR_PGA0_ON
    | AFEC_ACR_PGA1_ON );
}

```

8. Configure the Extended Mode Register (AFEC_EMR) by disabling averaging, enabling the channel tag and using the Single Trigger mode.

```

/**
 * \brief Initialize AFE.
 *
 */
static void _afe_initialization(void) {
    /* Step 1: Configure the pins for AFEC. (If not already done by
    default) */
    /* Step 2: AFEC init */
    AFEC_Initialize( AFEC0, ID_AFEC0 );

    /*Step 3: AFEC Mode Register Configuration */
    AFEC_SetModeReg(AFEC0,AFEC_MR_FREERUN_ON|AFEC_MR_TRANSFER(1)|
    AFEC_MR_TRACKTIM(2)| AFEC_MR_ONE| AFEC_MR_STARTUP_SUT64);

    /* Step 4 Set AFEC clock AFEC_SetClock() */
    AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK );
}

```

```

/* Step 5    Select the active channel using the Macro definition
AFEC_EnableChannel(). */
AFEC_EnableChannel(AFEC0, TEST_CHANNEL);

/* Step 6    Adjust the channel level offset */
AFEC_SetAnalogOffset(AFEC0, TEST_CHANNEL, CHANNEL_OFFSET);

/* Step 7 Enable the PGA 0 and 1 of the AFEC0 and adjust the
performances by tuning the AFE Bias Current Control */
AFEC_SetAnalogControl(AFEC0, AFEC_ACR_IBCTL(1) | AFEC_ACR_PGA0_ON
| AFEC_ACR_PGA1_ON );

/* Step 8 Configure the Extended Mode Register by disabling the
Averaging, enabling the channel tag and using the Single Trigger Mode
*/
AFEC_SetExtModeReg(AFEC0, 0 | AFEC_EMR_RES(256) | AFEC_EMR_TAG |
AFEC_EMR_STM );

}

```

At this point, the configuration of the AFEC is finished. The next steps in the sequence start a conversion and wait for the End Of Conversion flag. This is implemented directly inside the main function as described below.

9. Start conversion by software trigger.

```

/* Step 9: Start Conversion by software trigger */
AFEC_StartConversion(AFEC0);

```

10. Wait for the end of the conversion by polling the status with AFEC_GetStatus().

```

/* Step 10: Wait for the end of the conversion by polling status with
AFEC_GetStatus() */
while ( !(AFEC_GetStatus(AFEC0) & AFEC_ISR_EOC5));

```

11. Get the converted data using AFEC_GetConvertedData() or AFEC_GetLastConvertedData().

```

/* Step 11: Finally, get the converted data using
AFEC_GetConvertedData() or AFEC_GetLastConvertedData() */
printf("\n\rCH Voltage(mV) \n\r");

ch = (AFEC_GetLastConvertedData(AFEC0) & AFEC_LCDR_CHNB_Msk ) >>
AFEC_LCDR_CHNB_Pos;
voltage = ((AFEC_GetLastConvertedData(AFEC0) & AFEC_LCDR_LDATA_Msk)) *
3254/ 4096;

printf("%02u %04u\n\r" , (unsigned int)ch, (unsigned int)voltage)

```

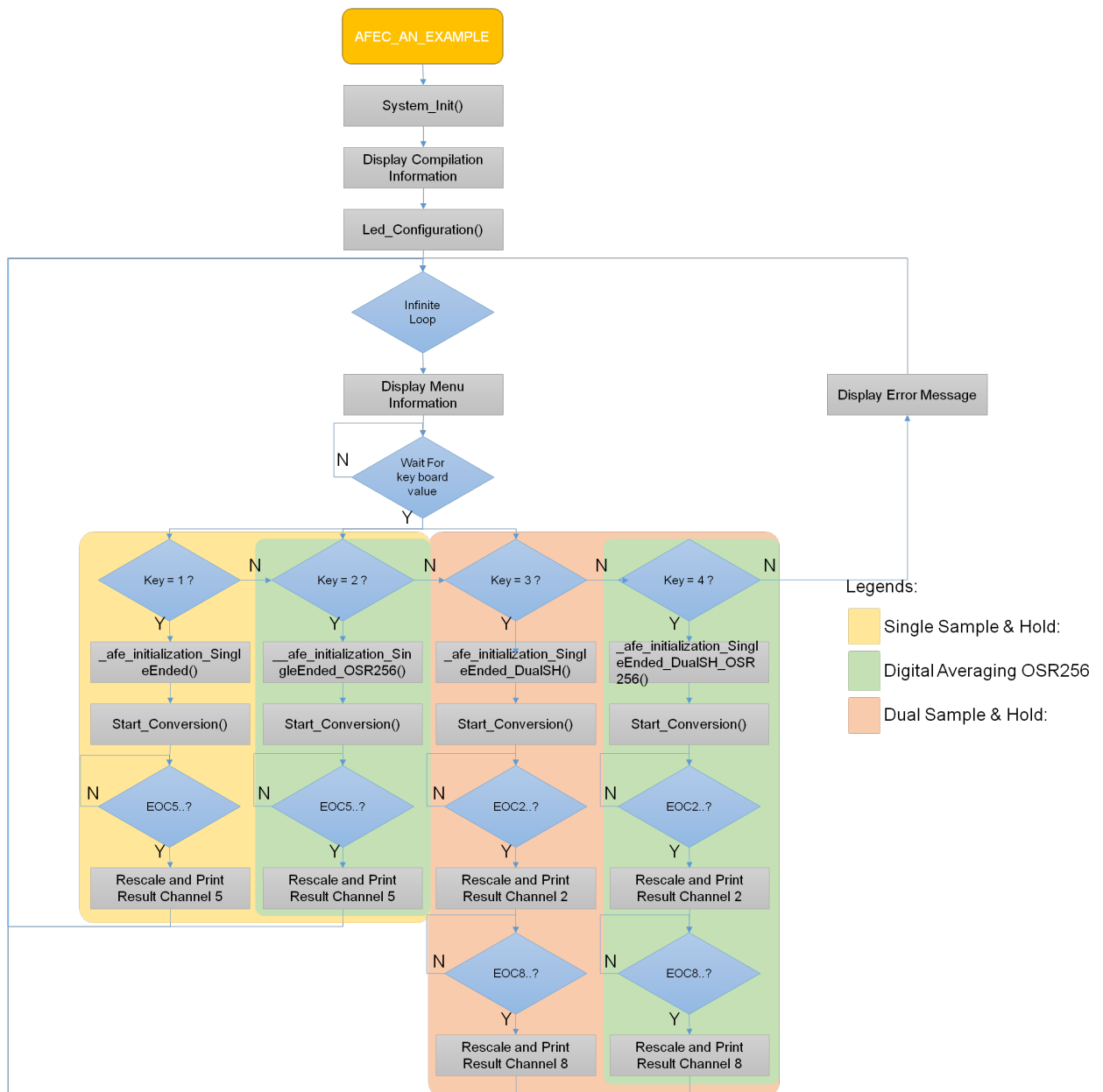
5. Application Example

This application example uses the Single-ended mode with averaging and dual Sample-and-Hold mode.

Several different AFEC configurations have been implemented into a 4-stage state machine by using four different AFEC configuration functions as described below:

- Case 1: Single-ended with software trigger
- Case 2: Single-ended with software trigger with averaging ON
- Case 3: Single-ended, dual Sample-and-Hold with software trigger
- Case 4: Single-ended, dual Sample-and-Hold with software trigger with averaging ON

Figure 5-1 Application Example Flow Chart



5.1. AFEC Main Configuration

In this example, the AFEC is configured as follows:

- AFEC frequency = 10 MHz
- VREFP = 3.253V (refer to the [Atmel SAMV71 Xplained Ultra User Guide, ref. 42408](#))
- Free Run mode OFF (because averaging is used)
- Single-ended (Differential is not possible with the existing hardware)
- Single and Dual Sample-and-Hold
- Channel 5 used for the Single Sample-and-Hold mode
- Channel 2 and 8 used for the Dual Sample-and-Hold mode
- Channel Offset: 500mV (0x200) applied on converted channel

5.2. Terminal Window Configuration

Once the application is compiled, a terminal window must be opened with the following settings:

- Com Port: Board dependent (refer to your device manager)
- Connection Type: Serial
- Speed: 115200
- Stop bit: 1
- Data Bit: 8
- Parity bit: None
- Flow Control: None

5.3. Bench Setup

The application is used to measure voltages on different channels. To do so, at least two DC power supplies are required to deliver two different accurate values.

The material used for the application is:

- Agilent E3632A DC Power Supply to deliver 2.4V DC
- Two Agilent 34410A Digital Multimeters
- Agilent 66311 High Accuracy Mobile Communications DC Source to deliver 1.0V DC

Figure 5-2 Bench Setup



5.4. Results

The application performs different types of conversions, as shown in the screenshot below.

Figure 5-3 Application Example Terminal Window Display

```
COM15 - PuTTY

-- AFE Application Note Example 1.3 --
-- SAM V71 Xplained Ultra
-- Compiled: Jul 15 2015 17:48:57 With GCC--

Choose your AFEC Configuration:
-1-: Single Ended with Software Trigger
-2-: Single Ended with Software Trigger with averaging ON
-3-: Single Ended, Dual Sample and Hold with Software Trigger
-4-: Single Ended, Dual Sample and Hold with Software Trigger with averaging ON

1: You chose : Single Ended with Software Trigger
CH Voltage (mV)
05 2379 (2996 is the code value)

Choose your AFEC Configuration:
-1-: Single Ended with Software Trigger
-2-: Single Ended with Software Trigger with averaging ON
-3-: Single Ended, Dual Sample and Hold with Software Trigger
-4-: Single Ended, Dual Sample and Hold with Software Trigger with averaging ON

2: You chose : Single Ended with Software Trigger and Digital averaging
CH Voltage (mV)
05 2381 (47978 is the code value)

Choose your AFEC Configuration:
-1-: Single Ended with Software Trigger
-2-: Single Ended with Software Trigger with averaging ON
-3-: Single Ended, Dual Sample and Hold with Software Trigger
-4-: Single Ended, Dual Sample and Hold with Software Trigger with averaging ON

3: You chose : Single Ended, Dual Sample and Hold with Software Trigger
CH Voltage (mV)
02 0995 (1254 is the code value)
08 2398 (3020 is the code value)

Choose your AFEC Configuration:
-1-: Single Ended with Software Trigger
-2-: Single Ended with Software Trigger with averaging ON
-3-: Single Ended, Dual Sample and Hold with Software Trigger
-4-: Single Ended, Dual Sample and Hold with Software Trigger with averaging ON

3: You chose : Single Ended, Dual Sample and Hold with Software Trigger with averaging ON
CH Voltage (mV)
02 0993 (20022 is the code value)
08 2393 (48223 is the code value)

Choose your AFEC Configuration:
-1-: Single Ended with Software Trigger
-2-: Single Ended with Software Trigger with averaging ON
-3-: Single Ended, Dual Sample and Hold with Software Trigger
-4-: Single Ended, Dual Sample and Hold with Software Trigger with averaging ON
```

5.4.1. Single-Ended with Software Trigger

In this configuration, the AFEC is defined as Single-ended, without averaging and in single Sample-and-Hold mode.

Channel 5 is connected to the 2.4V external DC voltage source. The result of this conversion is 2379 mV, and matches the resulting digital code 2996 out of 4096 (12-bit resolution).

Steps 1 to 5 have been described in the section [AFEC Module Introduction](#). Steps 6 to 8 describe the additional actions to perform if the application requires offset compensation.

Offset compensation is added by setting the register AFEC_COCCR of a dedicated channel. To do so, the related channel must be selected previously by setting the register AFEC_CSELR.

```
/**
 * \brief Initialize AFE in Single-Ended.
 *
 */
static void _afe_initialization_SingleEnded(void) {

    /* Step 1: Configure the pins for AFEC. (If not already done by
    default) */
    /* Step 2: AFEC init */
    AFEC_Initialize( AFEC0, ID_AFEC0 );

    /*Step 3: AFEC Mode Register Configuration */
    AFEC_SetModeReg(AFEC0,AFEC_MR_FREERUN_OFF|AFEC_MR_TRANSFER(1) |
    AFEC_MR_TRACKTIM(2) | AFEC_MR_ONE | AFEC_MR_STARTUP_SUT512);

    /* Step 4      Set AFEC clock AFEC_SetClock() */
    AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK );

    /* Step 5      Select the active channel using the Macro definition
    AFEC_EnableChannel(). */
    AFEC_EnableChannel(AFEC0, TEST_CHANNEL_SINGLE_ENDED);

    /* Step 6      Adjust the channel level offset */
    AFEC_SetAnalogOffset(AFEC0, TEST_CHANNEL_SINGLE_ENDED, CHANNEL_OFFSET);

    /* Step 7 Enable the PGA 0 and 1 of the AFEC0 and adjust the
    performances by tuning the AFE Bias Current Control */
    AFEC_SetAnalogControl(AFEC0, AFEC_ACR_IBCTL(1) | AFEC_ACR_PGA0_ON |
    AFEC_ACR_PGA1_ON );

    /* Step 8 Configure the Extended Mode Register by disabling the
    Averaging, enabling the channel tag and using the Single Trigger Mode */
    AFEC_SetExtModeReg(AFEC0,0|AFEC_EMR_RES_NO_AVERAGE|
    AFEC_EMR_SIGNMODE_SE_UNSG_DF_SIGN| AFEC_EMR_TAG | AFEC_EMR_STM );
}
```

5.4.2. Single-Ended with Software Trigger with averaging ON

In this configuration, the AFEC is defined as Single-ended, with averaging, and in single Sample-and-Hold mode.

Channel 5 is connected to the 2.4V external DC voltage source. The result of this conversion is 2381 mV, and matches the resulting digital code 47978 out of 65536 (16-bit resolution).

The source code for initialization is found below. Steps 1 to 5 have been described in the section [AFEC Module Introduction](#). Steps 6 to 8 describe the additional actions to perform if the application requires offset compensation or averaging.

To enable Averaging mode, the Free Run mode **must be disabled** and the AFEC_EMR.RES field must be set with the correct OSR value.

```
/**
 * \brief Initialize AFE in Single-Ended with digital averaging OSR 256.
 *
 */
static void _afe_initialization_SingleEnded_OSR256(void) {

    /* Step 1: Configure the pins for AFEC. (If not already done by
    default) */
    /* Step 2: AFEC init */
    AFEC_Initialize( AFEC0, ID_AFEC0 );

    /*Step 3: AFEC Mode Register Configuration */
    AFEC_SetModeReg(AFEC0,AFEC_MR_FREERUN_OFF|AFEC_MR_TRANSFER(1) |
    AFEC_MR_TRACKTIM(2) | AFEC_MR_ONE | AFEC_MR_STARTUP_SUT512);

    /* Step 4      Set AFEC clock AFEC_SetClock() */
    AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK );

    /* Step 5      Select the active channel using the Macro definition
    AFEC_EnableChannel(). */
    AFEC_EnableChannel(AFEC0, TEST_CHANNEL_SINGLE_ENDED);

    /* Step 6      Adjust the channel level offset */
    AFEC_SetAnalogOffset(AFEC0, TEST_CHANNEL_SINGLE_ENDED, CHANNEL_OFFSET);

    /* Step 7 Enable the PGA 0 and 1 of the AFEC0 and adjust the
    performances by tuning the AFE Bias Current Control */
    AFEC_SetAnalogControl(AFEC0, AFEC_ACR_IBCTL(1) | AFEC_ACR_PGA0_ON |
    AFEC_ACR_PGA1_ON );

    /* Step 8 Configure the Extended Mode Register by enabling the
    Averaging, enabling the channel tag and using the Single Trigger Mode */
    AFEC_SetExtModeReg(AFEC0,0|AFEC_EMR_RES_OSR256|
    AFEC_EMR_SIGNMODE_SE_UNSG_DF_SIGN | AFEC_EMR_TAG | AFEC_EMR_STM );
}
```

5.4.3. Single-Ended, Dual Sample-and-Hold with Software Trigger

In this configuration, the AFEC is defined as Single-ended, without averaging and in dual Sample-and-Hold mode.

In this mode, two channels are sampled at the same time.

Channel 2 is connected to the 1.0V external DC voltage source. The result of this conversion is 995mV, and matches the resulting digital code 1254 out of 4096 (12-bit resolution).

Channel 8 is connected to the 2.4V external DC voltage source. The result of this conversion is 2398mV, and matches the resulting digital code 3020 out of 4096 (12-bit resolution).

The source code for initialization is found below. Steps 1 to 5 have been described in the section [AFEC Module Introduction](#) on page 12. Steps 6 to 8 describe the additional actions to perform if the application requires offset compensation, averaging or dual Sample-and-Hold.

Single Sample-and-Hold mode is enabled by default. To enable dual Sample-and-Hold mode, the channel combination must be identified by using the table below.

Table 5-1 Input Pins and Channel Number in Dual Sample-and-Hold Mode

Single-Ended Input Pins	Differential Input Pins	Channel Numbers
AFE_AD0 and AFE_AD6	AFE_AD0-AD1 and AFE_AD6–AFE_AD7	CH0
AFE_AD1 and AFE_AD7	–	CH1
...	...	...
AFE_AD4 and AFE_AD10	AFE_AD4–AFE_AD5 and AFE_AD10–AFE_AD11	CH4
AFE_AD5 and AFE_AD11	–	CH5

For this application, channel 2 (composed of channel 2 and channel 8) is the channel to be configured. Once the channel combination is identified, the Sample-and-Hold Mode register (AFEC_SHMR) must be set. The DC offset correction must be applied on both channel 2 and channel 8 to get correct results.

```
static void _afe_initialization_SingleEnded_DualSH(void) {

    /* Step 1: Configure the pins for AFEC. (If not already done by
    default) */
    /* Step 2: AFEC init */
    AFEC_Initialize( AFEC0, ID_AFEC0 );

    /* Step 3: AFEC Mode Register Configuration */
    AFEC_SetModeReg(AFEC0,AFEC_MR_FREERUN_OFF|AFEC_MR_TRANSFER(1) |
    AFEC_MR_TRACKTIM(2) | AFEC_MR_ONE | AFEC_MR_STARTUP_SUT512);

    /* Step 4: Enable the FAEC Dual Sample And Hold feature for the
    channel 2 */
    AFEC0->AFEC_SHMR |= AFEC_SHMR_DUAL2;

    /* Step 5      Set AFEC clock AFEC_SetClock() */
    AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK );

    /* Step 6      Select the active channel using the Macro definition
    AFEC_EnableChannel(). */
    AFEC_EnableChannel(AFEC0, TEST_CHANNEL_SINGLE_ENDED_DUALSH); // the
    channel is different than in previous example.

    /* Step 7      Adjust the offset level of each channels */
    AFEC_SetAnalogOffset(AFEC0, TEST_CHANNEL_SINGLE_ENDED_DUALSH,
    CHANNEL_OFFSET);
    AFEC_SetAnalogOffset(AFEC0, 8, CHANNEL_OFFSET);

    /* Step 8 Enable the PGA 0 and 1 of the AFEC0 and adjust the
    performances by tuning the AFE Bias Current Control */
    AFEC_SetAnalogControl(AFEC0, AFEC_ACR_IBCTL(1) | AFEC_ACR_PGA0_ON |
    AFEC_ACR_PGA1_ON );

    /* Step 9 Configure the Extended Mode Register by enabling the
    Averaging, enabling the channel tag and using the Single Trigger Mode */
    AFEC_SetExtModeReg(AFEC0,0|AFEC_EMR_RES_NO_AVERAGE|
    AFEC_EMR_SIGNMODE_SE_UNSG_DF_SIGN| AFEC_EMR_TAG | AFEC_EMR_STM );
}
```

5.4.4. 5.4.4 Single-Ended, Dual Sample-and-Hold with Software Trigger with Averaging ON

In this configuration, the AFEC is defined as Single-ended, without averaging and in dual Sample-and-Hold.

In this mode, two channels are sampled at the same time.

Channel 2 is connected to the 1.0V external DC voltage source. The result of this conversion is 993mV, and matches the resulting digital code 20022 out of 65536 (16-bit resolution).

Channel 8 is connected to the 2.4V external DC voltage source. The result of this conversion is 2393mV, and matches the resulting digital code 48223 out of 65536 (16-bit resolution).

The source code for initialization is found below. Steps 1 to 5 have been described in the section [AFEC Module Introduction](#). Steps 6 to 8 describe the additional actions to perform if the application requires offset compensation, averaging or dual Sample-and-Hold.

Single Sample-and-Hold mode is enabled by default. To enable dual Sample-and-Hold mode, the channel combination must be identified by using the table below.

Table 5-2 Input Pins and Channel Number in Dual Sample-and-Hold Mode

Single-Ended Input Pins	Differential Input Pins	Channel Numbers
AFE_AD0 and AFE_AD6	AFE_AD0-AD1 and AFE_AD6–AFE_AD7	CH0
AFE_AD1 and AFE_AD7	–	CH1
...	...	...
AFE_AD4 and AFE_AD10	AFE_AD4–AFE_AD5 and AFE_AD10–AFE_AD11	CH4
AFE_AD5 and AFE_AD11	–	CH5

For this application, channel 2 (composed of the channel 2 and channel 8) is the channel to be configured. Once the channel combination is identified, the Sample-and-Hold Mode register (AFEC_SHMR) must be set.

The DC offset correction must be applied on both channel 2 and channel 8 to get correct results.

```
static void _afe_initialization_SingleEnded_DualSH_OSR256(void) {  
    /* Step 1: Configure the pins for AFEC. (If not already done by  
    default) */  
    /* Step 2: AFEC init */  
    AFEC_Initialize( AFEC0, ID_AFEC0 );  
  
    /* Step 3: AFEC Mode Register Configuration */  
    AFEC_SetModeReg( AFEC0, AFEC_MR_FREERUN_OFF | AFEC_MR_TRANSFER(1) |  
    AFEC_MR_TRACKTIM(2) | AFEC_MR_ONE | AFEC_MR_STARTUP_SUT512 );  
  
    /* Step 4: Enable the FAEC Dual Sample And Hold feature for the  
    channel 2 */  
    AFEC0->AFEC_SHMR |= AFEC_SHMR_DUAL2;  
  
    /* Step 5      Set AFEC clock AFEC_SetClock() */  
    AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK );  
  
    /* Step 6      Select the active channel using the Macro definition  
    AFEC_EnableChannel(). */  
}
```

```

    AFEC_EnableChannel(AFEC0, TEST_CHANNEL_SINGLE_ENDED_DUALSH);    // the
    channel is different than in previous example.

    /* Step 7    Adjust the offset level of each channels */
    AFEC_SetAnalogOffset(AFEC0, TEST_CHANNEL_SINGLE_ENDED_DUALSH,
    CHANNEL_OFFSET);
    AFEC_SetAnalogOffset(AFEC0, 8, CHANNEL_OFFSET);

    /* Step 8 Enable the PGA 0 and 1 of the AFEC0 and adjust the
    performances by tuning the AFE Bias Current Control */
    AFEC_SetAnalogControl(AFEC0, AFEC_ACR_IBCTL(1) | AFEC_ACR_PGA0_ON |
    AFEC_ACR_PGA1_ON );

    /* Step 9 Configure the Extended Mode Register by enabling the
    Averaging, enabling the channel tag and using the Single Trigger Mode */
    AFEC_SetExtModeReg(AFEC0, 0 | AFEC_EMR_RES_OSR256 |
    AFEC_EMR_SIGNMODE_SE_UNSG_DF_SIGN | AFEC_EMR_TAG | AFEC_EMR_STM );
}

```

6. Reference Documents

The following documents were used to issue this application note:

Table 6-1 Reference Documents

Document Name	Release	Link
Atmel SMART SAM E70 Datasheet	11296C - 06/2015	http://www.atmel.com/Images/Atmel-11296-32-bit-Cortex-M7-Microcontroller-SAM-E70Q-SAM-E70N-SAM-E70J_Datasheet.pdf
Atmel SAMV71 Xplained Ultra User Guide	42408B - 06/2015	http://www.atmel.com/tools/ATSAMV71-XULT.aspx?tab=documents
Atmel AT03078: Using the AFEC in SAM4E Microcontroller	42143A-06/2013	http://www.atmel.com/Images/Atmel-42143-SAM-AT03078-Using-the-AFEC-in-SAM4E-Microcontroller_Application-Note.pdf

7. Appendix: Example Application Source Code

```
/*
-----
*           SAM Software Package License
*
-----
* Copyright (c) 2011, Atmel Corporation
*
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are
met:
*
* - Redistributions of source code must retain the above copyright notice,
* this list of conditions and the disclaimer below.
*
* Atmel's name may not be used to endorse or promote products derived from
* this software without specific prior written permission.
*
* DISCLAIMER: THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS
OR
* IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF
* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT
ARE
* DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
-----
*/

/** \file
*
* This file contains all the specific code for the AFEC Application Note
example.
*
*/

/
*-----
--
*           Headers
*
-----
--*/
```

```

#include "board.h"
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

/
*-----
--
*           Local definitions
*-----
--*/

/** First AFE Channel used*/
#define TEST_CHANNEL_SINGLE_ENDED          5           //AFE0_AD5->PB2
#define TEST_CHANNEL_SINGLE_ENDED_DUALSH   2           //AFE0_AD2->PB3 &
AFE0_AD8->PA19 (Channel 2 in Dual Sample & hold Single Ended mode)

/** AFE Channel DAC Offset */
#define CHANNEL_OFFSET      0x200

/* AFE Clock Value */
#define AFE_CLK              10000000

/
*-----
--
*           Local functions
*-----
--*/

/**
 * \brief Initialize AFE in Single-Ended.
 */
static void _afe_initialization_SingleEnded(void) {

    /* Step 1: Configure the pins for AFEC. (If not already done by
    default) */
    /* Step 2: AFEC init */
    AFEC_Initialize( AFEC0, ID_AFEC0 );

    /*Step 3: AFEC Mode Register Configuration */
    AFEC_SetModeReg(AFEC0,AFEC_MR_FREERUN_OFF|AFEC_MR_TRANSFER(1)|
    AFEC_MR_TRACKTIM(2)| AFEC_MR_ONE| AFEC_MR_STARTUP_SUT512);

    /* Step 4      Set AFEC clock AFEC SetClock() */
    AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK );

    /* Step 5      Select the active channel using the Macro definition
    AFEC_EnableChannel(). */
    AFEC_EnableChannel(AFEC0, TEST_CHANNEL_SINGLE_ENDED);

    /* Step 6      Adjust the channel level offset */
    AFEC_SetAnalogOffset(AFEC0, TEST_CHANNEL_SINGLE_ENDED, CHANNEL_OFFSET);
/* Step 7 Enable the PGA 0 and 1 of the AFEC0 and adjust the performances

```

```

by tuning the AFE Bias Current Control */
AFECA_SetAnalogControl(AFEC0, AFEC_ACR_IBCTL(1) | AFEC_ACR_PGA0_ON |
AFECA_ACR_PGA1_ON );

/* Step 8 Configure the Extended Mode Register by disabling the
Averaging, enabling the channel tag and using the Single Trigger Mode */
AFECA_SetExtModeReg(AFEC0,0|AFEC_EMR_RES_NO_AVERAGE|
AFECA_EMR_SIGNMODE_SE_UNSG_DF_SIGN| AFEC_EMR_TAG | AFEC_EMR_STM );
}

/**
 * \brief Initialize AFE in Single-Ended with digital averaging OSR 256.
 *
 */
static void _afe_initialization_SingleEnded_OSR256(void) {

/* Step 1: Configure the pins for AFEC. (If not already done by
default) */
/* Step 2: AFEC init */
AFECA_Initialize( AFEC0, ID_AFEC0 );

/*Step 3: AFEC Mode Register Configuration */
AFECA_SetModeReg(AFEC0,AFEC_MR_FREERUN_OFF|AFEC_MR_TRANSFER(1)|
AFECA_MR_TRACKTIM(2)| AFEC_MR_ONE| AFEC_MR_STARTUP_SUT512);

/* Step 4 Set AFEC clock AFEC_SetClock() */
AFECA_SetClock( AFEC0, AFE_CLK, BOARD_MCK );

/* Step 5 Select the active channel using the Macro definition
AFECA_EnableChannel(). */
AFECA_EnableChannel(AFEC0, TEST_CHANNEL_SINGLE_ENDED);

/* Step 6 Adjust the channel level offset */
AFECA_SetAnalogOffset(AFEC0, TEST_CHANNEL_SINGLE_ENDED, CHANNEL_OFFSET);

/* Step 7 Enable the PGA 0 and 1 of the AFEC0 and adjust the
performances by tuning the AFE Bias Current Control */
AFECA_SetAnalogControl(AFEC0, AFEC_ACR_IBCTL(1) | AFEC_ACR_PGA0_ON |
AFECA_ACR_PGA1_ON );

/* Step 8 Configure the Extended Mode Register by enabling the
Averaging, enabling the channel tag and using the Single Trigger Mode */
AFECA_SetExtModeReg(AFEC0,0|AFEC_EMR_RES_OSR256|
AFECA_EMR_SIGNMODE_SE_UNSG_DF_SIGN| AFEC_EMR_TAG | AFEC_EMR_STM );
}

static void _afe_initialization_SingleEnded_DualSH(void) {

/* Step 1: Configure the pins for AFEC. (If not already done by
default) */
/* Step 2: AFEC init */
AFECA_Initialize( AFEC0, ID_AFEC0 );

/* Step 3: AFEC Mode Register Configuration */
AFECA_SetModeReg(AFEC0,AFEC_MR_FREERUN_OFF|AFEC_MR_TRANSFER(1)|
AFECA_MR_TRACKTIM(2)| AFEC_MR_ONE| AFEC_MR_STARTUP_SUT512);

/* Step 4: Enable the FAEC Dual Sample And Hold feature for the
channel 2 */
AFECA->AFECA_SHMR |= AFECA_SHMR_DUAL2;

```

```

/* Step 5    Set AFEC clock AFEC_SetClock() */
AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK );

/* Step 6    Select the active channel using the Macro definition
AFEC_EnableChannel(). */
AFEC_EnableChannel(AFEC0, TEST_CHANNEL_SINGLE_ENDED_DUALSH);    // the
channel is different than in previous example.

/* Step 7    Adjust the offset level of each channels */
AFEC_SetAnalogOffset(AFEC0, TEST_CHANNEL_SINGLE_ENDED_DUALSH,
CHANNEL_OFFSET);
AFEC_SetAnalogOffset(AFEC0, 8, CHANNEL_OFFSET);

/* Step 8 Enable the PGA 0 and 1 of the AFEC0 and adjust the
performances by tuning the AFE Bias Current Control */
AFEC_SetAnalogControl(AFEC0, AFEC_ACR_IBCTL(1) | AFEC_ACR_PGA0_ON |
AFEC_ACR_PGA1_ON );

/* Step 9 Configure the Extended Mode Register by enabling the
Averaging, enabling the channel tag and using the Single Trigger Mode */
AFEC_SetExtModeReg(AFEC0,0|AFEC_EMR_RES_NO_AVERAGE|
AFEC_EMR_SIGNMODE_SE_UNSG_DF_SIGN| AFEC_EMR_TAG | AFEC_EMR_STM );
}

static void _afe_initialization_SingleEnded_DualSH_OSR256(void) {

/* Step 1: Configure the pins for AFEC. (If not already done by
default) */
/* Step 2: AFEC init */
AFEC_Initialize( AFEC0, ID_AFEC0 );

/* Step 3: AFEC Mode Register Configuration */
AFEC_SetModeReg(AFEC0,AFEC_MR_FREERUN_OFF|AFEC_MR_TRANSFER(1)|
AFEC_MR_TRACKTIM(2)| AFEC_MR_ONE| AFEC_MR_STARTUP_SUT512);

/* Step 4: Enable the FAEC Dual Sample And Hold feature for the
channel 2 */
AFEC0->AFEC_SHMR |= AFEC_SHMR_DUAL2;

/* Step 5    Set AFEC clock AFEC_SetClock() */
AFEC_SetClock( AFEC0, AFE_CLK, BOARD_MCK );

/* Step 6    Select the active channel using the Macro definition
AFEC_EnableChannel(). */
AFEC_EnableChannel(AFEC0, TEST_CHANNEL_SINGLE_ENDED_DUALSH);    // the
channel is different than in previous example.

/* Step 7    Adjust the offset level of each channels */
AFEC_SetAnalogOffset(AFEC0, TEST_CHANNEL_SINGLE_ENDED_DUALSH,
CHANNEL_OFFSET);
AFEC_SetAnalogOffset(AFEC0, 8, CHANNEL_OFFSET);

/* Step 8 Enable the PGA 0 and 1 of the AFEC0 and adjust the
performances by tuning the AFE Bias Current Control */
AFEC_SetAnalogControl(AFEC0, AFEC_ACR_IBCTL(1) | AFEC_ACR_PGA0_ON |
AFEC_ACR_PGA1_ON );

/* Step 9 Configure the Extended Mode Register by enabling the
Averaging, enabling the channel tag and using the Single Trigger Mode */
AFEC_SetExtModeReg(AFEC0,0|AFEC_EMR_RES_OSR256|
AFEC_EMR_SIGNMODE_SE_UNSG_DF_SIGN| AFEC_EMR_TAG | AFEC_EMR_STM );

```

```

}

/
*-----
--
*           Exported functions
*-----
--*/
/**
 * \brief getting-started Application entry point.
 *
 * \return Unused (ANSI-C compatibility).
 */
extern int main( void )
{
    /* Local variable */
    uint32_t ch;
    uint32_t voltage;
    uint8_t key = 0;
    uint32_t i=0;

    /*** SYSTEM INIT ***/
    /* Disable watchdog */
    WDT_Disable( WDT );
    /* Enable I and D cache */
    SCB_EnableICache();
    SCB_EnableDCache();

    /* Output example information */
    printf( "\n\r-- AFE Application Note Example %s --\n\r",
    SOFTPACK_VERSION );
    printf( "-- %s\n\r", BOARD_NAME );
    printf( "-- Compiled: %s %s With %s--\n\r", __DATE__, __TIME__,
    COMPILER_NAME);

    /*** LED CONFIGURATION ***/
    /* Configure led */
    LED_Configure( 1 );
    /* The led is lit ON */
    LED_Set( 1 );

    while ( 1 )
    {

        /* Display menu lines */
        printf( "\n\r Choose your AFEC Configuration:") ;
        printf( "\n\r\t -1-: Single Ended with Software Trigger" ) ;
        printf( "\n\r\t -2-: Single Ended with Software Trigger with
    averaging ON" ) ;
        printf( "\n\r\t -3-: Single Ended, Dual Sample and Hold with
    Software Trigger" ) ;
        printf( "\n\r\t -4-: Single Ended, Dual Sample and Hold with
    Software Trigger with averaging ON" ) ;

        /*waiting for a typed
    value*/
        key = DBG_GetChar();

        /* State machine depending on the wanted AFEC configuration */

```

```

switch (key){

    /* -1-: Single Ended with Software Trigger */
    case '1':
        printf( "\n\n\n\r 1: You chose : Single Ended with
Software Trigger") ;

        /* Step 1 to step 8: Initialize the AFE */
        _afe_initialization_SingleEnded();

        /* Step 9: Start Conversion by software trigger */
        AFEC_StartConversion(AFEC0);

        /* Step 10: Wait for the end of the conversion by
polling status with AFEC_GetStatus() */
        while ( !(AFEC_GetStatus(AFEC0) & AFEC_ISR_EOC5));

        LED_Toggle( 1 );

        /* Step 11: Finally, get the converted data and the
channel used using AFEC_GetConvertedData() or AFEC_GetLastConvertedData()
*/;
        printf("\n\rCH Voltage(mV) \n\r");
        ch = (AFEC_GetLastConvertedData(AFEC0) &
AFEC_LCDR_CHNB_Msk ) >> AFEC_LCDR_CHNB_Pos;

        /* Rescaling and displaying of the result */
        voltage = ((AFEC0->AFEC_CDR & AFEC_CDR_DATA_Msk)) *
3253/4096; //3253 is the Vref volatge in mV
        printf("%02u %04u (%04u is the code value)\n\r" ,
(unsigned int)ch, (unsigned int)voltage, (AFEC_GetLastConvertedData(AFEC0) &
AFEC_LCDR_LDATA_Msk));

        break;

    /* -2-: Single Ended with Software Trigger with averaging
ON */
    case '2':
        printf( "\n\n\n\r 2: You chose : Single Ended with
Software Trigger and Digital averaging") ;

        /* Step 1 to step 8: Initialize the AFE */
        _afe_initialization_SingleEnded_OSR256();

        /* Step 9: Start Conversion by software trigger */
        AFEC_StartConversion(AFEC0);

        /* Step 10: Wait for the end of the conversion by
polling status with AFEC_GetStatus() */
        while ( !(AFEC_GetStatus(AFEC0) & AFEC_ISR_EOC5));

        LED_Toggle( 1 );

        /* Step 11: Finally, get the converted data and the
channel used using AFEC_GetConvertedData() or AFEC_GetLastConvertedData()
*/
        printf("\n\rCH Voltage(mV) \n\r");

        /* Rescaling and displaying of the result */
        ch = (AFEC_GetLastConvertedData(AFEC0) &
AFEC_LCDR_CHNB_Msk ) >> AFEC_LCDR_CHNB_Pos;
        voltage = ((AFEC_GetLastConvertedData(AFEC0) &
AFEC_LCDR_LDATA_Msk)) * 3253/65536; //3253 is the Vref voltage in mV

```

```

this time the scale factor is 65536 (16-bit resolution)
    printf("%02u %04u (%04u is the code value)\n\r" ,
(unsigned int)ch, (unsigned int)voltage, (AFEC_GetLastConvertedData(AFEC0) &
AFEC_LCDR_LDATA_Msk));

    break;

    /* -3-: Single Ended, Dual Sample and Hold with Software
Trigger */
    case '3':
        printf( "\n\n\n\r 3: You chose : Single Ended, Dual
Sample and Hold with Software Trigger" ) ;

        /* Step 1 to step 9: Initialize the AFE */
        _afe_initialization_SingleEnded_DualSH();

        /* Step 10: Start Conversion by software trigger */
        AFEC_StartConversion(AFEC0);

        /* Step 11: Wait for the end of the conversion by
polling status with AFEC_GetStatus() */
        while (! (AFEC_GetStatus(AFEC0) & AFEC_ISR_EOC2)); //
this time the channel is replaced by channel 2: First Channel

        LED_Toggle( 1 );

        /* Step 12: Finally, get the converted data and the
channel used using AFEC_GetConvertedData() or AFEC_GetLastConvertedData()
*/
        printf("\n\rCH Voltage(mV) \n\r");

        /* Rescaling and displaying of the result of value 1 */
        ch = 2;
        voltage = (AFEC_GetConvertedData(AFEC0,
AFEC_CSELR_CSEL(2))) * 3253/4096; //3253 is the Vref voltage in mV
        printf("%02u %04u (%04u is the code value)\n\r" ,
(unsigned int)ch, (unsigned int)voltage, AFEC_GetConvertedData(AFEC0,
AFEC_CSELR_CSEL(2)));

        /* Step 13: Reproducing step 11 for the next channel :
Wait for the end of the conversion by polling status with AFEC_GetStatus()
*/
        while (! (AFEC_GetStatus(AFEC0) & AFEC_ISR_EOC8)); //
this time the channel is replaced by channel 8: second Channel

        /* Rescaling and displaying of the result of value 2*/
        ch = (AFEC_GetLastConvertedData(AFEC0) &
AFEC_LCDR_CHNB_Msk ) >> AFEC_LCDR_CHNB_Pos;
        voltage = (AFEC_GetConvertedData(AFEC0,
AFEC_CSELR_CSEL(8))) * 3253/4096; //3253 is the Vref volatge in mV
        printf("%02u %04u (%04u is the code value)\n\r" ,
(unsigned int)ch, (unsigned int)voltage, AFEC_GetConvertedData(AFEC0,
AFEC_CSELR_CSEL(8)));

        break;

    /* -4-: Single Ended, Dual Sample and Hold with Software
Trigger with averaging ON */
    case '4':
        printf( "\n\n\n\r 3: You chose : Single Ended, Dual
Sample and Hold with Software Trigger with averaging ON" ) ;

```

```

        /* Step 1 to step 9: Initialize the AFE */
        _afe_initialization_SingleEnded_DualSH_OSR256();

        /* Step 10: Start Conversion by software trigger */
        AFEC_StartConversion(AFEC0);

        /* Step 11: Wait for the end of the conversion by
polling status with AFEC_GetStatus() */
        while (!(AFEC_GetStatus(AFEC0) & AFEC_ISR_EOC2)); //
this time the channel is replaced by channel 2: First Channel

        LED_Toggle( 1 );

        /* Step 12: Finally, get the converted data and the
channel used using AFEC_GetConvertedData() or AFEC_GetLastConvertedData()
*/
        printf("\n\rCH Voltage(mV) \n\r");

        /* Rescaling and displaying of the result of value 1 */
        ch = 2;
        voltage = (AFEC_GetConvertedData(AFEC0,
AFEC_CSELR_CSEL(2))) * 3253/65536; //3253 is the Vref voltage in mV this
time the scale factor is 65536 (16-bit resolution)
        printf("%02u %04u (%04u is the code value)\n\r" ,
(unsigned int)ch, (unsigned int)voltage, AFEC_GetConvertedData(AFEC0,
AFEC_CSELR_CSEL(2)));

        /* Step 13: Reproducing step 11 for the next channel :
Wait for the end of the conversion by polling status with AFEC_GetStatus()
*/
        while (!(AFEC_GetStatus(AFEC0) & AFEC_ISR_EOC8)); //
this time the channel is replaced by channel 8: second Channel

        /* Rescaling and displaying of the result of value 2*/
        ch = (AFEC_GetLastConvertedData(AFEC0) &
AFEC_LCDR_CHNB_Msk ) >> AFEC_LCDR_CHNB_Pos;
        voltage = (AFEC_GetConvertedData(AFEC0, AFEC_CSELR_CSEL(8))) *
3253/65536; //3253 is the Vref volatge in mV this time the scale factor
is 65536 (16-bit resolution)
        printf("%02u %04u (%04u is the code value)\n\r" ,
(unsigned int)ch, (unsigned int)voltage, AFEC_GetConvertedData(AFEC0,
AFEC_CSELR_CSEL(8)));

        break;

        default:
            printf( "\n\n\n\r !!!: the entered value is not
correct, back to the menu" ) ;
            break;
    }
    key =0;
}
}

```

8. Revision History

Table 8-1 Using the Analog Front End in the SAM V7x/E7x/S7x MCUs - Rev. 44046A - Revision History

Date	Changes
07-Aug-15	First issue.

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. ARM®, ARM Connected® logo, and others are the registered trademarks or trademarks of ARM Ltd. Other terms and product names may be trademarks of others.

DISCLAIMER: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death ("Safety-Critical Applications") without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.