Freescale Semiconductor
Application Note

Document Number:AN4631
Rev. 0, 12/2012

Using the Asynchronous DMA features of the Kinetis L Series

by: Chris Brown

Contents

1 Introduction

The power consumption of devices and the implications around designing an application for low power are common topics. In modern MCU systems, a common way to reduce the current consumption of the MCU is to reduce the number of transistors that are switching. This in turn means powering down peripherals and /or the core whenever possible. In most MCU systems, this results in an MCU that is essentially static and can have very limited interaction with the outside world. This application note discusses how to use the asynchronous DMA feature of the Kinetis L series of microcontrollers to enable interaction with the outside world, while still reducing the number of transistors that are switching, thus saving power in an application.

This application note will discuss ways to achieve this goal using the Kinetis L series microcontrollers and the asynchronous DMA feature.

It is important to indicate that this application note is based on the KL25Z128VLK4 microcontroller, so some features might not be available in all Kinetis L series of devices.

2 Asynchronous features

Asynchronous features are, for the purposes of this document, features that operate without a clock or with a clock that does not have to be synchronized to the core clock (that is, does not have to be clocked from the core clock, or a divided down copy of the core clock or a multiple of the core clock frequency). The asynchronous DMA provides the most significant benefit toward the goal of providing interaction with the outside world while limiting power consumption, as this feature allows data to be saved from, or copied to registers of peripheral modules. However, to be able to take advantage of the asynchronous DMA feature, the part must have other asynchronous features besides just the DMA. The Kinetis L series provides many asynchronous features to facilitate low power consumption while still performing useful tasks. Other asynchronous peripherals of the part include:

- Analog-to-Digital Converter (ADC)
- Inter-Integrated Circuit (I2C)
- Universal Asynchronous Receiver Transmitter (UART)
- Digital-to-Analog Converter (DAC)
- Timer Pulse Module (TPM)
- Low-Power Timer (LPTMR)
- Serial Peripheral Interface (SPI)

The following sections will discuss how to use the asynchronous DMA feature and how this feature reduces power consumption. In addition, a real world application of using the asynchronous DMA feature of the L series will be provided.

3 Configuring an application for asynchronous DMA operation

Configuring an application for asynchronous DMA operation consists of two tasks:

- Configuring the DMA
- Configuring the peripheral module(s) to be used with the asynchronous DMA.

The following section discusses the DMA configuration.

3.1 Configuring the DMA for asynchronous operation

Configuring the DMA for asynchronous operation is enabled by simply setting the Enable Asynchronous DMA Requests bit (DMA_DCRn[EADREQ] = 1). All other bits and registers must be configured as they would be for synchronous operation. Follow these general guidelines when configuring the DMA for asynchronous operation.

However, the user must be careful to not use PIT gated transfers when planning to use asynchronous DMA operations in power modes lower than Very Low-Power Wait (VLPW) as the PIT will not operate in power modes lower than VLPW mode. If the application requires a gated transfer and asynchronous operation, the cycle-steal functionality must be implemented as this function will force the DMA channel to perform one transfer, and wait for another trigger event before performing another transfer. This function will be demonstrated in the example application showcased in this application note.

3.2 Configuring modules for asynchronous DMA operation

Each module will be configured differently for different types of applications. Therefore, there is no defined set of directions that can be used for every module. However, there are certain guidelines that the user must follow to achieve operation with the asynchronous DMA. First of all, the following listed modules that are capable of generating asynchronous DMA requests, must be reviewed.

- UART0 (TX)
- UART0 (RX)
- TPM (Channel interrupt)
- TPM (Overflow)
- ADC
- CMP
- PORTA
- PORTD

NOTE

These modules are specific to the KL25 device. The specific chip being used may or may not have more or less modules that are capable of generating asynchronous DMA requests. In addition, only some of the channels of the timers may be capable of generating asynchronous DMA requests. It is recommended to consult the device-specific reference manual available on **freescale.com** when choosing to use a specific peripheral with asynchronous DMA requests.

When configuring the desired modules for operation with the asynchronous DMA functionality, follow these general guidelines:

1. Enable the clock to be used. This clock must be able to source the module and operate in the low-power mode which is to be used. Note that not all the peripherals such as PORTA, PORTD, and CMP, may need a clock enabled.
2. Disable the DMA channel to be used.
3. Disable the module being configured.
4. Configure the module in the same manner as for interrupt operation.
5. Configure the module to use the clock that the user has enabled for this application (if applicable).
6. Set the appropriate DMA enable bit in the module (for example, ADC0_SC2[DMAEN] = 1).
7. Enable the module.
8. Enable the DMA channel being used for this module.
9. Enter the desired low-power mode.

4 Application of asynchronous DMA operation

The real world example that has been chosen to showcase is an application where the brightness of an LED is varied from 0-100% according to a user input. The user input must also be recorded and the average and average change should be calculated and output to a terminal. This is a moderately complex application that will require the use of asynchronous DMA to achieve the lowest possible power consumption. The block diagram of the application is shown in the following figure.

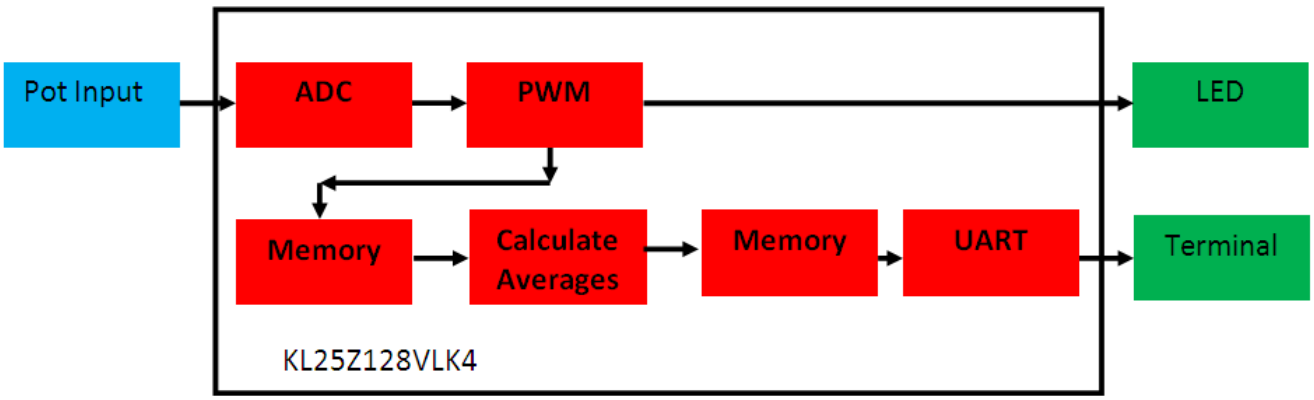

Figure 1. Asynchronous DMA application block diagram

The requirements of the application (from the standpoint of the microcontroller) are as follows:

- Measure the voltage of the potentiometer wiper.
- Output a 50 Hz PWM.
- Vary the pulse width of a PWM signal based on the ADC results.
- Store the ADC readings in a memory array.
- Use the asynchronous DMA to perform all data transfers between the peripherals/memory.
- Compute the average and average derivative of the measurements.
- Display the average and average derivative of the measurements to a terminal output.
- Utilize Very Low-Power Run (VLPR) mode.

The code for this application is structured into three different parts:

- Initialization
- Low-power mode
- Calculation

Each of these parts will be discussed in detail in the following sections.

NOTE

This application utilizes ProcessorExpert 10.0 and IAR to generate driver code.

4.1 ADC and TPM configuration

The first components of the block diagram to be implemented are the ADC and PWM as these components are critical to the operation of the system. As described in the requirements list, the PWM output must be set up to produce a 50 Hz signal in a low-power mode. The TPM module of the L series is well suited for this task as it generates PWM signals without the CPU intervention and is available in low-power modes. The next task is to select the clock which sources the module. There are four choices for the TPM clock:

- MCGIRCLK
- OSCERCLK
- MCGFLLCLK
- MCGPLLCLK

The lowest power of these that will be available in low power modes is the MCGIRCLK. Therefore, this clock will be used.

When configuring the ADC, always consider the conversion resolution, conversion time, and whether the application calls for differential inputs or not. For simplicity, it will be assumed that this application will not operate in a high-noise environment, and thus will not require differential inputs.

Starting with the conversion resolution, recall that the example application required the LED brightness to be varied from 0–100%. This, in effect means that the current delivery to the LED must range from 0–100%. The duty cycle of the PWM (output by the TPM) is controlled by the value written to the 16-bit channel value register. Writing a 0x0000 to the channel value register results in a PWM with a 0% duty cycle and writing 0xFFFF to the channel value register results in a PWM with a 100% duty cycle. Therefore, the ADC will be configured for 16-bit conversions.

When selecting the conversion time, the user must remember that the goal of this application is to be low power. There are essentially two possible courses of action to achieve low-power:

- Run the part slower so that less power is consumed.
- Run the part as fast as possible and put the part in a low-power state so that less power is consumed.

There are no requirements in this application that restrict when or how long the user must wait between the samples. Therefore, it is expected that the part may utilize the low power modes and may do so for significant amounts of time. Thus, the fastest conversion time possible will be selected. Given that the ADC will be using its own internal clock (because it will be operating in an asynchronous mode), the fastest conversion that can be performed is 10.42 µs conversions. Using a fast conversion time will also alleviate errors due to changes in the input.

4.2 DMA configuration

The first variable to consider when using DMA for this application is the number of DMA channels that will be needed. There are six boxes inside the KL25Z128VLK4 device (See Figure 1):

- ADC
- PWM (TPM)
- Two Memory boxes (one will be an array; the other will hold the results of the calculations)
- Calculations (calculation of the average and average change)
- UART output

In addition, several of these components are connected, indicating that they need to interact with each other. Therefore, it is important to understand which components/connections will not need intervention by the CPU or DMA.

- The ADC can perform conversions asynchronously without help from the CPU or DMA, so the connection from the potentiometer will not need DMA intervention.

  The same is true for the TPM PWM output. It occurs asynchronously without intervention, therefore the connection to the LED will not need a DMA channel.

- The UART can also transmit asynchronously, so the UART connection to the terminal will not require a DMA channel.
- However, data cannot get to the UART without intervention from either the CPU or DMA. The calculations must be performed by the CPU. Since the CPU will be active for this action, it can be used to get the data from memory and store the results in memory. Therefore, these connections will not require DMA.

So, now the connections from the ADC to the TPM, from the TPM to memory, and from the memory to the UART, are remaining. Each of these connections will be discussed as follows.

- The ADC to TPM DMA configuration is fairly straight forward. The application requires that each ADC result be moved to the TPM channel value register of the appropriate channel. Therefore, the ProcessorExpert (PEx) configuration is shown in the following figure.

Component Inspector - DMAT_ADC

Properties | Methods | Events

Name	Value	Details
Component name	DMAT_ADC	
DMA controller device	DMA1	
Channel		
Channel select	Fixed	
Channel	DMA_Channel0	DMA_Channel0
Interrupts	Enabled	
Allocate channel	yes	
Trigger		
Trigger source type	Peripheral device	
Trigger source	ADC0_DMA_Request	ADC0_DMA_Request
Periodic trigger	Disabled	
Data source		
External object declaration		
Address	(uint32_t)&ADC0_RA	
Transfer size	16-bit	
Address offset	0 D	
Circular buffer	Buffer disabled	
Data destination		
External object declaration		
Address	(uint32_t)&TPM2_C0V	
Transfer size	16-bit	
Address offset	0 D	
Circular buffer	Buffer disabled	
Data size		
External object declaration		
Value	0x80	
Transfer control	Cycle-steal	
Disable after transfer	no	
Asynchronous requests	Enabled	
Channel linking	Enabled	
Linked channel	DMA_Channel1	DMA_Channel1
Linking after request service	Enabled	
Linked channel	DMA_Channel1	DMA_Channel1
After request complete	No action	
After transfer complete	No action	
Initialization		
Auto initialization	yes	
Half complete	Disabled	
Event mask		

Figure 2. PEx ADC to TPM DMA configuration

Notice that the trigger mechanism that has been chosen is the ADC0 DMA request. With this configuration, the ADC will trigger a DMA request once a conversion is complete.

For the data source and destination, the ADC0 result register and TPM2 channel 0 value register, respectively, have been specified. This means that the ADC result will be transferred to the TPM channel value register. Also notice that the source and destination buffers and address offsets have been disabled. This sets up the DMA so that the same source and destination locations will be used for each transfer.

In the transfer control section, the cycle-steal option has been selected. This prevents the DMA from continuously transferring from the source location to the destination location and forces only one transfer per DMA request.

Channel linking has also been enabled. This option was selected to generate a request to another DMA after every transfer by this module. This transfer will in turn, trigger the DMA transfer from the TPM Channel Value Register to memory. This ensures that every ADC conversion will be saved into memory.

- The ProcessorExpert configuration of the DMA channel from the TPM channel value register to memory is shown in the following figure.

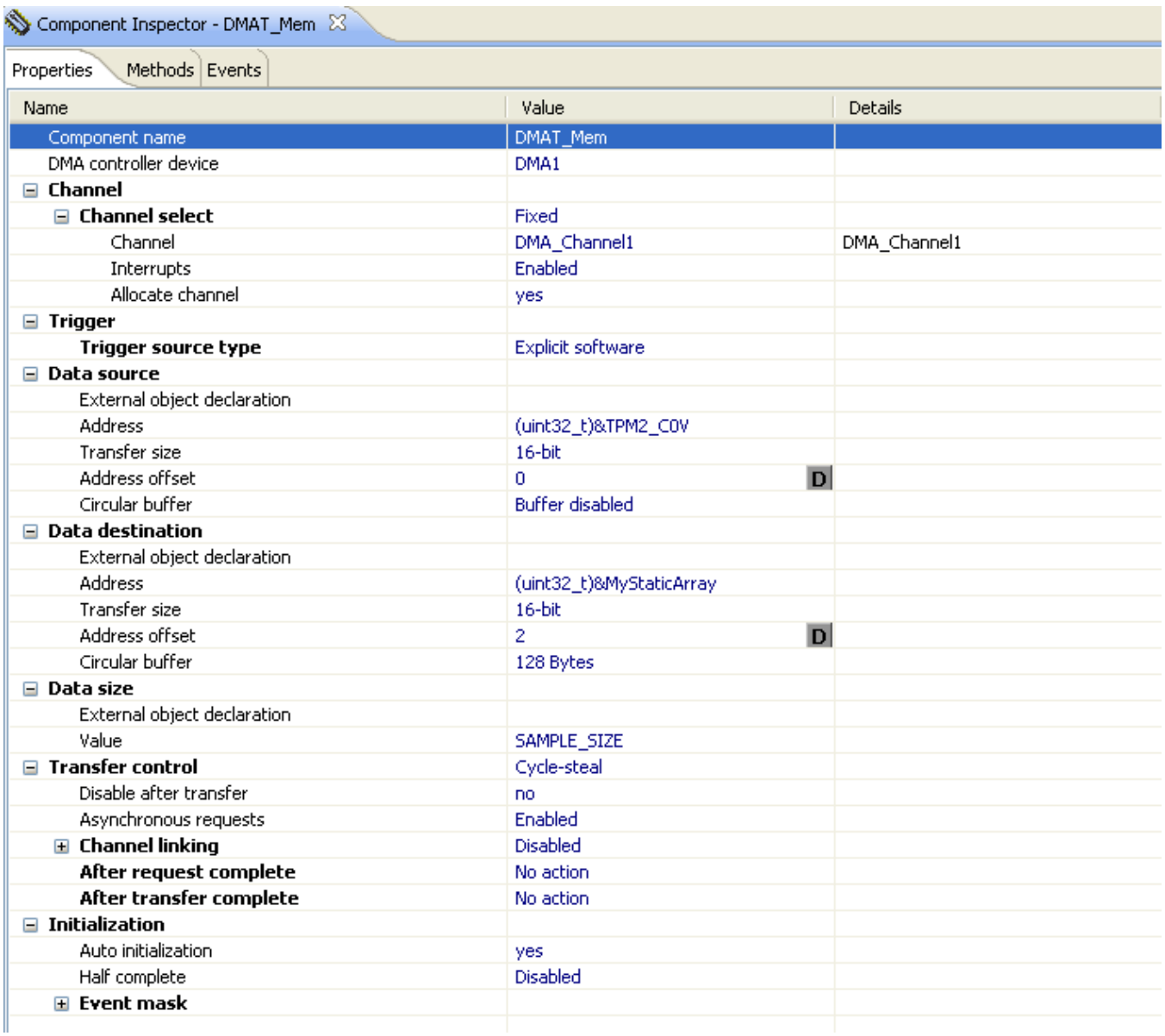

Figure 3. PEx TPM to memory DMA configuration

Remember that this DMA transfer is to be linked from the ADC to TPM transfer. Therefore, no hardware trigger is necessary and the user can select explicit software for trigger source type. Notice that TPM2 Channel 0 Value register is selected for the source (with buffer disabled) and the destination is an array memory location (with a buffer enabled). This allows a static array to be declared in the program and the DMA will transfer the TPM2 channel 0 values from this register to the memory array that has been designated.

NOTE

When using the buffer option and a user-defined array, it is very important that the array be properly aligned in the user software when defining the array. Otherwise, the DMA may write to locations outside of the defined array, which can cause unexpected results.

Also notice that the cycle-steal option has been selected for this DMA transfer. This will effectively synchronize these transfers with the ADC conversion transfers ensuring that each ADC result is saved for calculation.

- The ProcessorExpert configuration of the DMA channel from the TPM channel value register to memory is shown in the following figure.

Component Inspector - DMAT_UART_OUT

Properties | Methods | Events

Name	Value	Details
Component name	DMAT_UART_OUT	
DMA controller device	DMA1	
Channel		
Channel select	Fixed	
Channel	DMA_Channel2	DMA_Channel2
Interrupts	Disabled	
Allocate channel	yes	
Trigger		
Trigger source type	Peripheral device	
Trigger source	UART0_Transmit_DMA_Request	UART0_Transmit_DMA_Request
Periodic trigger	Disabled	
Data source		
External object declaration	#include "Events.h"	
Address	(uint32_t)&MyOutputArray	
Transfer size	8-bit	
Address offset	1 D	
Circular buffer	Buffer disabled	
Data destination		
External object declaration		
Address	(uint32_t)&UART0_D	
Transfer size	8-bit	
Address offset	0 D	
Circular buffer	Buffer disabled	
Data size		
External object declaration		
Value	OUTPUT_ARRAY_SIZE	
Transfer control	Cycle-steal	
Disable after transfer	yes	
Asynchronous requests	Enabled	
Channel linking	Disabled	
After request complete	No action	
After transfer complete	No action	
Initialization		
Auto initialization	yes	
Half complete	Disabled	
Event mask		

Figure 4. PEx Memory to UART DMA configuration

The memory to UART DMA transfer is the least complicated of the DMA configurations in this project. It is configured to move data from a user defined array (MyOutputArray) to the UART D register. No buffers are necessary for this operation. Once the DMA channel has been properly configured and enabled, the UART immediately begins requesting data (provided it has been configured and enabled already) as the Transmit Data Register Empty flag is clear (UARTx_S1[TDRE]). The DMA continues transferring as soon as the UARTx_D register becomes empty until the byte count register decrements to zero. At this point, the hardware requests are disabled as specified by the ProcessorExpert configuration.

Notice that the cycle-steal option has again been selected. If this option were not selected, then as soon as the UARTx_D register is written, the DMA would transfer all of the data in the output array to the D register at once. This would in turn cause the UART to output only the first character in the output array. This is because the user must wait for the transmit register to become empty before writing another character. The cycle-steal option allows this to happen.

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 10 5879 8000
support.asia@freescale.com

Document Number: AN4631
Rev. 0, 12/2012