Freescale Semiconductor
Application Note

Document Number: AN4688
Rev. A, 01/2013

Sampling Audio with ADC on Kinetis KwikStik

ADC, DMA and LCD demo using Processor Expert and Peripheral Initialization Components.

.

Periodical sampling of analog signal is a common task required in many microcontroller (MCU) applications. The Kinetis MCUs provide analog to digital converters (ADCs) with hardware sample triggering that can be easily leveraged for this task. This application note describes the design of an application that uses Processor Expert to configure the ADC and other peripherals to perform periodic sampling of an analog signal obtained from a microphone on a KwikStik device. Its on-board matrix LCD presents the sampled data as a waveform. The source code of the example is available in the archive file AN4688SW.zip.

This note assumes the use of CodeWarrior for MCUs v10.3 or later.

Contents

1 Introduction

Processor Expert is a rapid application development tool that provides a quick way to setup initialization and runtime control of a Freescale MCU and its on-chip peripheral devices. Peripheral Initialization components go a step further by providing pre-configured drivers for these peripheral devices. The Code Warrior for MCUs v10.x tools and Microcontrollers Driver Suite provide support for the Kinetis MCUs.

2 Hardware

The example project uses a Kinetis KwikStik. It is a development board used to evaluate, develop, and debug software written for Kinetis MCUs.

The KwikStik board features:

- 32-bit K40X256 MCU that contains 256KB flash memory and 256KB FlexMemory
- LCD screen with 306 segments
- 3.5 mm audio output jack and two micro USB connectors
- Omni-directional microphone and a buzzer
- On-board J-Link USB programmer
- Infrared communications
- Micro SD card slot
- Capacitive touch sensing interface
- General purpose Tower plug-in (TWRPI) socket
- Manganese lithium rechargeable battery

Detailed information on the KwikStik board and Kinetis K40 MCU are available at: http://www.freescale.com/kwikstik.

3 Description of Operation

The demo application repeatedly samples a single ADC channel at a given frequency and stores the data to a memory buffer via an enhanced Direct Memory Access (eDMA) module.

ADC conversions are triggered with pulses provided continuously by the Programmable Delay Block (PDB) peripheral. After each conversion, the ADC raises a DMA request and the eDMA transfers the data from the sample register to the memory buffer. The benefit of this approach is that the CPU core is not involved with the sequence of operations: it is free to perform other tasks rather than handling an ADC sample stream.

The example code uses a double buffering technique, which means that as one half of the buffer fills with ADC values, the application code processes the accumulated data in other half. When one half the buffer fills with new samples, an interrupt informs the MCU that the buffer end has been reached. Incoming data samples are then transferred to the other half of the buffer, while the application code starts processing the newly-filled half of the buffer. The application code reads the sampled values and displays them as a waveform.

Figure 1 shows the basic signal and data flow in the demo application.

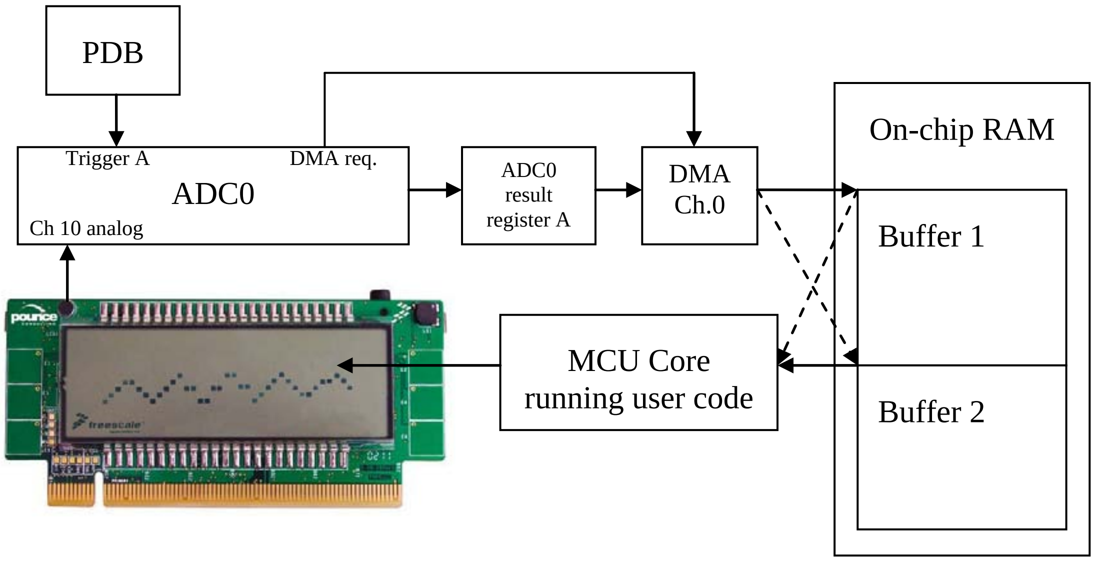


Figure 1. Block Diagram of Signal and Data Flow

4 Development tools

The demo application is created using Processor Expert, a rapid application design tool. Processor Expert enables the quick design and implementation of a microcontroller application. The application is built using Processor Expert’s Embedded Components. These components are building blocks that generate source code files that contain the initialization and driver code for the selected MCU and its on-chip peripherals.

For the Kinetis K40 platform, there are the following types of Embedded Components available:

- **Peripheral Initialization Components** – These components generate the initialization code for a peripheral. The interface of these components gives a user access to all possible features that can be initialized. By pointing and clicking, you can active specific peripheral features and configure them.
- **Logical Device Drivers (LDD)** – These components provide basic initialization code and drivers for runtime use of the peripherals.

To achieve maximum throughput of the ADC data flow, the application’s Peripheral Initialization Components are used to configure ADC, eDMA and PDB. Peripheral Initialization components are used because the required access to certain low-level hardware specific features is not available in LDD components.

Processor Expert is available as a plug-in for CodeWarrior for MCUs or within the standalone Microcontroller Driver Suite. For the compilation and debugging of the demo, CodeWarrior for MCUs v10.2 was used.

For more details on the development tools, see the website http://www.freescale.com.

5 Components configuration

Table 1show the components used in the example application project.

Table 1. Summary of components used in the example application

Embedded Component	Task
CPU (MK40DX256ZVLQ10)	Configures basic settings of K40 MCU on the KwikStik board
GPIO_LDD	Provides driver for a digital pin for testing purposes
SegLCD_LDD	Provides a simple LCD driver for the display on the KwikStik
Init_ADC	Provides ADC (Analog to Digital Converter) initialization code
Init_PDB	Provides PDB (Programmable Delay Block) initialization code
Init_eDMA	Provides eDMA (Direct Memory Access) initialization code

Figure 2 presents the **CodeWarrior Projects** view of the example program in CodeWarrior for MCUs v10.2. It shows the source files and the various Processor Expert components used to implement the functions of the example program.

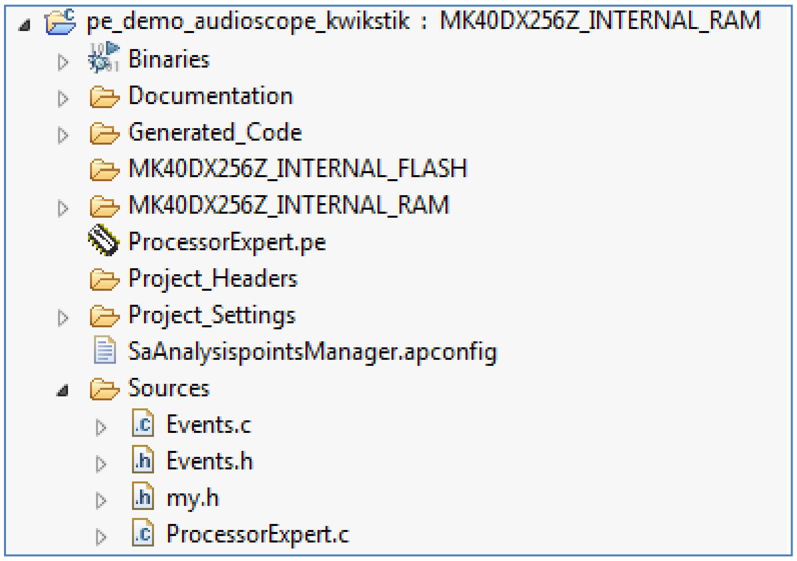
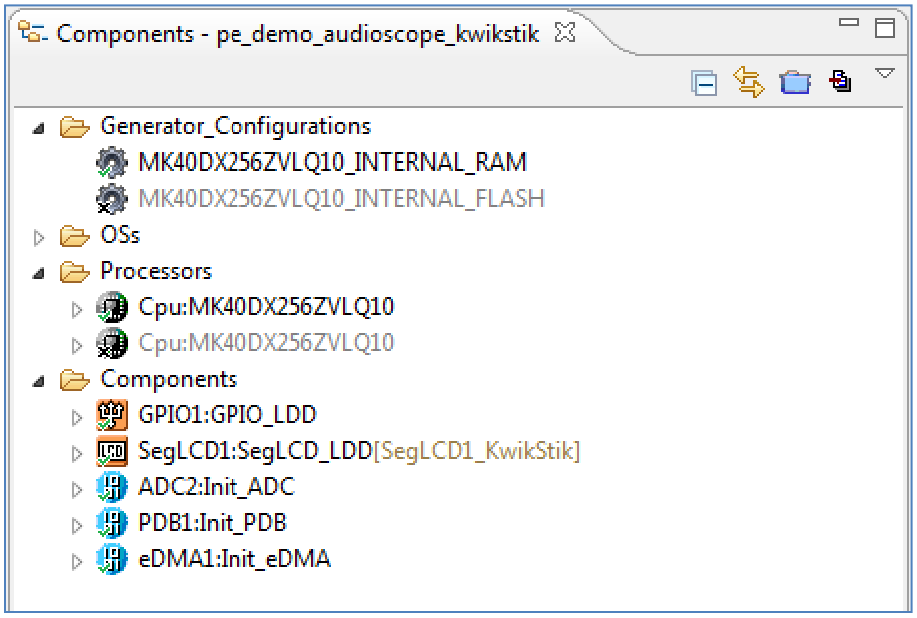

```
pe_demo_audioscope_kwikstik : MK40DX256Z_INTERNAL_RAM
    Binaries
    Documentation
    Generated_Code
    MK40DX256Z_INTERNAL_FLASH
    MK40DX256Z_INTERNAL_RAM
    ProcessorExpert.pe
    Project_Headers
    Project_Settings
    SaAnalysispointsManager.apconfig
    Sources
        Events.c
        Events.h
        my.h
        ProcessorExpert.c
```

Figure 2. CodeWarrior Project Structure of Application Using Processor Expert



Components - pe_demo_audioscope_kwikstik

```
◢ Generator_Configurations
      MK40DX256ZVLQ10_INTERNAL_RAM
      MK40DX256ZVLQ10_INTERNAL_FLASH
▷ OSs
◢ Processors
   ▷ Cpu:MK40DX256ZVLQ10
   ▷ Cpu:MK40DX256ZVLQ10
◢ Components
   ▷ GPIO1:GPIO_LDD
   ▷ SegLCD1:SegLCD_LDD[SegLCD1_KwikStik]
   ▷ ADC2:Init_ADC
   ▷ PDB1:Init_PDB
   ▷ eDMA1:Init_eDMA
```

Figure 3. Processor Expert components used if the project

The following subsections describe important settings of the individual components. You can view complete settings of in the Processor Expert Component Inspector after opening the project and selecting the component.

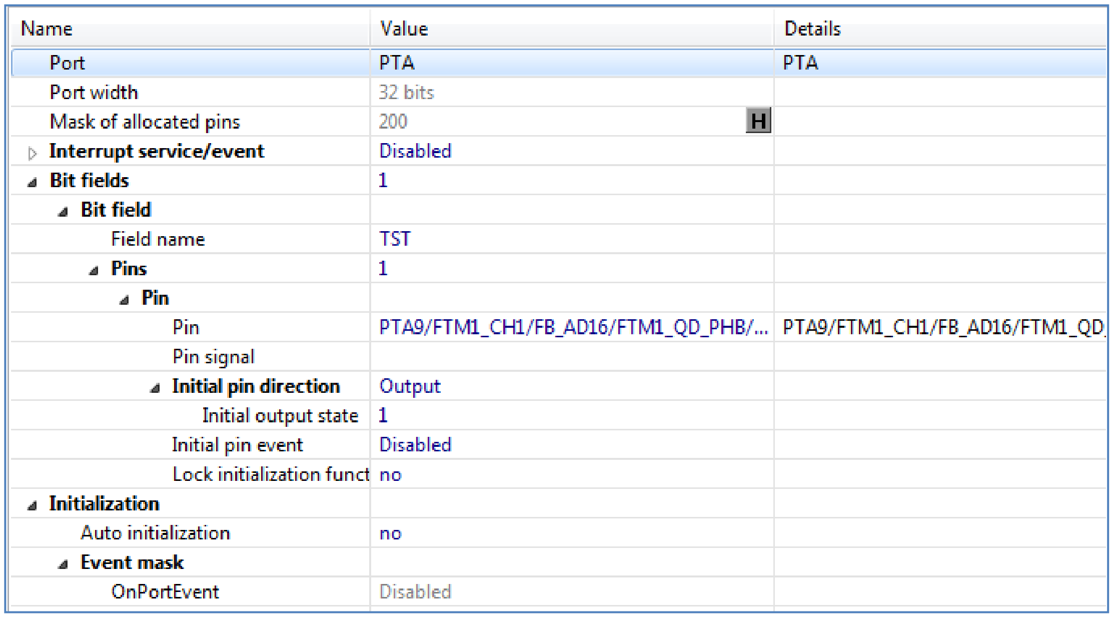
Note: To access component settings in the **CodeWarrior Project** view, click on the `ProcessorExpert.pe` object in the **CodeWarrior Projects** view in the project. Then select the component in the **Components** view and see the settings in Inspector view. For more information on how to change these settings, refer to the Processor Expert User Guide.

5.1 CPU

The CPU component configures the MCU to use the external 50MHz crystal on the KwikStik board. The core clock is initialized to run at 100MHz.

5.2 GPIO_LDD

The GPIO_LDD component provides a driver code for the PTA9 pin. This pin is accessible on the KwikStik Tower connector and is used for debugging and test purposes (for example, measuring the sampling frequency).



Name	Value	Details
Port	PTA	PTA
Port width	32 bits	
Mask of allocated pins	200	
Interrupt service/event	Disabled	
Bit fields	1	
Bit field		
Field name	TST	
Pins	1	
Pin		
Pin	PTA9/FTM1_CH1/FB_AD16/FTM1_QD_PHB/...	PTA9/FTM1_CH1/FB_AD16/FTM1_QD
Pin signal		
Initial pin direction	Output	
Initial output state	1	
Initial pin event	Disabled	
Lock initialization funct	no	
Initialization		
Auto initialization	no	
Event mask		
OnPortEvent	Disabled	

Figure 4. GPIO_LDD settings in Component Inspector

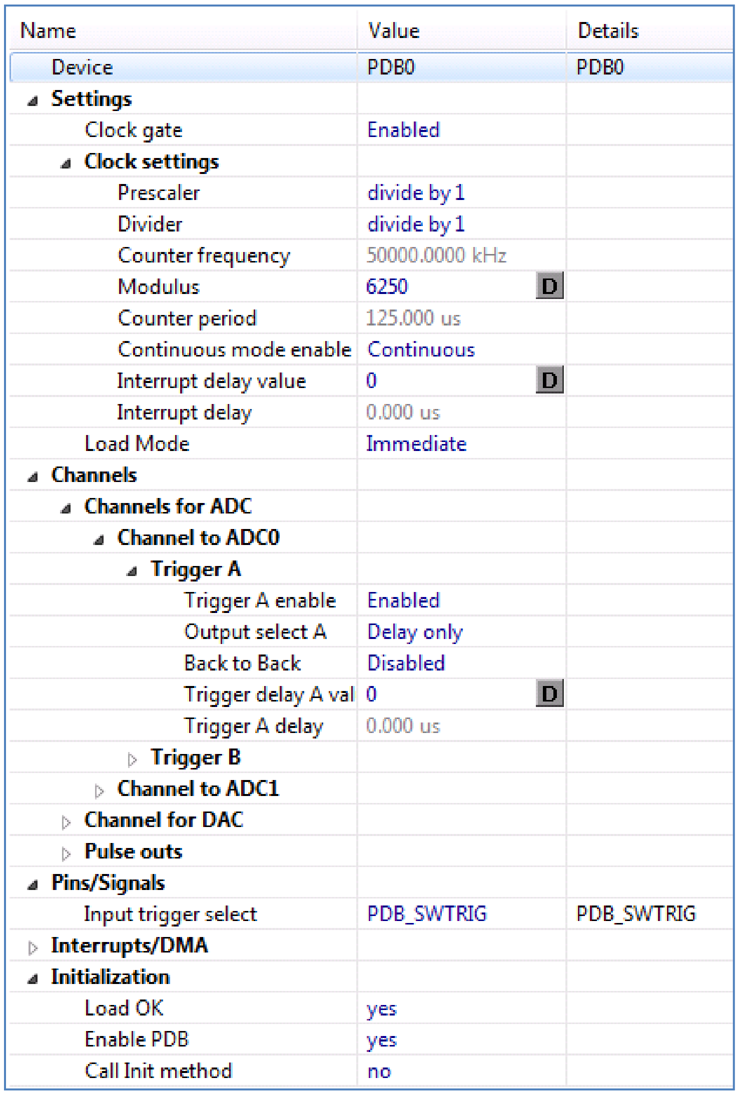
5.3 Init_PDB

The Init_PDB component initializes and configures the PDB. The PDB provides controllable delays from either an internal or external trigger, or a programmable interval tick. These can be used as hardware trigger inputs for the ADCs and/or generate the interval triggers to DACs.

In this example application, the *PDB channel 0* is configured by Init_PDB component with the following settings:

- Prescaler and Divider are set to *1* so the counter input frequency is 50MHz.
- The counter modulus is set to *6250*, which makes the period in 125 µS. This value is equal to sampling frequency 8 kHz.
- Enabled *Continuous mode* means that the counter will continue running in a loop once it’s started.
- Input trigger select is set to *PDB_SWTRIG* which means that PDB is triggered (started) by software only. The PDB starts to provide trigger pulses to ADC after the PDB1_Init() function is called in main().
- Only *trigger A* of the *ADC0* is *enabled* and no delay is configured for it (delay with value 0).

Figure 5 shows the Component Inspector interface for the PDB, and how the settings are described appear in it.



Name	Value	Details
Device	PDB0	PDB0
◢ Settings		
Clock gate	Enabled	
◢ Clock settings		
Prescaler	divide by 1	
Divider	divide by 1	
Counter frequency	50000.0000 kHz	
Modulus	6250 D	
Counter period	125.000 us	
Continuous mode enable	Continuous	
Interrupt delay value	0 D	
Interrupt delay	0.000 us	
Load Mode	Immediate	
◢ Channels		
◢ Channels for ADC		
◢ Channel to ADC0		
◢ Trigger A		
Trigger A enable	Enabled	
Output select A	Delay only	
Back to Back	Disabled	
Trigger delay A val	0 D	
Trigger A delay	0.000 us	
▷ Trigger B		
▷ Channel to ADC1		
▷ Channel for DAC		
▷ Pulse outs		
◢ Pins/Signals		
Input trigger select	PDB_SWTRIG	PDB_SWTRIG
▷ Interrupts/DMA		
◢ Initialization		
Load OK	yes	
Enable PDB	yes	
Call Init method	no	

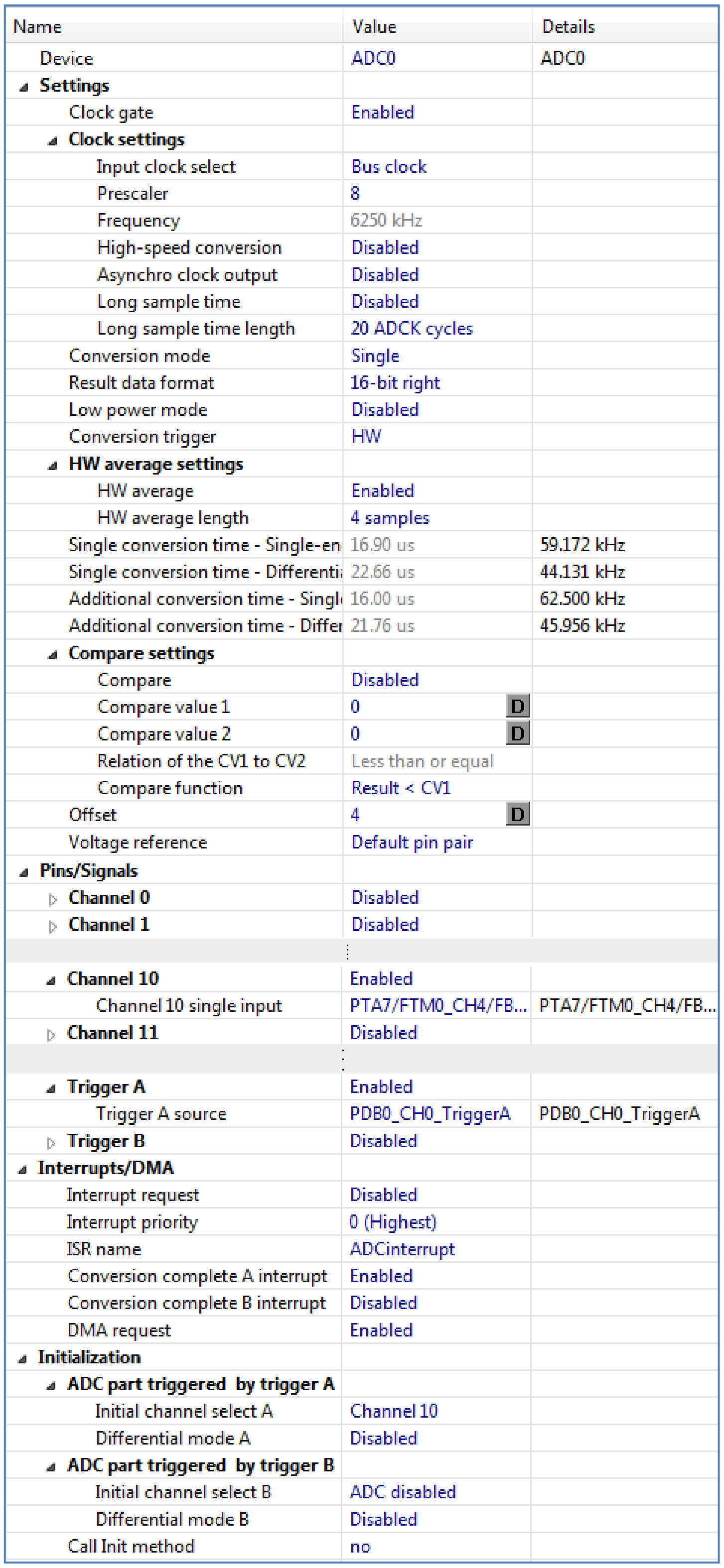
Figure 5. Init_PDB settings in Component Inspector

5.4 Init_ADC

The MCU features a 16-bit analog-to-digital converter (ADC). It uses a successive approximation conversion algorithm and supports up to four pairs of differential and 24 single-ended inputs. It offers a hardware average function, selectable hardware conversion trigger, selectable voltage reference and many other advanced features.

In the example application the Init_ADC component is configured to initializethe ADC converter (ADC0) the following way:

- Result data format is *16-bit* unsigned.
- Conversion mode is *Single.* (That is, every single measurement is started by a trigger pulse).
- ADC clock is set to use the *Bus clock* with Prescaler of 8. The resulting conversion time is 16.9 µs.
- Conversion trigger is *hardware (HW).*
- HW average is *Enabled* with HW average length *4 samples.*
- Channel 10 single input is *enabled*. This channel input is connected to on-board microphone.
- Trigger A is *enabled*. Trigger A source is set to *PDB0_CH0_TriggerA.*
- Interrupt request is *Disabled.*
- DMA request *is Enabled.*
- Initial channel select for trigger A is *Channel 10.* This channel is connected to the on-board microphone.



Name	Value	Details
Device	ADC0	ADC0
◢ **Settings**		
Clock gate	Enabled	
◢ **Clock settings**		
Input clock select	Bus clock	
Prescaler	8	
Frequency	6250 kHz	
High-speed conversion	Disabled	
Asynchro clock output	Disabled	
Long sample time	Disabled	
Long sample time length	20 ADCK cycles	
Conversion mode	Single	
Result data format	16-bit right	
Low power mode	Disabled	
Conversion trigger	HW	
◢ **HW average settings**		
HW average	Enabled	
HW average length	4 samples	
Single conversion time - Single-en	16.90 us	59.172 kHz
Single conversion time - Differenti	22.66 us	44.131 kHz
Additional conversion time - Singl	16.00 us	62.500 kHz
Additional conversion time - Differ	21.76 us	45.956 kHz
◢ **Compare settings**		
Compare	Disabled	
Compare value 1	0 D	
Compare value 2	0 D	
Relation of the CV1 to CV2	Less than or equal	
Compare function	Result < CV1	
Offset	4 D	
Voltage reference	Default pin pair	
◢ **Pins/Signals**		
▷ **Channel 0**	Disabled	
▷ **Channel 1**	Disabled	
⋮		
◢ **Channel 10**	Enabled	
Channel 10 single input	PTA7/FTM0_CH4/FB...	PTA7/FTM0_CH4/FB...
▷ **Channel 11**	Disabled	
⋮		
◢ **Trigger A**	Enabled	
Trigger A source	PDB0_CH0_TriggerA	PDB0_CH0_TriggerA
▷ **Trigger B**	Disabled	
◢ **Interrupts/DMA**		
Interrupt request	Disabled	
Interrupt priority	0 (Highest)	
ISR name	ADCInterrupt	
Conversion complete A interrupt	Enabled	
Conversion complete B interrupt	Disabled	
DMA request	Enabled	
◢ **Initialization**		
◢ **ADC part triggered by trigger A**		
Initial channel select A	Channel 10	
Differential mode A	Disabled	
◢ **ADC part triggered by trigger B**		
Initial channel select B	ADC disabled	
Differential mode B	Disabled	
Call Init method	no	

Figure 6. Init_ADC settings in Component Inspector

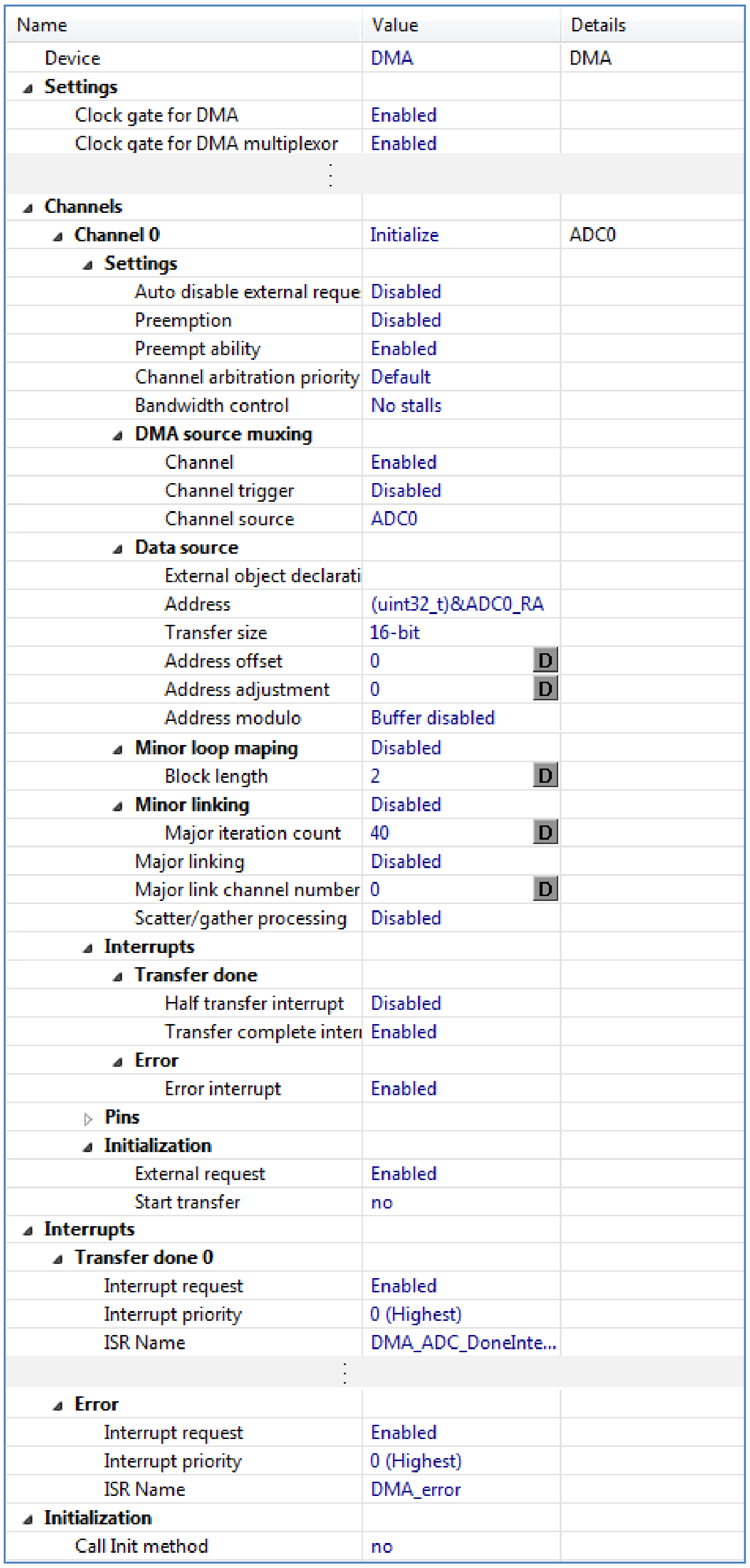
5.5 Init_eDMA

The enhanced direct memory access (eDMA) controller can perform complex data transfers with minimal intervention from a host.

In this application the eDMA is used to transfer the results of the ADC conversion to a location in the memory buffer. Individual transfers are triggered by a DMA request signal from the ADC.

In the example application the *Init_eDMA* component is configured to initialized eDMA the following way:

- Clock gate for DMA and DMA multiplexor are *Enabled.*
- Channel 0 is *initialized* with the following values:
  - DMA source muxing is *Enabled* and channel source is *ADC0.*
  - Data source is set to address of ADC0 Result register A: *(uint32_t)&ADC0_RA*. Address offset is 0, the values is still read from the same register.
  - Data destination is initially set to the beginning of the array *&MasuredValues and* Address offset is *2* (to increment address with 2 byte after transfer.
  - Block length is set to *2*, as 16-bits (2 bytes) are transferred.
  - Major iteration count is set to *40*. This is the length of one half of the buffer.
  - Transfer complete interrupt is *Enabled*.
  - Error interrupt is *Enabled.*
    - The interrupts have assigned ISR names: Transfer Done 0 ISR Name : *DMA_ADC_DoneInterrupt*
    - Error ISR Name : *DMA_error*



Name	Value	Details
Device	DMA	DMA
◢ **Settings**		
Clock gate for DMA	Enabled	
Clock gate for DMA multiplexor	Enabled	
⋮		
◢ **Channels**		
◢ **Channel 0**	Initialize	ADC0
◢ **Settings**		
Auto disable external reque	Disabled	
Preemption	Disabled	
Preempt ability	Enabled	
Channel arbitration priority	Default	
Bandwidth control	No stalls	
◢ **DMA source muxing**		
Channel	Enabled	
Channel trigger	Disabled	
Channel source	ADC0	
◢ **Data source**		
External object declarati		
Address	(uint32_t)&ADC0_RA	
Transfer size	16-bit	
Address offset	0 D	
Address adjustment	0 D	
Address modulo	Buffer disabled	
◢ **Minor loop maping**	Disabled	
Block length	2 D	
◢ **Minor linking**	Disabled	
Major iteration count	40 D	
Major linking	Disabled	
Major link channel number	0 D	
Scatter/gather processing	Disabled	
◢ **Interrupts**		
◢ **Transfer done**		
Half transfer interrupt	Disabled	
Transfer complete interr	Enabled	
◢ **Error**		
Error interrupt	Enabled	
▷ **Pins**		
◢ **Initialization**		
External request	Enabled	
Start transfer	no	
◢ **Interrupts**		
◢ **Transfer done 0**		
Interrupt request	Enabled	
Interrupt priority	0 (Highest)	
ISR Name	DMA_ADC_DoneInte...	
⋮		
◢ **Error**		
Interrupt request	Enabled	
Interrupt priority	0 (Highest)	
ISR Name	DMA_error	
◢ **Initialization**		
Call Init method	no	

Figure 7. Init_ADC settings in Component Inspector

6 Source Code

The archive AN4688SW.zip includes the full commented source code of the demo application.

The code generated by Processor Expert for individual components can be found in the folder *Generated code*. This folder is visible in Figure 2. The hand-written modules and modules intended for manual modification are placed in the *Sources* folder. What follows is a brief summary of the source files in the Sources folder:

- `ProcessorExpert.c` – This is the main Processor Expert module that contains the application code. The module supplies all of the component header files and header files required to define the information required by Processor Expert and any variables definitions. It also defines the `ShowData()` helper functions that display the data, and the `main()` function. The `main()` function first calls all of the individual initialization functions of the various components. After that, it spins in an endless loop where the data are displayed. For better visibility on the LCD screen, a delay is provided by displaying the contents of every tenth buffer.

```
  for(;;) {
        // check if DMA cycle is finished
        if (Measured) {
              cntr++;
              // display only time after time
              if (cntr > 10) {
                    // display data
                    ShowData();
                    cntr = 0;
              }
              // reset flag
              Measured = FALSE;
        }

  }
```

- `Events.c` – This file contains the interrupt handler for the DMA interrupt that is invoked when the DMA buffer fulfills up:

```
PE_ISR(DMA_ADC_DoneInterrupt)
{
  // Clear all DMA interrupt flags
  DMA_CINT = 0x40;
  // switch bufstart to other part of buffer
  BufStart = ADC_BUFFER_SIZE - BufStart;
  // reset destination address
  DMA_TCD0_DADDR = (uint32_t)&(MeasuredValues[BufStart]);
  // toggle pin
  GPIO1_ToggleFieldBits(GPIOPtr, TST,1);
  // toggle flag
  Measured = TRUE;
}
```

There is also very simple handler for the DMA error interrupt that doesn't do any specific action except clearing the DMA interrupt flags.

```
PE_ISR(DMA_error)
{
  // Clear all DMA interrupt flags
  DMA_CINT = 0x40;
}
```

- `My.h` – contains basic definitions and external references to objects shared with event code in *Events.c* module.

```
// current buffer with valid values
extern volatile uint16_t BufStart;
// double buffer for values
extern uint16_t MeasuredValues[];
// flag for end of measurement
extern bool Measured;
// GPIO component reference (for testpin)
extern LDD_TDeviceData *GPIOPtr;

// size of the buffer
// must correspond to Init_EDMA property channels / channel 0 / Settings /
Major iteration count
#define ADC_BUFFER_SIZE 40
```

7 Conclusion

The ADC, PDB and eDMA peripherals of the Kinetis MCU provide the user the powerful means to implement various analog measurement tasks without need of any intervention from the MCU core. Processor Expert provides a valuable means to speed the development of initialization and peripheral driver code within its visual user interface.

8 Revision History

Table 2. Revision History

Rev. Number	Date	Substantive Change
A	1/31/2013	Initial creation

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4688
Rev. A
01/2013