Freescale Semiconductor
Application Note

Document Number:AN4598
Rev. 0, 10/2012

Using DMA Transfers with Enhanced Flexible PWM on MC56F84xxx

by: Pavel Sustek

Contents

1 Introduction

Controlling electric motors or different Switched Mode Power Supplies (SMPS) topologies requires computation power of a controller core together with powerful and flexible peripherals. Freescale MC56F84xxx Digital Signal Controllers (DSCs) contain such PWM module that provides sufficient flexibility to generate various switching patterns, including highly sophisticated waveforms. To maximize system performance, a Direct Memory Access (DMA) peripheral can be used for data transfers to and from the peripheral.

This application note deals with how to properly set the Freescale eFlex PWM peripheral to enable writing to value registers and reading from capture registers using DMA channels and the DMA configuration enables reading of Analog-to-Digital Converter (ADC) result registers. The application note is supported by the application code that provides ready-to-use functions. The document provides additional information based on MC56F84xxx Reference Manual with main focus on the use of application.

2 Peripherals

The DMA controller supports most of DSC's MC56F84xx peripherals. This document focuses only on eFlex PWM and ADC modules that are main peripherals in motor control and

SMPS applications. Following sections describe basic features of the modules. The detail information can be found in MC56F84xxx Reference Manual.

2.1 Enhanced Flexible Pulse Width Modulator

The pulse width modulator contains PWM submodules, each of which is set up to control a single half-bridge power stage.

Selected features include:

- 16 bits of resolution for center, edge aligned, and asymmetrical PWMs
- Fractional delay for enhanced resolution of the PWM period and edge placement
- PWM outputs that can operate as complementary pairs or independent channels
- Ability to accept signed numbers for PWM generation
- Double buffered PWM registers
  - Integral reload rates from 1 to 16
  - Half cycle reload capability
- Multiple output trigger events can be generated per PWM cycle via hardware
- Programmable filters for fault inputs
- Independently programmable PWM output polarity
- Independent top and bottom deadtime insertion
- PWM_X pin can optionally output a third PWM signal from each submodule
- Channels not used for PWM generation can be used for buffered output compare functions
- Channels not used for PWM generation can be used for input capture functions
- Enhanced dual edge capture functionality
- External ADC input, taking into account values set in ADC high and low limit registers
- Each submodule can request a DMA read access for its capture FIFOs and a DMA write request for its double buffered VALx registers

Figure 1 shows PWM submodule block diagram.

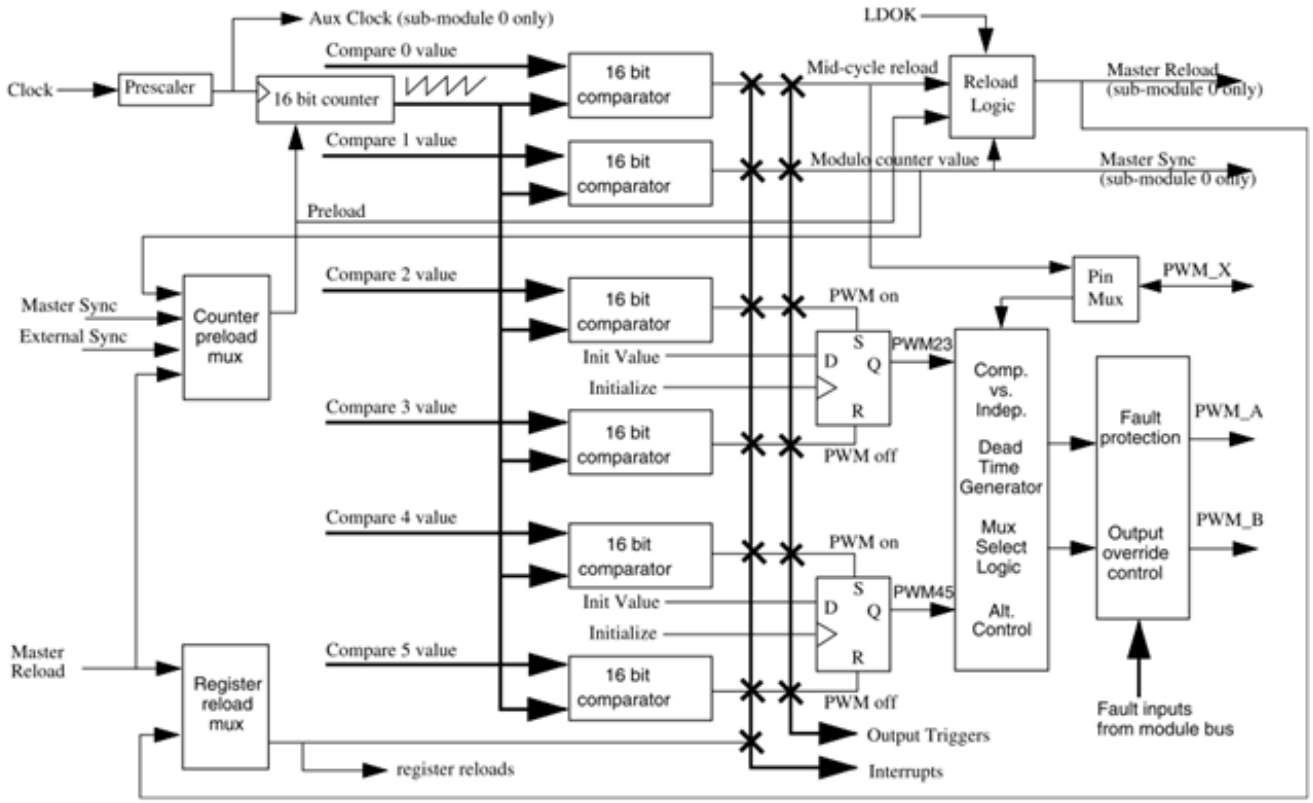

Figure 1. PWM submodule block diagram

2.2 DMA controller

The DMA controller module enables fast transfers of data, providing an efficient way to move blocks of data with minimal processor interaction. The DMA module, shown in Figure 2, has four channels that allow byte, word, or longword data transfers. Each channel has a dedicated source address register (SARn), destination address register (DARn), status register (DSRn), byte count register (BCRn), and control register (DCRn). Collectively, the combined program-visible registers associated with each channel define a transfer control descriptor (TCD). All transfers are dual address, moving data from a source memory location to a destination memory location with the module operating as a 32-bit bus master connected to the system bus. The programming model is accessed through a 32-bit connection with the slave peripheral bus. DMA data transfers may be explicitly initiated by software or by peripheral hardware requests.

The DMA controller module features:

- Four independently programmable DMA controller channels
- Dual-address transfers via 32-bit master connection to the system bus
- Data transfers in 8-, 16-, or 32-bit blocks
- Continuous-mode or cycle-steal transfers from software or peripheral initiation
- One programmable input selected from 16 possible peripheral requests per channel
- Automatic hardware acknowledge/done indicator from each channel
- Independent source and destination address registers
- Optional modulo addressing and automatic updates of source and destination addresses
- Independent transfer sizes for source and destination
- Optional auto-alignment feature for source or destination accesses
- Optional automatic single or double channel linking
- Programming model accessed via 32-bit slave peripheral bus
- Channel arbitration on transfer boundaries using fixed priority scheme

Figure 2 shows a simplified block diagram of the 4-channel DMA controller.

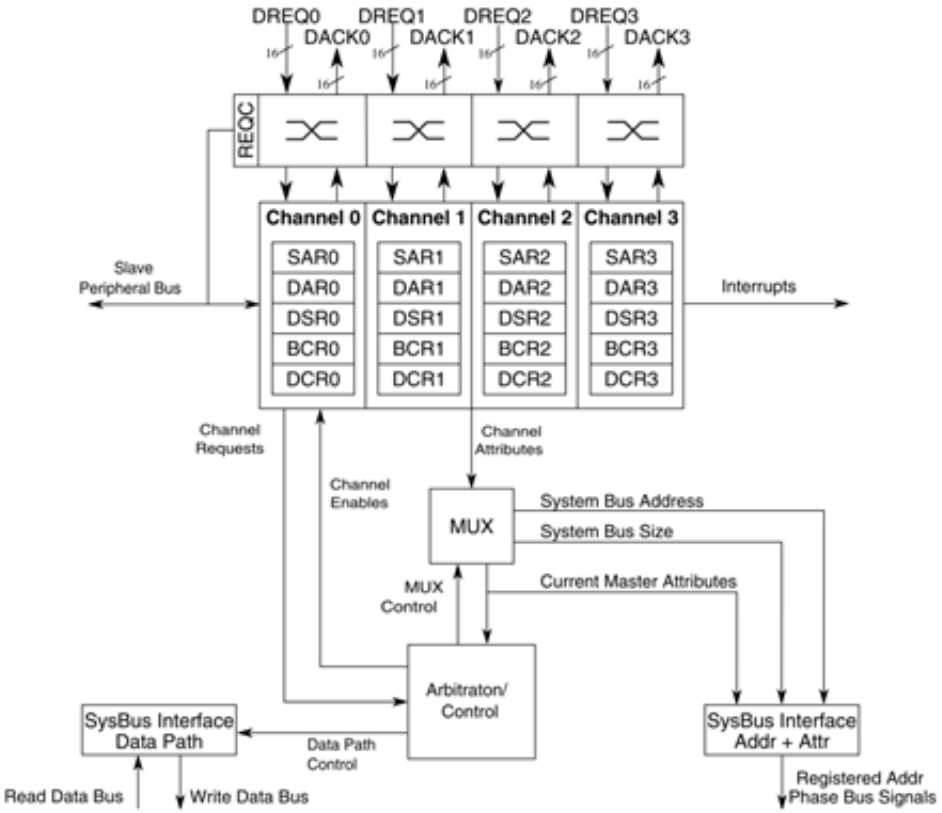

Figure 2. DMA controller block diagram

2.3 12-bit cyclic analog-to-digital converter (ADC)

The analog-to-digital (ADC) converter function consists of two separate analog-to-digital converters, each with eight analog inputs and its own sample and hold circuit. A common digital control module configures and controls the functioning of the converters. ADC selected features include:

- 12-bit resolution
- Designed for maximum ADC clock frequency of 20 MHz with 50 ns period
- Sampling rate up to 6.67 million samples per second
- Can be synchronized to other peripherals that are connected to an internal Inter-Peripheral Crossbar module, such as the PWM, through the SYNC0/1 input signal
- Sequentially scans and stores up to 16 measurements
- Scans and stores up to eight measurements each on two ADC converters operating simultaneously and in parallel
- Optional DMA function to transfer conversion data at the end of a scan or when a sample is ready to be read.
- Signed or unsigned result

Figure 3 illustrates the dual ADC configuration:

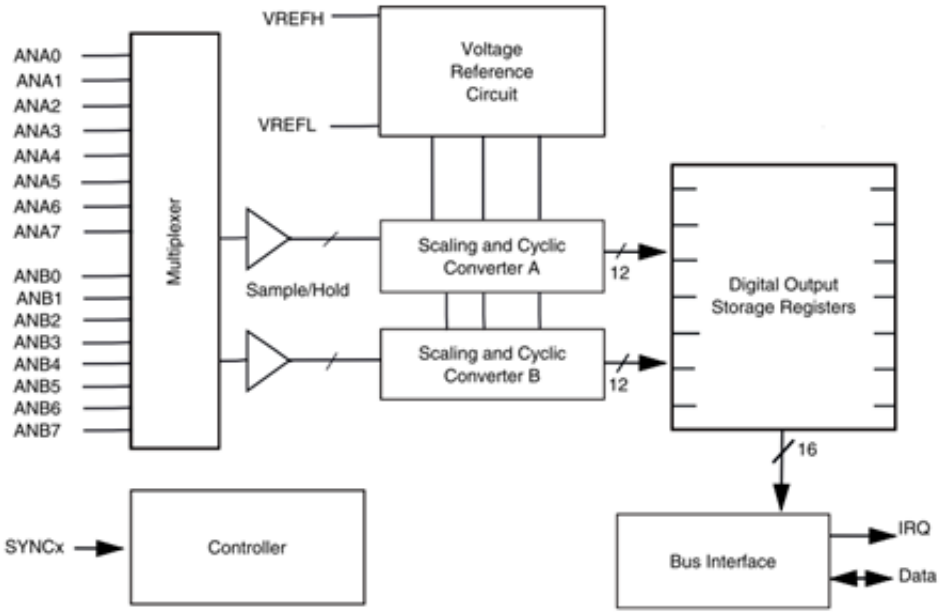

Figure 3. Cyclic ADC block diagram

3 Application modes

3.1 Writing PWM value registers using DMA controller

The eFlex PWM module contains DMA Enable Register which allows setting of DMA requests for reading or writing. Setting VALDE (Value Register DMA Enable) bit enables DMA write requests for VALx and FRACVALx registers when Reload Flag is set.

To have proper data transfer between the data memory (variable structure PWMA_REG_UPDATE) and PWM value registers, it is essential to configure dedicated DMA channel.

The application example demonstrates how to update PWM submodule value registers in each PWM reload period using one DMA channel. The basic block diagram of the transfer is shown in Figure 4.

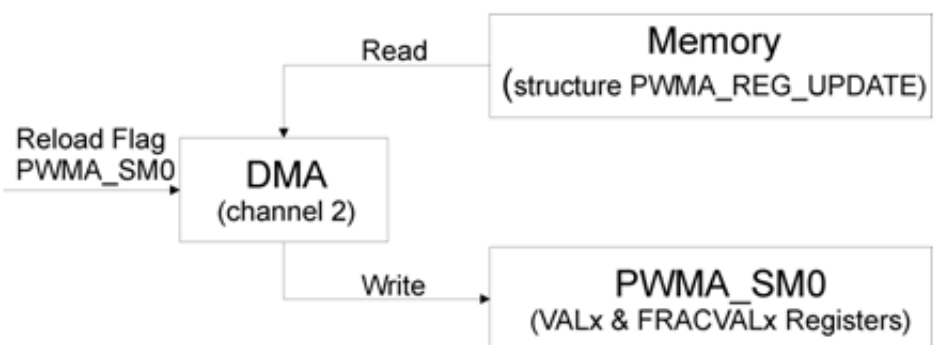

Figure 4. Writing PWM value registers using DMA transfer

eFlex PWM A Submodule 0 Configuration

The configuration of DMA channel 2 is performed in DMA2_Init function in DMA_cfg.c file which is part of Sources folder in the following order:

1. Enable clock to GPIO_E where PWMA_A0 and PWMA_ B0 signals are terminated
   - *SIM_PCE0 |= SIM_PCE0_GPIOE;*
2. Enable PWM peripheral on GPIO_E pins (pin GPIOE0 and GPIOE1)
   - *GPIOE_PER |= (GPIOE_PER_PE_0 | GPIOE_PER_PE_1);*
3. Enable clock to PWMA Submodule 0
   - *SIM_PCE3 |= SIM_PCE3_PWMACH0;*
4. Select full reload cycle for SM0
   - *PWMA_SM0CTRL = PWMA_SM0CTRL_FULL;*
5. Select local reload signal for SM0
   - *PWMA_SM0CTRL2 = PWMA_SM0CTRL2_FORCE_SEL_1;*
6. Output PWM frequency is set to 16 kHz → 62.5 µs = 10 ns (cycle) * 3125 * 2
   - *PWMA_SM0INIT = –3125;*
   - *PWMA_SM0VAL1 = 3124;*
7. Initialize duty cycle is set to 50%
   - *PWMA_SM0VAL2 = –(PWMA_SM0VAL1>>1);*
   - *PWMA_SM0VAL3 = PWMA_SM0VAL1>>1;*
8. Deadtime is set to 1 µs
   - *PWMA_SM0DTCNT0 = 100;*
   - *PWMA_SM0DTCNT1 = 100;*
9. All PWM faults are disabled for this example application
   - *PWMA_SM0DISMAP0 = 0;*
   - *PWMA_SM0DISMAP1 = 0;*
10. PWM SM0 outputs PWMA_A0 and PWMA_B0 are enabled
    - *PWMA_OUTEN = (PWMA_OUTEN_PWMA_EN_0 | PWMA_OUTEN_PWMB_EN_0);*
11. DMA write value register request is enabled
    - *PWMA_SM0DMAEN = PWMA_SM0DMAEN_VALDE;*
12. Trigger 0 signal is enabled for PWM and ADC synchronization through XBAR peripheral
    - *PWMA_SM0TCTRL |= PWMA_SM0TCTRL_OUT_TRIG_EN_0;*
13. PWM start sequence is performed
    - *PWMA_MCTRL |= PWMA_MCTRL_CLDOK;*
    - *PWMA_MCTRL |= PWMA_MCTRL_LDOK;*
    - *PWMA_MCTRL |= PWMA_MCTRL_RUN;*

The PWMA SM0 generates a PWM pair of complementary signal with added deadtime. The PWM input clock is IPBCLK 100 MHz.

The PWMA_REG_UPDATE structure defined in PMW_A_cfg.h file consists of variables where the data for PWM value registers are stored. The variables are defined in the same order as PWM value registers in a flash memory to allow simple linear DMA transfer. The PWM configuration is performed during peripheral initialization calling PWM_A_Init function with pointer to the structure as a function parameter.

DMA Channel 2 Configuration

The configuration of DMA channel 2 is performed in DMA2_Init function in DMA_cfg.c file which is part of Sources folder in the following order:

1. Set Source Address Register to value of the structure address which is passed from input pointer
   - *DMA_SAR2 = ((uint32_t)(ptr)<<1);*
2. Set Destination Address Register to value of PWMA_SM0VAL0 register address
   - *DMA_DAR2 = ((uint32_t)(&PWMA_SM0VAL0)<<1);*
3. Clear state machine for DMA channel 2
   - *DMA_REQC |= DMA_REQC_CFSM2;*
4. Set request source for the channel – PWMA SM0 value write request (11). This defines the logical connection between the DMA requesters and the DMA channel 2
   - DMA_REQC |= DMA_REQC_DMAC2_3 | DMA_REQC_DMAC2_1 | *DMA_REQC_DMAC2_0;*
5. Enable peripheral request for the DMA channel

- *DMA_DCR2 |= DMA_DCR2_ERQ;*

6. Set size of source and destination data as WORD (16-bit) and increasing source and destination addresses after each DMA transfer
   - *DMA_DCR2 |= DMA_DCR2_SINC | DMA_DCR2_DINC | DMA_DCR2_SSIZE_1 | DMA_DCR2_DSIZE_1;*
7. Enable DMA transfer completed interrupt (vector 34) with priority 1
   - *DMA_DCR2 |= DMA_DCR2_EINT;*
   - *INTC_IPR3 |= INTC_IPR3_DMACH2_1;*
8. Clear channel status register
   - *DMA_DSR_BCR2 |= DMA_DSR_BCR2_DONE;*
9. Set number of bytes to be transferred – for 11 16-bit registers it equals 22
   - *DMA_DSR_BCR2 |= (DMA_DSR_BCR2_BCR & 22);*

DMA controller expects a byte address of source and destination address register. The peripheral registers as well as data memory are addressed in terms on 16-bit words. Because of this the SAR and DAR register values are doubled.

Transfer Process

The DMA request is generated from PWMA SM0 once Reload Flag (RF) is set. This is performed every reload period which corresponds to PWM cycle (62.5 µs) for this example. The request starts DMA transfer of eleven 16-bit variable values stored in PWMA_REG_UPDATE structure to PWMA SM0 registers VAL0–5 and FRACVAL1–5. As soon as the transfer is completed, the DONE flag is set and the interrupt service routine (ISR) is invoked.

The DMA2 ISR calls DMA2_Preset function that performs:

1. Read actual values of PWM SM0 VALx and FRACVALx registers
2. Change values of some variable in structure just for demonstration purposes to see that modified values are transferred to the PWM submodule registers
3. Reset SAR and DAR registers because they contain addresses of last 16-bit variable and PWM register
4. Clear status register setting DONE bit
5. Set number of byte to be transferred in next cycle—the number is decreased after each byte is transferred to zero

Concurrently with a transfer completion, the RF flag is cleared and LDOK (Load OK) bit is set automatically to allow synchronous update of PWM parameters. At the beginning of next PWM reload cycle, VAL and FRACVAL register values are used by the PWM generator.

The DMA transfer diagram is shown in Figure 5 :

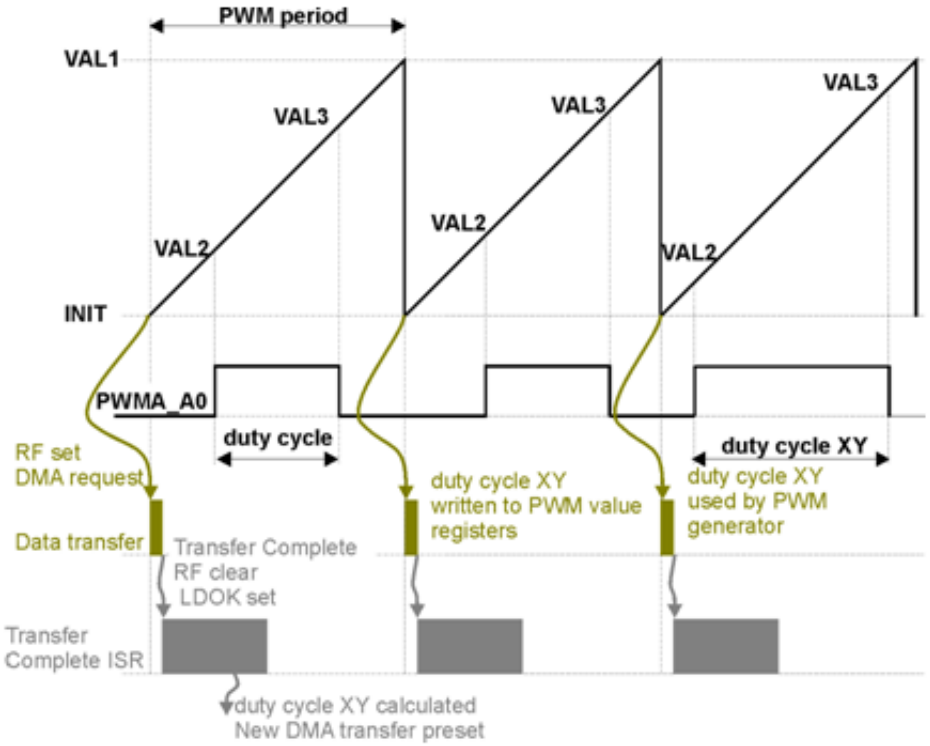

Figure 5. Time diagram of PWM DMA write request processing

From the diagram it is obvious that the PWM VALx and FRACVALx registers are updated with the delay of one PWM reload period cycle. This can influence time critical applications that require the update in a successive period. Setting LDMOD bit in PWM CTRL register allows immediate register update once LDOK is set. Then the delay is caused only by data transferring (time between RF and LDOK are set).

3.2 Reading PWM capture registers using DMA controller

Each PWM submodule contains six capture registers with one-level FIFO (CVAL0 – 5). Each pair of registers is assigned to a PWM pin (CVAL0&1–PWM_X, CVAL2&3–PWM_A, CVAL4&5–PWM_B). It means that during one PWM period maximum six values can be captured, two on each PWM pin.

The PWM module DMA Enable Register contains bit field supporting DMA requests for capture registers reading. Setting CAxDE, CBxDE, or CXxDE allows selection of a Capture FIFO that generates DMA request for reading. If more than one Capture FIFO DMA request is enabled, the bit field FAND (FIFO Watermark AND Control) determines whether selected FIFO Watermarks are OR'ed or AND'ed together. It works only in conjunction with the bit field CAPTDE (Capture DMA Enable Source Select) that selects the read DMA request source:

00—Disable

01—Exceeding a FIFO watermark sets the request (at least one of CxxDE bit must be set)

10—A local sync (VAL1 matches counter) sets the request

11—A local reload (STS[RF]) sets the request

To have proper data transfer between the PWM capture registers and data memory (variable structure PWMB_REG_READ) it is essential to configure dedicated DMA channel.

The application example demonstrates how to read PWM capture registers every PWM period (a local sync signal) using one DMA channel. The basic transfer block diagram is shown in Figure 6.

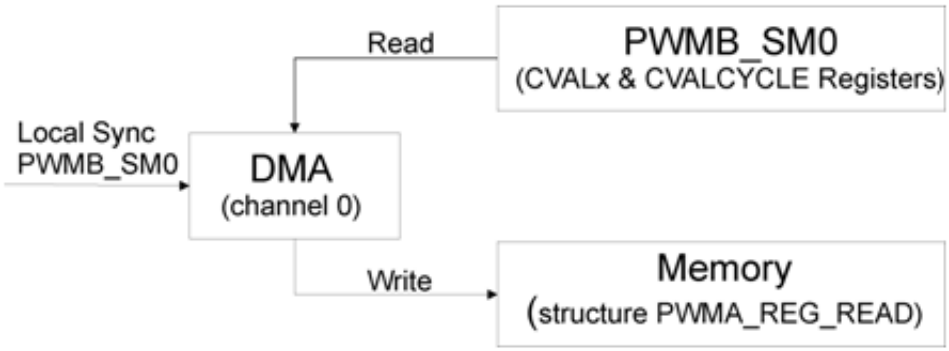

Figure 6. Reading PWM capture registers using DMA transfer

eFlex PWM B Submodule 0 Configuration

The configuration of PWM B SM0 is performed in PWM_B_Init function in PWM_B_cfg.c file which is part of Sources folder in the following order:

1. Enable clock to GPIO_G where PWMB_A0, PWMB_ B0 and PWMB_ X0 signals are terminated
   - *SIM_PCE0 |= SIM_PCE0_GPIOG;*
2. Enable PWM peripheral on GPIO_G pins (pin GPIOG2, GPIOG3 and GPIOG8)
   - *GPIOG_PER |= (GPIOG_PER_PE_2 | GPIOG_PER_PE_3 | GPIOG_PER_PE_8);*
3. Enable clock to PWMB Submodule 0
   - *SIM_PCE3 |= SIM_PCE3_PWMBCH0;*
4. Select full reload cycle for SM0
   - *PWMB_SM0CTRL = PWMB_SM0CTRL_FULL;*
5. Select local reload signal for SM0
   - *PWMB_SM0CTRL2 = PWMB_SM0CTRL2_FORCE_SEL_1;*
6. Output PWM frequency is set to 10 kHz→100 µs = 10 ns (cycle) * 5000 * 2
   - *PWMB_SM0INIT = –5000;*
   - *PWMB_SM0VAL1 = 4999;*
7. Deadtime is set to 0 µs
   - *PWMB_SM0DTCNT0 = 0;*
   - *PWMB_SM0DTCNT1 = 0;*
8. All PWM faults are disabled for this example application
   - *PWMB_SM0DISMAP0 = 0;*
   - *PWMB_SM0DISMAP1 = 0;*
9. Capture mode is set to capture rising edges in even registers (CVAL0,2,4) and falling edges in odd registers (CVAL1,3,5)
   - *PWMB_SM0CAPTCTRLA |= PWMB_SM0CAPTCTRLA_EDGA0_0 | PWMB_SM0CAPTCTRLA_EDGA1_1;*
   - *PWMB_SM0CAPTCTRLB |= PWMB_SM0CAPTCTRLB_EDGB0_0 | PWMB_SM0CAPTCTRLB_EDGB1_1 ;*
   - *PWMB_SM0CAPTCTRLX |= vPWMB_SM0CAPTCTRLX_EDGX0_0 | PWMB_SM0CAPTCTRLX_EDGX1_1 ;*
10. Start capturing in free-run mode
    - *PWMB_SM0CAPTCTRLA |= PWMB_SM0CAPTCTRLA_ARMA;*
    - *PWMB_SM0CAPTCTRLB |= PWMB_SM0CAPTCTRLB_ARMB;*
    - *PWMB_SM0CAPTCTRLX |= PWMB_SM0CAPTCTRLX_ARMX;*
11. Enable read DMA request on local sync event
    - *PWMB_SM0DMAEN |= PWMB_SM0DMAEN_CAPTDE_1;*
12. PWM start sequence is performed
    - *PWMB_MCTRL |= PWMB_MCTRL_CLDOK;*
    - *PWMB_MCTRL |= PWMB_MCTRL_LDOK;*
    - *PWMB_MCTRL |= PWMB_MCTRL_RUN;*

The PWMB SM0 is configured to capture six edges within a PWM period from PWM input pins A0, B0, X0. The PWM input clock is IPBCLK 100 MHz.

The PWMB_REG_READ structure defined in PMW_B_cfg.h file consists of variables where captured values from PWM capture registers are stored. The variables are defined in the same order as PWM capture registers in a flash memory to allow simple linear DMA transfer. The PWM configuration is performed during peripheral initialization calling PWM_B_Init function with pointer to the structure as a function parameter.

It is not recommended to set both Capture x FIFO DMA Enable bits (CxxDE) and Interrupt Enable INTEN(CxxIE).

DMA Channel 0 Configuration

The configuration of DMA channel 0 is performed in DMA0_Init function in DMA_cfg.c file which is part of Sources folder in the following order:

1. Set Source Address Register to value of the structure address which is passed from input pointer
   a. *DMA_SAR0 = ((uint32_t)(&PWMB_SM0CVAL0)<<1);*
2. Set Destination Address Register to value of PWMA_SM0VAL0 register address
   a. *DMA_DAR0 = ((uint32_t)(ptr)<<1);*
3. Clear state machine for DMA channel 0
   a. *DMA_REQC |= DMA_REQC_CFSM0;*
4. Set request source for the channel—PWMB SM0 read request (10). This defines the logical connection between the DMA requesters and the DMA channel 0.
   a. *DMA_REQC |= DMA_REQC_DMAC0_3 | DMA_REQC_DMAC0_0;*
5. Enable peripheral request for the DMA channel
   a. *DMA_DCR0 |= DMA_DCR0_ERQ;*
6. Set size of source and destination data as WORD (16-bit) and increasing source and destination addresses after each DMA transfer
   a. *DMA_DCR0 |= DMA_DCR0_SINC | DMA_DCR0_DINC | DMA_DCR0_SSIZE_1 | DMA_DCR0_DSIZE_1;*
7. Enable DMA transfer completed interrupt (vector 36) with priority 1
   a. *DMA_DCR0 |= DMA_DCR0_EINT;*
   b. *INTC_IPR3 |= INTC_IPR3_DMACH0_1;*
8. Clear channel status register
   a. *DMA_DSR_BCR0 |= DMA_DSR_BCR0_DONE;*
9. Set number of bytes to be transferred—for 12 16-bit registers it equals 24
   a. *DMA_DSR_BCR0 |= (DMA_DSR_BCR0_BCR & 24);*

DMA controller expects a byte address of source and destination address register. The peripheral registers as well as data memory are addressed in terms on 16-bit words. Because of this the SAR and DAR register values are doubled.

Transfer Process

The DMA request is generated from PWMB SM0 after a local sync signal is set. This is performed every PWM period (100 µs) for this example. The request starts DMA transfer of twelve 16-bit values from capture CVALx and cycle CVALCYC registers to the structure PWMB_REG_READ. As soon as the transfer is completed, the DONE flag is set and the ISR is invoked.

The DMA0 ISR calls DMA0_Preset function that performs following:

1. Reset SAR and DAR registers because they contain address of last 16-bit variable and PWM register
2. Clear status register setting DONE bit
3. Set number of byte to be transferred in next cycle—the number is decreased after each byte is transferred to the zero

The DMA controller can clear a capture flag if the CxxDE bit is set in DMAEN register.

The DMA transfer diagram is shown in Figure 7 :

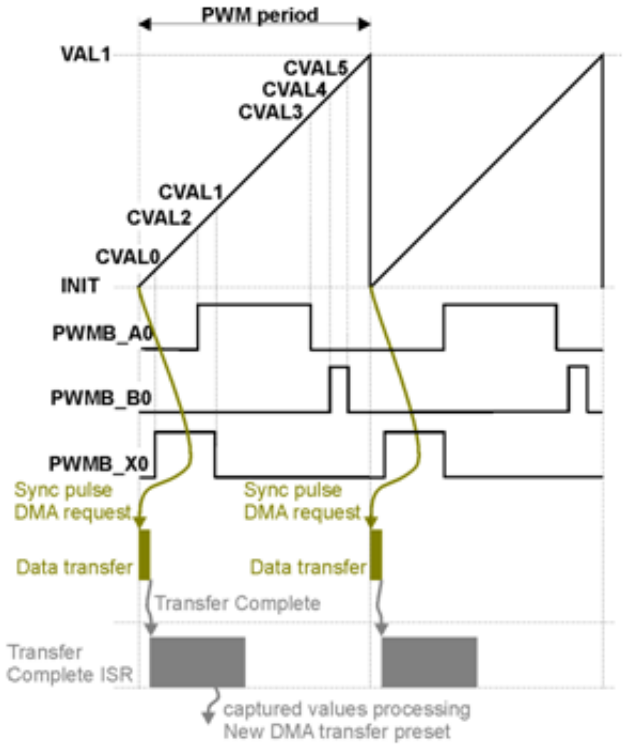

Figure 7. Time diagram of PWM DMA write request processing

3.3 Reading ADC cyclic result registers using DMA controller

The ADC consists of two eight-channel input functions, which are two independent sample and hold circuits feeding two separate 12-bit ADCs. The two separate convertors store their results in an accessible buffer.

When the conversion is completed, the result is placed in the Rn data result register. The DMA transfer can be started either setting End of Scan (EOS) bit or Ready (RDY) bit. For simultaneous conversion as it is in the application example, the EOS bit is used for a DMA request. The DMA trigger source can be selected in the CTRL3 register (DMASRC bit).

To have proper data transfer between the ADC result registers and data memory (result array i16ADCresults []) it is essential to configure dedicated DMA channel.

The application example demonstrates how to read ADC result registers every PWM period using DMA transfers. The ADC is synchronized with the PWMA_SM0 trigger 0 signal using XBAR peripheral (input XBAR_IN20–PWMA0_TRIG0, output XBAR_OUT12 – ADCA_TRIG). The configuration of PWMA module is described in “eFlex PWM A Submodule 0 Configuration” under the section Writing PWM value registers using DMA controller. The basic transfer block diagram is shown in Figure 8.

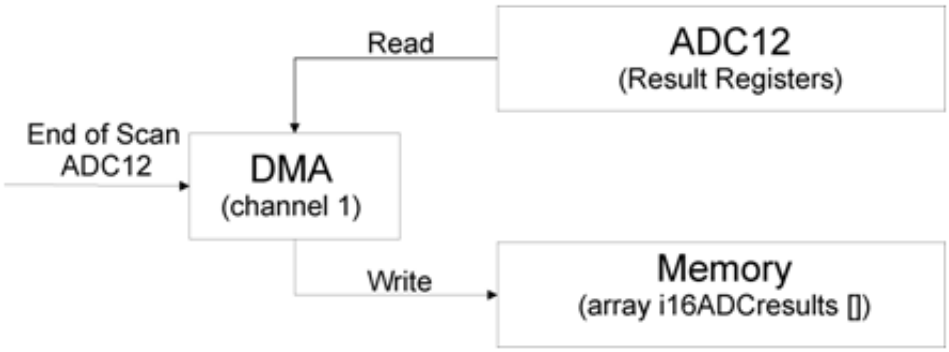

Figure 8. Reading ADC12 result registers using DMA transfer

ADC Cyclic Configuration

The configuration of ADC12 is performed in ADC12_Init function in ADC12_cfg.c file which is part of Sources folder in the following order:

1. Enable clock to GPIO_A where analog channels ANAx signals are connected
   - *SIM_PCE0 |= SIM_PCE0_GPIOA;*
2. Enable ADC peripheral on GPIO_A pins (pin GPIOA0-7)
   - *GPIOA_PER = 0x00FFU;*
3. Enable clock to GPIO_B where analog channels ANBx signals are connected
   - *SIM_PCE0 |= SIM_PCE0_GPIOB;*
4. Enable ADC peripheral on GPIO_B pins (pin GPIOB0-7)
   - *GPIOB_PER = 0x00FFU;*
5. Enable clock to ADC12 module
   - *SIM_PCE2 |= SIM_PCE2_CYCADC;*
6. Enable DMA transfer, enable SYNC 0 for synchronization PWM → ADC,
   - *ADC12_CTRL1 = 0x9005U;*
7. Set Simultaneous mode, clock divisor 4 → clock = 20 MHz
   - *ADC12_CTRL2 |= ADC12_CTRL2_DIV0_2 | ADC12_CTRL2_SIMULT;*
8. Output PWM frequency is set to 10 kHz → 100 µs = 10 ns (cycle) * 5000 * 2
   - *PWMB_SM0INIT = –5000;*
   - *PWMB_SM0VAL1 = 4999;*
9. Set channel lists for all ADC inputs
   - *ADC12_CLIST1 = 0x3210U;*
   - *ADC12_CLIST2 = 0x7654U;*
   - *ADC12_CLIST3 = 0xBA98U;*
   - *ADC12_CLIST4 = 0xFEDCU;*
10. Enable all ADC channels
    - *ADC12_SDIS = 0x0000U;*
11. Set power-up delay to 26 clocks as recommended
    - *ADC12_SDIS = 0x0000U;*
12. Set ADCA & ADCB speed control bits to conversion clock frequency <=20 MHz
    - *ADC12_PWR2 = 0x0405U;*
13. Sync 0 signal connected via XBAR to PWMA_SM0 trigger 0 signal
    - *XBARA_SEL6 = 0x0014U;*

The ADC12 is configured to sample all 16 channels—ANA0–7 & ANB0–7 in simultaneous mode after every PWMA_SM0 period. The synchronization provides Trigger 0 signal which is connected through XBAR to ADC SYNC0 signal.

The i16ADCresults[16] array defined in main.c file is assigned for ADC result register data storing. The ADC configuration is performed during peripheral initialization calling ADC12_Init.

DMA Channel 1 Configuration

The configuration of DMA channel 1 is performed in DMA1_Init function in DMA_cfg.c file which is part of Sources folder in the following order:

1. Set Source Address Register to ADC Result register 0
   - *DMA DMA_SAR1 = ((uint32_t)(&ADC12_RSLT0)<<1);*
2. Set Destination Address Register to value of result array address
   - *DMA_DAR1 = ((uint32_t)(i16ADCresults)<<1);*
3. Clear state machine for DMA channel 1
   - *DMA_REQC |= DMA_REQC_CFSM1;*
4. Set request source for the channel–ADCA end of scan request (12). This defines the logical connection between the DMA requesters and the DMA channel 1
   - *DMA_REQC |= DMA_REQC_DMAC1_3 | DMA_REQC_DMAC1_2;*
5. Enable peripheral request for the DMA channel
   - *DMA_DCR1|= DMA_DCR0_ERQ;*
6. Set size of source and destination data as WORD (16-bit) and increasing source and destination addresses after each DMA transfer
   - *DMA_DCR1 |= DMA_DCR1_SINC | DMA_DCR1_DINC | DMA_DCR1_SSIZE_1 | DMA_DCR1_DSIZE_1;*
7. Enable DMA transfer completed interrupt (vector 35) with priority 1
   - *DMA_DCR1 |= DMA_DCR1_EINT;*
   - *INTC_IPR3 |= INTC_IPR3_DMACH1_1;*
8. Clear channel status register
   - *DMA_DSR_BCR1 |= DMA_DSR_BCR1_DONE;*
9. Set number of bytes to be transferred–for sixteen 16-bit registers it equals 32
   - *DMA_DSR_BCR1 |= (DMA_DSR_BCR1_BCR & 32);*

DMA controller expects a byte address of source and destination address register. The peripheral registers as well as data memory are addressed in terms of 16-bit words. Because of this the SAR and DAR register values are doubled.

Transfer Process

The DMA request is generated from ADC12 once the end of scan bit is set. This is performed after every PWMA_SM0 period which is 62.5 µs for this example. The request starts DMA transfer of sixteen 16-bit values from result registers to the structure of the result array. As soon as the transfer is completed, the DONE flag is set and the interrupt service routine ISR is invoked.

The DMA1 ISR performs:

1. Reset SAR and DAR registers because they contain addresses of last 16-bit variable and PWM register
2. Clear status register setting DONE bit
3. Set number of byte to be transferred in next cycle—the number is decreased after each byte is transferred to the zero

The DMA controller can clear an EOS flag after DONE bit is set.

The DMA transfer diagram is shown in Figure 9 :

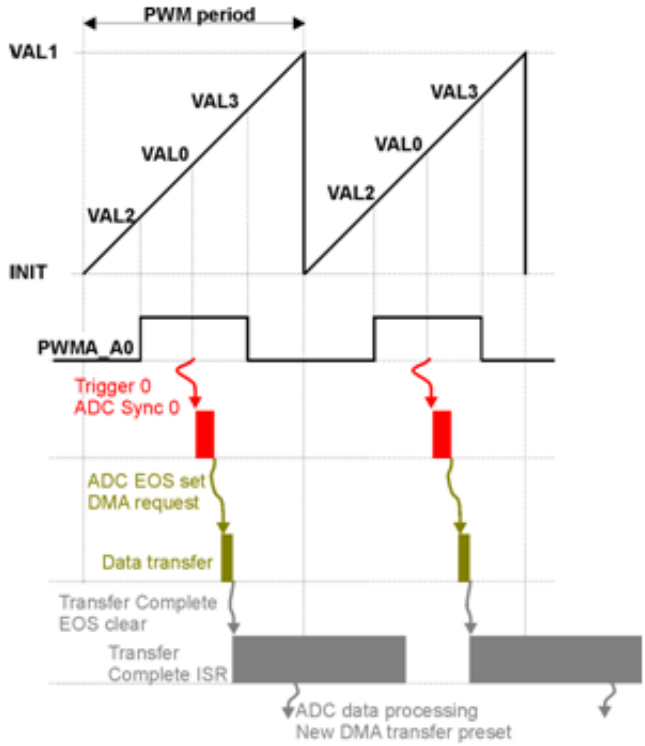

Figure 9. Time diagram of ADC end of scan DMA read request processing

4 Application configuration

The application software supporting the application note is developed using Freescale's CodeWarrior 10.2. The debug interface can be selected among USB-TAP, Universal Multilink, and OpenSource JTAG.

The TWR-56F8400 board is used as a hardware platform. Analog signals as well as input digital signals for edge capturing can be connected through on-board headers.

The application code is supported by a FreeMASTER control page showing application variables separated into three sections PWMA, PWMB, and ADC.

5 Definitions and acronyms

Definitions and acronyms used in this application note are defined below:

Table 1. Definitions and acronyms used

eFlex	enhanced Flexible
DMA	Direct Memory Access
GPIO	General Port Input Output
ADC	Analog-to-Digital Converter
XBAR	Cross-Bar Switch
PWM	Pulse-Width Modulation

Table continues on the next page...

Table 1. Definitions and acronyms used (continued)

ISR	Interrupt Service Routine
DSC	Digital Signal Controller
EOS	End of Scan
FIFO	First In First Out
SMPS	Switch Mode Power Supply

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: AN4598
Rev. 0, 10/2012