Freescale Semiconductor
Application Note

Document Number: AN4828
Rev. 0, 12/2013

MC56F827xx DSCs Crossbar and Signal Multiplexing on 56F827xx for Motor Control Applications

by *Libor Prokop*

1 Introduction

The crossbar switch (XBAR), which was introduced as part of the new generation of Freescale controllers, together with the AND/OR/INVERT (AOI) module are implemented in the MC56F827xx family of Digital Signal Controllers (DSC), dedicated to motor control.

The crossbar module implements an array of M N-input combinational muxes. All muxes share the same N inputs in the same order, but each mux has its own independent select field. The intended application of this module is to provide a flexible crossbar switch function that allows any input to be connected to any output under user control.

The motor control applications are a complex system that utilizes many peripheries, such as the Pulse Width Modulation module (PWM), Analog to Digital Converter (ADC), timers, I/Os, and communication. The crossbar module is a key element of the system's versatility. This application note focuses on the utilization of the XBAR switch for the motor control applications running on MC56F827xx DSC.

Contents

2 Digital signal controllers MC56F827xx

One suitable DSC for a motor control application is MC56F827xx. This is a complex device with many features described in the device reference manual and data sheets. This application note is focused on the connections of the following features and processor modules:

- Core and peripheral clock 50 MHz (Core clock can be set to 100 MHz in the Fast mode.)
- Input signal multiplexing (SIM_GPS registers)
- Two crossbar units with AOI module to interconnect signals between the peripherals
- Core and peripheral clock 50MHz (Core clock can be set to 100MHz in the Fast Mode.)
- Pulse Width Modulator
- 12-bit ADC converter

3 DSC signal paths with the crossbar switches XBARA, XBARB, and AOI module

The DSC 56F827xxx internal signals connection between modules is characterized by two crossbar switches and the AOI module.

The block schematic is shown in Figure 1.

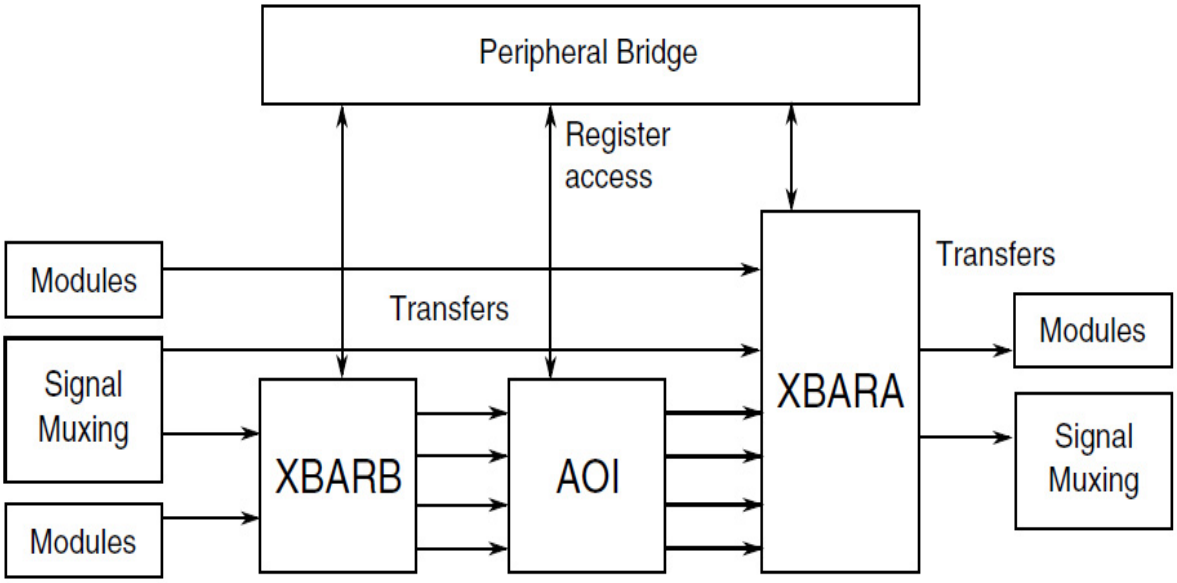

Figure 1. XBARA, XBARB, and AOI integration

4 Signal multiplexing with the crossbar switches, GPIO and SIM GPS

The DSC MC56F827xx signal path flexible philosophy is based on input pin multiplexing with GPIO and SIM GPS modules and crossbar switches XBARA and XBARB with AOI modules.

The GPIO_PER register setting indicates if the dedicated pin will be used as general input/output or in periphery mode. In case the pin is used for periphery this can be connected to one of up to four peripheries (depending on the pin and device). One of the peripheries can be XBAR input or output.

In the XBARA any of the 32 inputs can be connected to any of the 41 XBARA outputs. The inputs can be any signal from pins, hardware module triggers, etc.

In the XBARB any of the 26 inputs can be connected to any of the 16 outputs.

These crossbar connections of a dedicated MC56F827xx DSC are described in sections: XBARA and XBARB Inputs, XBAR Interconnections and XBARA Outputs of the MC56F827xx Reference Manual.

5 Motor control application example with the crossbar XBARA, GPIO and SIM GPS multiplexing

The first motor control application example in this application note utilizes XBARA only.

A typical 3-phase motor control system requires at least one 3*2 Pulse With Modulation signal with a top and bottom signal for each phase. The analog signals usually cover phase current, voltage, temperature and other signals like resolver. The ADC sampling should be synchronized with PWM signals. For debugging purposes the external synchronization signal is usually required.

Internal DSC connection for such a 3-phase motor control system is displayed in Figure 2.

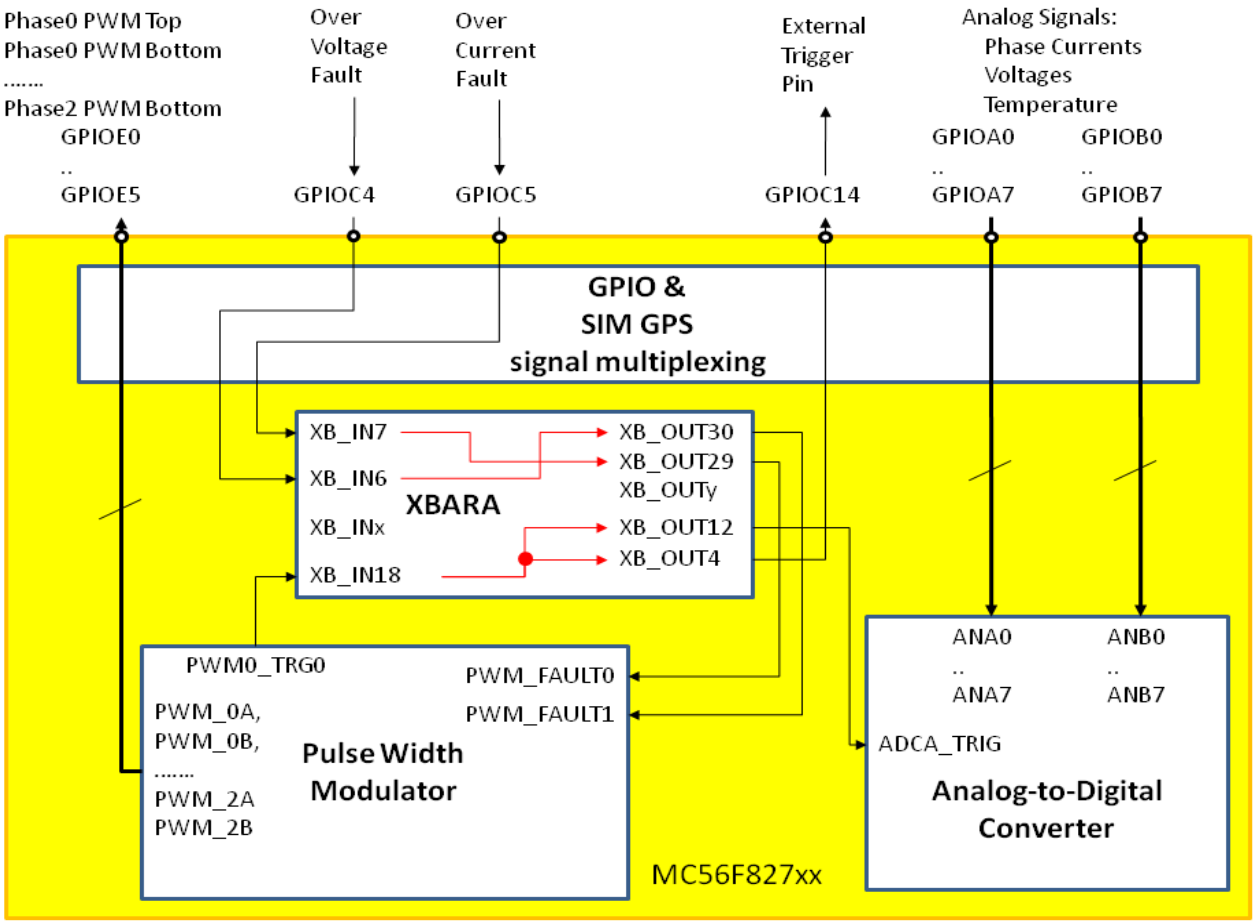

Figure 2. Motor control system internal connectivity example using DSC MC56F827xx

The flexible signal path setting is provided by software. The software initialization of the signal paths between DSC internal modules and input/outputs is provided as follows:

5.1 Initialization of the phase PWM signal paths

The PWM signals are to be connected to pins GPIOE0 to GPIOE5.

The periphery clock must be enabled in the SIM module Peripheral Clock Enable Register before using any periphery. The GPIOE port clock enable bit is set:

- SIM_PCE0[GPIOE] = 1 = GPIOE IP Bus Clock Enable - The peripheral is clocked

The syntax is as follows:

```
SIM_PCE0 |= SIM_PCE0_GPIOE;
```

The GPIOE0 to GPIOE5 must be initialized as peripherals with the following bit groups:

- GPIOE_PER[PE] = 0b0100 000 0011 1111 = Pins 0 to 5 are peripheral (peripheral mode)

This code line is used to initialize GPIO_PER:

```
GPIOE_PER = (GPIOE_PER_PE_5 | GPIOE_PER_PE_4 | GPIOE_PER_PE_3 | GPIOE_PER_PE_2|\
             GPIOE_PER_PE_1 | GPIOE_PER_PE_0);
```

The SIM GPS registers are responsible for selecting the ALT functionality available on most pins. The peripheral pin with pwm signals PWM_0A to PWM_2B has one functionality only. The other PWM_2A to PWM_2B are to be set as ALT0.

This configuration is described in Signal Multiplexing and Pin Assignments in the MC56F827xx Reference Manual. The initialization of the dedicated GPIOE LSBs Peripheral Select Register (SIM_GPSEL):

- SIM_GPSEL[E4] = 0 = GPIO E4 Function = PWMA_2B; Peripheral = PWMA; Direction = IO
- SIM_GPSEL[E5] = 0 = GPIO E5 Function = PWMA_2A; Peripheral = PWMA; Direction = IO

The syntax is as follows:

```
SIM_GPSEL = 0;
```

5.2 The overcurrent and over-voltage signal initialization

In our example, the overcurrent fault and over-voltage fault pins are connected to GPIOC5 and GPIOC4 respectively. The following initialization must be provided:

GPIOC port clock enable bit setting syntax:

```
SIM_PCE0 |= SIM_PCE0_GPIOC;
```

The peripheral mode initialization syntax:

```
GPIOC_PER[PE] = (GPIOC_PER_PE_4 | GPIOC_PER_PE_5);
```

As described in the section, “Signal Multiplexing and Pin Assignments” in the reference manual:

The GPIOC4 utilization as XB_IN6 requires to be set as ALT2, GPIOC5 for XB_IN7 is ALT1, which can be provided with the following syntax:

```
SIM_GPSC|= (SIM_GPSCL_C5_0| SIM_GPSCL_C4_1);
```

The signals are connected to the XBARA inputs XB_IN6, XB_IN7 respectively. Further configuration of the signals is very versatile.

The signal can be connected to any XBARA output. In the example Figure 2, the signals are connected to PWM_FAULT0, PWM_FAULT1, PWM module inputs. The required XBARA output can be found in the table titled “XBARA Outputs” in the device reference manual. The table shows the PWM_FAULT0 and PWM_FAULT1 inputs of the Pulse Width Modulator module are connected to XBARA outputs XB_OUT29 and XB_OUT30.

In the register section of the reference manual chapter “Inter-Peripheral Crossbar Switch A (XBARA)”, the setting for output 29 is provided in the Crossbar A Select Register 14.

The input XB_IN7(=GPIOC5 pin) will be connected to the output XB_OUT29 (=PWM_FAULT0) with the setting:

- XBARA_SEL14 [SEL29] = 7 = 000111 = Connects the XB_IN7 to XB_OUT29

And the syntax is:

```
XBARA_SEL14 |= (XBARA_SEL14_SEL29_2 | XBARA_SEL14_SEL29_1 | XBARA_SEL14_SEL29_0);
```

The setting for the output 30 is provided in the Crossbar A Select Register 15.

So the input XB_IN6(=GPIOC4 pin) will be connected to the output XB_OUT30 (=PWM_FAULT0) with the setting of the Crossbar A Select Register 15:

- XBARA_SEL15 [SEL30] = 6 = 0b000110 = Connects the XB_IN7 to XB_OUT29

With the syntax:

```
XBARA_SEL15 |= (XBARA_SEL15_SEL30_2 | XBARA_SEL15_SEL30_1);
```

5.3 Initialization of the synchronization trigger

The most important feature of the crossbar module is a versatile configuration of any synchronization signal (in case they are connected to an crossbar module). In the example from Figure 2, the PWM0_TRG0 signal from the PWM submodule 0 is used to trigger the Analog to Digital Conversion. However the DSC can be configured to use any other XBARA input as a trigger. And the PWM0_TRG0 signal can be propagated to any XBARA output. In the example, the PWM0_TRG0 signal is connected to two outputs XB_OUT4 and XBOUT12.

In the reference manual, table “XBARA and XBARB Inputs” shows that the PWM0_TRG0 is connected to XBAR input XB_IN18. The table “XBARA Outputs” shows that the XB_OUT12 is connected to ADCA_TRIG of the ADCA analog-to-digital converter. The XBARA Output XB_OUT4 is connected to GPIOC14.

The XBARA initialization for the Analog-to-Digital Conversion module trigger signal ADCA_TRIG is then:

- XBARA_SEL6 [SEL12] = 18 = 0b0010010 Connects the XB_IN18 to XB_OUT12

With the code line:

```
XBARA_SEL6 |= (XBARA_SEL6_SEL12_4 | XBARA_SEL6_SEL12_1);
```

For the External Trigger connected to GPIOC14:

- XBARA_SEL2 [SEL4] = 18 = 0b010010 Connects the XB_IN18 to XB_OUT4

The code line is:

```
XBARA_SEL2 |= (XBARA_SEL2_SEL4_4 | XBARA_SEL2_SEL4_1);
```

Finally the GPIOC14 pin must be configured in the SIM Peripheral Select Register C (SIM_GPSCH) and GPIOC14.

According to Signal Multiplexing and Pin Assignments in the reference manual and GPIOC MSBs Peripheral Select Register (SIM_GPSCH), the XB_OUT4 ALT is ALT1. And so:

- SIM_GPSCH[C14] = 01 = Function = XB_OUT4; Peripheral = XBAR; Direction = OUT

With the syntax:

```
SIM_GPSCH |= SIM_GPSCH_0;
```

The pin GPIOC14 needs to be initialized to peripheral mode:

- GPIOC_PER[PE] |= 0b0100 000 000 000

With the syntax:

```
GPIOC_PER |= GPIOE_PER_PE_14;
```

5.4 The paths of the ADC input signals initialization

The ADC inputs ANA0 to ANA7 are connected to the port GPIOA0 to GPIOA7 multiplex.

The ADC inputs ANB0 to ANB7 are connected to the port GPIOB0 to GPIOB7 multiplex. Before the ADC module initialization the signal connections must be provided.

First, the clock for the PORT A and B must be enabled by setting the SIM_PCE0_GPIOA and SIM_PCE0_GPIOB bits:

```
SIM_PCE0 |= SIM_PCE0_GPIOA;

SIM_PCE0 |= SIM_PCE0_GPIOB;
```

Enabling peripheral mode for GPIOA0 to GPIOA7:

- GPIOA_PER[PE] |= 0b0000 000 1111 1111

With the syntax:

```
GPIOA_PER |= (GPIOE_PER_PE_7 | GPIOE_PER_PE_6 | GPIOE_PER_PE_5 | GPIOE_PER_PE_4|\
                  GPIOE_PER_PE_3 | GPIOE_PER_PE_2 | GPIOE_PER_PE_1 | GPIOE_PER_PE_0);
```

Enabling peripheral mode for GPIOB0 to GPIOB7:

- GPIOB_PER[PE] |= 0b0000 000 1111 1111

With the syntax:

```
GPIOB_PER |= (GPIOB_PER_PE_7 | GPIOB_PER_PE_6 | GPIOB_PER_PE_5 | GPIOB_PER_PE_4|\
                  GPIOB_PER_PE_3 | GPIOB_PER_PE_2 | GPIOB_PER_PE_1 | GPIOB_PER_PE_0);
```

The ALT multiplexing setting is:

- SIM_GPSAL[A] = 0
- SIM_GPSBL[B] = 0

This is a default setting.

```
SIM_GPSAL = SIM_GPSBL = 0;
```

It is not necessary. The analog input signal connection has been established.

This way the DSC signals and synchronization trigger connections are initialized. Finally the individual modules like PWM and ADC will be set up according to the required functionality. This will not be described in detail.

6 DSC multiplexing with XBAR B and AOI module

The second example from Figure 3 utilizes both XBARA, XBARB crossbar switches together with the AOI module.

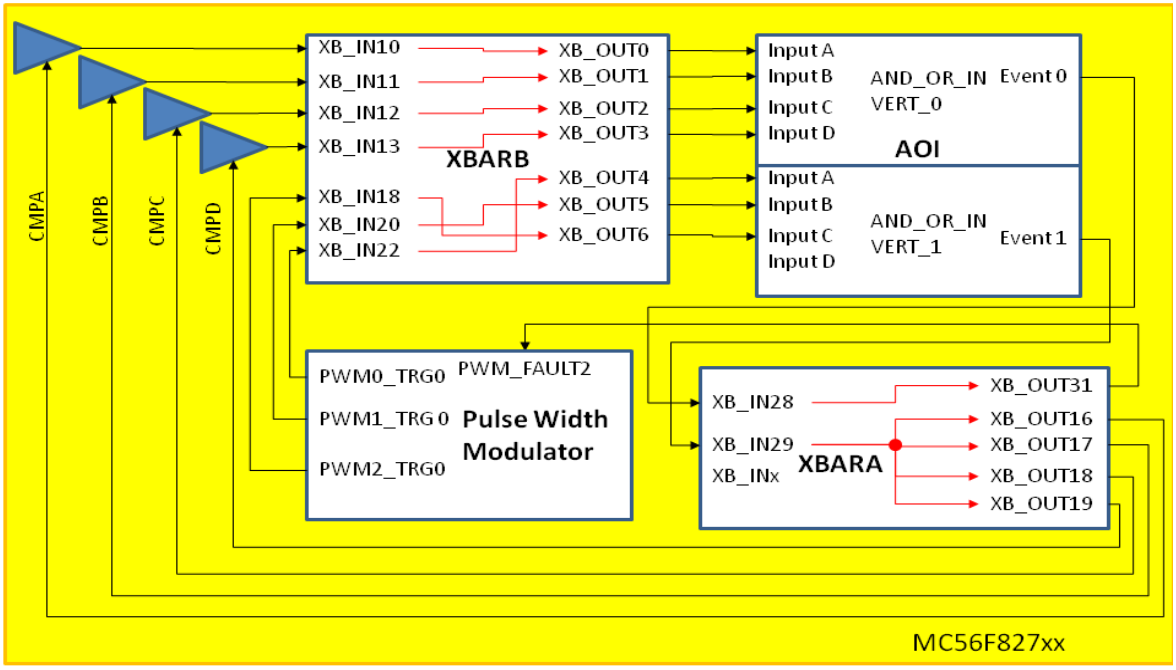

Figure 3. Example 2: Multiplexing with the XBAR B and AOI module

6.1 Initialization of PWM Fault2 signal according to comparator output state

In this example, the PWM Fault2 (PWM_FAULT2) signal is generated according to the comparator output state using the formula:

PWM_FAULT2 = (CMPA_OUT&CMPB_OUT) | (CMPA_OUT&CMPC_OUT) | \
(CMPB_OUT&CMPC_OUT) | CMPD_OUT

The CMPA_OUT comparator output signals A to D are connected to the XBARB inputs (and XBARA as well) XB_IN10 to XB_IN13 respectively. The signal can be connected to any AND/OR/INVERT module inputs. In the example from Figure 3, the signals are connected to AND_OR_INVERT_0 module signals XBAR_OUT0 to XBAR_OUT3.

This requires the following initialization:

- XBARB_SEL0[SEL0] = 10 = 0b001010 Connects the XB_IN10 to XB_OUT0

- XBARB_SEL0[SEL1] = 11 = 0b001011 Connects the XB_IN11 to XB_OUT1
- XBARB_SEL1[SEL2] = 12 = 0b001100 Connects the XB_IN12 to XB_OUT2
- XBARB_SEL1[SEL3] = 13 = 0b001101 Connects the XB_IN13 to XB_OUT3

This code is used to initialize the four signals XBARB connections:

```
XBARB_SEL0 = (XBARB_SEL0_SE1_3 | XBARB_SEL0_SEL1_1 | XBARB_SEL0_SEL1_0) |\
             (XBARB_SEL0_SEL0_3 | XBARB_SEL0_SEL0_1);
XBARB_SEL1 = (XBARB_SEL1_SEL3_3 | XBARB_SEL1_SEL3_2 | XBARB_SEL1_SEL3_0) |\
             (XBARB_SEL1_SE2_3 | XBARB_SEL1_SEL2_2);
```

The AND_OR_INVERT_0 module initialization needs to be:

- AOI_BFCRT010[PT0_AC] = 01 Pass the A input in this product term
- AOI_BFCRT010[PT0_BC] = 01 Pass the B input in this product term
- AOI_BFCRT010[PT0_CC] = 11 Force the C input in this product term to a logical one
- AOI_BFCRT010[PT0_DC] = 11 Force the D input in this product term to a logical one

- AOI_BFCRT010[PT1_AC] = 01 Pass the A input in this product term
- AOI_BFCRT010[PT1_CC] = 01 Pass the C input in this product term
- AOI_BFCRT010[PT1_BC] = 11 Force the B input in this product term to a logical one
- AOI_BFCRT010[PT1_DC] = 11 Force the D input in this product term to a logical one

- AOI_BFCRT230[PT2_BC] = 01 Pass the B input in this product term
- AOI_BFCRT230[PT2_CC] = 01 Pass the C input in this product term
- AOI_BFCRT230[PT2_AC] = 11 Force the A input in this product term to a logical one
- AOI_BFCRT230[PT2_DC] = 11 Force the D input in this product term to a logical one

- AOI_BFCRT230[PT3_AC] = 11 Force the A input in this product term to a logical one
- AOI_BFCRT230[PT3_BC] = 11 Force the B input in this product term to a logical one
- AOI_BFCRT230[PT3_CC] = 11 Force the C input in this product term to a logical one
- AOI_BFCRT230[PT3_DC] = 01 Pass the D input in this product term

The syntax is:

```
AOI_BFCRT010 = (AOI_BFCRT010_PT0_AC_0 | AOI_BFCRT010_PT0_BC_0 |\
                OI_BFCRT010_PT0_CC | AOI_BFCRT010_PT1_DC |\
                AOI_BFCRT010_PT1_AC_0 | AOI_BFCRT010_PT1_CC_0 |\
                AOI_BFCRT010_PT1_BC | AOI_BFCRT010_PT1_DC);
AOI_BFCRT230 = (AOI_BFCRT230_PT2_BC_0 | AOI_BFCRT230_PT2_CC_0 |\
                AOI_BFCRT230_PT2_AC | AOI_BFCRT230_PT2_DC |\
                AOI_BFCRT230_PT3_AC | AOI_BFCRT230_PT3_BC |\
                AOI_BFCRT230_PT3_CC | AOI_BFCRT230_PT3_DC_0);
```

According to the XBARA Outputs table in the reference manual, the AND_OR_INVERT_0 output is the XB_IN28 input of the XBARA. The PWM_FAULT2 is XBAR_OUT31. So the XBARA initialization for the output 31 is provided in the Crossbar A Select Register 15:

- XBARA_SEL15 [SEL31] = 28 = 0b011100 = Connects the XB_IN28 to XB_OUT31

The code syntax is:

```
XBARA_SEL15 = (XBARA_SEL15_SEL29_4 | XBARA_SEL15_SEL29_3 | XBARA_SEL15_SEL29_2);
```

This initialization provides the PWM Fault2 (PWM_FAULT2) generation according to the required CMPA_OUT to CMPD_OUT comparators outputs logical function.

6.2 Initialization of the Comparator A to D Sample according to PWM trigger signals

The second required signal path in the application example from Figure4 generates the Comparator A to D Sample signals as the logical OR function of the PWM trigger signals PWM0_TRG0, PWM1_TRG0 and PWM2_TRG0. All the 4 comparators will be triggered with the same signal. The required logical function is:

CMPA = CMPB = CMPC = CMPD = PWM0_TRG0 | PWM1_TRG0 | PWM2_TRG0

The PWM0_TRG0 (submodule 0) to PWM2_TRG0 (submodule 2) PWM module trigger signals are connected to the XBARB inputs (and XBARA as well) XB_IN18 to XB_IN20 and XB_IN22. The signal can be connected to any AND/OR/INVERT module inputs. In the example from Figure 3, the signals are connected to the AND_OR_INVERT_1 module signals connected to XBAR_OUT4 to XBAR_OUT6.

This requires the following initialization:

- XBARB_SEL2[SEL4] = 18 = 0b010010 Connects the XB_IN18 to XB_OUT4
- XBARB_SEL2[SEL5] = 20 = 0b010100 Connects the XB_IN20 to XB_OUT5
- XBARB_SEL3[SEL6] = 22 = 0b010110 Connects the XB_IN22 to XB_OUT6

The syntax is:

```
XBARB_SEL2  = (XBARB_SEL2_SEL4_4 | XBARB_SEL2_SEL4_1) |\
              (XBARB_SEL2_SEL5_4 | XBARB_SEL2_SEL5_2);
XBARB_SEL3  |= (XBARB_SEL3_SEL6_4 | XBARB_SEL3_SEL6_2 | XBARB_SEL3_SEL6_1 );
```

The AND_OR_INVERT_0 module initialization needs to be:

Following initialization:

- AOI_BFCRT010[PT0_AC] = 01 Pass the A input in this product term
- AOI_BFCRT010[PT0_BC] = 01 Pass the B input in this product term
- AOI_BFCRT010[PT0_CC] = 01 Pass the C input in this product term
- AOI_BFCRT010[PT0_DC] = 11 Force the D input in this product term to a logical one

The syntax is:

```
AOI_BFCRT011 = (AOI_BFCRT011_PT0_AC_0 | AOI_BFCRT011_PT0_BC_0 | AOI_BFCRT011_PT0_CC_0 |
                AOI_BFCRT011_PT0_DC);
```

According to the “XBARA Outputs” table in the reference manual, the AND_OR_INVERT_1 output is the XB_IN29 input of the XBARA.

The CMPx Comparator Window/Sample triggering signals are XBAR_OUT16 to XBAR_OUT19. The XBARA initialization for the outputs 16,17 is provided in the Crossbar A Select Register 8 and outputs 18,19 is provided in the Crossbar A Select Register 9 and:

- XBARA_SEL8 [SEL16] = 29 = 0b011101 Connects the XB_IN29 to XB_OUT16
- XBARA_SEL8 [SEL17] = 29 = 0b011101 Connects the XB_IN29 to XB_OUT17
- XBARA_SEL9 [SEL18] = 29 = 0b011101 Connects the XB_IN29 to XB_OUT18
- XBARA_SEL9 [SEL19] = 29 = 0b011101 Connects the XB_IN29 to XB_OUT19

With the syntax:

```
XBARA_SEL8 = (XBARA_SEL8_SEL16_4 | XBARA_SEL8_SEL16_3 |\
              XBARA_SEL8_SEL16_2 | XBARA_SEL8_SEL16_0) |\
             (XBARA_SEL8_SEL17_4 | XBARA_SEL8_SEL17_3 |\
              XBARA_SEL8_SEL17_2 | XBARA_SEL8_SEL17_0);
XBARA_SEL9 = (XBARA_SEL9_SEL18_4 | XBARA_SEL9_SEL18_3 |\
              XBARA_SEL9_SEL18_2 | XBARA_SEL9_SEL18_0) |\
             (XBARA_SEL9_SEL19_4 | XBARA_SEL9_SEL19_3 |\
              XBARA_SEL9_SEL19_2 | XBARA_SEL9_SEL19_0);
```

This initialization provides the generation of the four identical CMPx Comparator Window/Sample triggering signals. This signal is created as a logical OR function of the PWM trigger signals PWM0_TRG0, PWM1_TRG0 and PWM2_TRG0.

7 Application example 1 code

The following sections elaborate an example software with the connections from Figure 2.

All the code lines used in the context of this application are given below. The code incorporates input settings and crossbar settings. The final code is more complex than the previous samples, as it also provides interrupt vectors, the Pulse Width Modulation Module, and the Analog-to-Digital Converter settings. The signal path initializations, described in Section 5, “Motor control application example with the crossbar XBARA, GPIO and SIM GPS multiplexing are provided by the functions GPIOA_Init(), GPIOB_Init(), GPIOC_Init(), GPIOE_Init() and XBARA_Init().

7.1 Interrupt Vector Table

The file MC56F827xx_vector.asm is located in *Project_Settings\Startup_Code*.

```
JSR  >ADC12_EOS_ISR               ; /* 0x3c Interrupt no. 30 - ivINT_ADC_CC0  */
```

7.2 Included Header Files

The most important headers used in the following code:

```
#include "MC56F82723.h"           /* MC56F82723 Peripheral description header */
#include <intrinsics_56800E.h>    /* intrinsics arithmetic header */
```

7.3 Constants and definitions

```
typedef struct

{

    Word16 adc_result0;

    Word16 adc_result1;

    Word16 adc_result2;

    Word16 adc_result3;

    Word16 adc_result4;

    Word16 adc_result5;

    Word16 adc_result6;

    Word16 adc_result7;

    Word16 adc_result8;

    Word16 adc_result9;

    Word16 adc_result10;

    Word16 adc_result11;

    Word16 adc_result12;

    Word16 adc_result13;

    Word16 adc_result14;

    Word16 adc_result15;

} ADC_RESULT;


typedef volatile unsigned short int vuint16_t;

typedef struct

{

    unsigned short int pwmsminit;
```

```
    vuint16_t pwmsmval0;
    vuint16_t pwmsmval1;
    vuint16_t pwmsmval2;
    vuint16_t pwmsmval3;
    vuint16_t pwmsmval4;
    vuint16_t pwmsmval5;
} PWMA_REG;

typedef struct
{
    PWMA_REG SM0;
    PWMA_REG SM1;
    PWMA_REG SM2;
    PWMA_REG SM3;
} PWMA_REGS;
```

7.4 Variables

```
ADC_RESULT    udtADCresults;
PWMA_REGS     udtPWMAreg;
unsigned      int uwPWM_Update = 0;
unsigned      int uwPWM_ClearFaults = 0;
unsigned      int uwOverCurrentHWFault;
unsigned      int uwOverVoltageHWFault;
```

7.5 Prototypes

```
static void GPIOA_Init(void);
static void GPIOB_Init(void);
static void GPIOC_Init(void);
static void XBARA_Init(void);
static void PWM_A_Init(PWMA_REGS_ALL *ptr);
static void ADC12_Init(void);
void PWM_A_Update(PWMA_REGS_ALL *ptr);
void PWM_Clear_Faults(PWMA_REGS *ptr);

void ADC12_EOS_ISR(void);
```

7.6 Functions

```
static void GPIOA_Init(void)
{
        /* Enable GPIOA */
    SIM_PCE0 |= SIM_PCE0_GPIOA;
    /* ADC Inputs A setting */
        /* Set GPIOA0 to GPIOA7 as peripheral (ANB0 to ANB7) */
    GPIOA_PER |= (GPIOE_PER_PE_7 | GPIOE_PER_PE_6 | GPIOE_PER_PE_5 | GPIOE_PER_PE_4|\
                GPIOE_PER_PE_3 | GPIOE_PER_PE_2 | GPIOE_PER_PE_1 | GPIOE_PER_PE_0);
        /* Select ANA0 to ANA7 */
    SIM_GPSAL = 0;
}


static void GPIOB_Init(void)
{
        /* Enable GPIOA, GPIOB clock */
    SIM_PCE0 |= SIM_PCE0_GPIOB;
    /* ADC Inputs B setting */
        /* set GPIOB0 to GPIOB7 as peripheral (ANB0 to ANB7) */
    GPIOB_PER |= (GPIOB_PER_PE_7 | GPIOB_PER_PE_6 | GPIOB_PER_PE_5 | GPIOB_PER_PE_4|\
                GPIOB_PER_PE_3 | GPIOB_PER_PE_2 | GPIOB_PER_PE_1 | GPIOB_PER_PE_0);
        /* Select ANB0 to ANB7 */
    SIM_GPSBL = 0;
}

static void GPIOC_Init(void)
{
        /* Enable GPIOC clock */
    SIM_PCE0 |= SIM_PCE0_GPIOC;
    /* Over-current and over-voltage fault inputs multiplex setting */
        /* Set GPIOC4 as XB_IN6, GPIOC5 as XB_IN7 XBAR inputs */
    SIM_GPSC|= (SIM_GPSCL_C5_0| SIM_GPSCL_C4_1);
        /* GPIOC4(over-voltage) and Set GPIOC5(over-current) pins are for peripheral */
    GPIOC_PER[PE] = (GPIOC_PER_PE_4 | GPIOC_PER_PE_5);
```

```
    /* XB_OUT4 to GPIOC14 to External Trigger Pin */
        /* Set GPIOC14 as XB_OUT4 XBAR outputs */
    SIM_GPSCH |= SIM_GPSCH_0;
        /* GPIOC14 pin is for (XB_OUT4 XBAR) peripheral */
    PIOC_PER |= GPIOE_PER_PE_14;
}

static void GPIOE_Init (void)
{
        /* Enable GPIOE clock */
    SIM_PCE0 |= SIM_PCE0_GPIOE;
        /* PWM_0A to PWM_2B set as peripheral */
    GPIOE_PER = (GPIOE_PER_PE_5 | GPIOE_PER_PE_4| GPIOE_PER_PE_3 | GPIOE_PER_PE_2|\
                GPIOE_PER_PE_1 | GPIOE_PER_PE_0);
        /* PWM_0A to PWM_2B select */
    SIM_GPSEL = 0;
}

static void XBARA_Init(void)
{
    /* Over-current and Over-voltage signals */
        /* XB_IN7 to XB_OUT29 */
    XBARA_SEL14 |= (XBARA_SEL14_SEL29_2 | XBARA_SEL14_SEL29_1 | XBARA_SEL14_SEL29_0);
        /* XB_IN6 to XB_OUT30 */
    XBARA_SEL15 |= (XBARA_SEL15_SEL30_2 | XBARA_SEL15_SEL30_1);

    /* PWM to ADC and GPIOC14 synchronization trigger signals */
    /* ADC Sync pulse generated through XBARA_12 */
        /* XB_IN18 to XB_OUT12 */
    XBARA_SEL6 |= (XBARA_SEL6_SEL12_4 | XBARA_SEL6_SEL12_1);
        /* XB_IN18 to XB_OUT4 */
    XBARA_SEL2 |= (XBARA_SEL2_SEL4_4 | XBARA_SEL2_SEL4_1);
}

static void PWM_A_Init(PWMA_REGS *ptr)
```

```
{
        /* enable PWMA clock to SM0, SM1, SM2 */
    SIM_PCE3 |= (SIM_PCE3_PWMACH0 | SIM_PCE3_PWMACH1 | SIM_PCE3_PWMACH2);

/*****************************************************************************/
    /* SM0 Module */
    PWMA_SM0CTRL = /* PWMA_SM0CTRL_FULL |*/ PWMA_SM0CTRL_HALF;  /* half reload cycle */

    /* Complementary PWM
     * Initialization Local Sync
     * Local force signal
     * Force enabled - force initializes the counter
     * Clock IP Bus
     * Local Reload */
    PWMA_SM0CTRL2 = PWMA_SM0CTRL2_FRCEN;

    /* set 25kHz PWM period --> 40.0us = 10ns * 2000 * 2   */
    PWMA_SM0INIT = -2000;
    PWMA_SM0VAL0 = 0;
    PWMA_SM0VAL1 = 1999;

    PWMA_SM0VAL2 = -((PWMA_SM0VAL1+1)>>1); /* 50% duty cycle */
    PWMA_SM0VAL3 = (PWMA_SM0VAL1+1)>>1;    /* 50% duty cycle */

    /* dead time = 1us */
    PWMA_SM0DTCNT0 = 100;
    PWMA_SM0DTCNT1 = 100;

    /* enable PWM 0 A, B mask at Fault 1 and Fault 2 inputs */
    PWMA_SM0DISMAP0 = (PWMA_SM0DISMAP0_DIS0B | PWMA_SM0DISMAP0_DIS0B)|\
                      (PWMA_SM0DISMAP0_DIS0A | PWMA_SM0DISMAP0_DIS0A);
    PWMA_SM0DISMAP1 = 0;

/*****************************************************************************/
    /* SM1 Module */
```

```
    PWMA_SM1CTRL = PWMA_SM1CTRL_HALF;  /* half reload cycle */
        /* Fractinal PWM enable for 2,3 registers */
    PWMA_SM1FRCTRL = PWMA_SM1FRCTRL_FRAC_PU | PWMA_SM1FRCTRL_FRAC23_EN;
    ptr->SM1.pwmfrctrl = PWMA_SM1FRCTRL;/* prepare for modifications */

    /* Complementary PWM
     * Initialization Sync from SM0
     * Master force signal from submodule 0 causes initialisation
     * Force enabled - force initializes the counter
     * Clock from SM0
     * Reload from SM0 */
    PWMA_SM1CTRL2 = PWMA_SM1CTRL2_INIT_SEL_1 | PWMA_SM1CTRL2_FRCEN | \
                    PWMA_SM1CTRL2_FORCE_SEL_0| PWMA_SM1CTRL2_RELOAD_SEL | \
                    PWMA_SM1CTRL2_CLK_SEL_1;

    /* set 25kHz PWM period --> 40.0us = 10ns * 2000 * 2   */
    PWMA_SM1INIT = -2000;
    PWMA_SM1VAL0 = 0;

    PWMA_SM1VAL2 = -((PWMA_SM0VAL1+1)>>1); /* 50% duty cycle */
    PWMA_SM1VAL3 = (PWMA_SM0VAL1+1)>>1;    /* 50% duty cycle */

    /* dead time = 1us */
    PWMA_SM1DTCNT0 = 100;
    PWMA_SM1DTCNT1 = 100;

    /* enable PWM 1 A, B mask at Fault 1 and Fault 2 inputs */
    PWMA_SM1DISMAP0 = (PWMA_SM1DISMAP0_DIS0B | PWMA_SM1DISMAP0_DIS0B)|\
                      (PWMA_SM1DISMAP0_DIS0A | PWMA_SM1DISMAP0_DIS0A);
    PWMA_SM1DISMAP1 = 0;

/*****************************************************************************/
    /* SM2 Module */
    PWMA_SM2CTRL = /* PWMA_SM2CTRL_FULL |*/ PWMA_SM2CTRL_HALF;  /* half reload cycle */
    /* Fractinal PWM enable for 2,3 registers */
```

Application example 1 code

```
PWMA_SM2FRCTRL = PWMA_SM2FRCTRL_FRAC_PU | PWMA_SM2FRCTRL_FRAC23_EN;
ptr->SM2.pwmfrctrl = PWMA_SM2FRCTRL;/* prepare for modifications */

/* Complementary PWM
 * Initialization Sync from SM0
 * Force enabled - force initializees the counter
 * Master force signal from submodule 0
 * Reload from SM0
 * Clock from SM0 */
PWMA_SM2CTRL2 = PWMA_SM2CTRL2_INIT_SEL_1 | PWMA_SM2CTRL2_FRCEN | \
                PWMA_SM2CTRL2_FORCE_SEL_0 | PWMA_SM2CTRL2_RELOAD_SEL |\
PWMA_SM2CTRL2_CLK_SEL_1 ;

/* set 25kHz PWM period --> 40.0us = 10ns * 2000 * 2   */
PWMA_SM2INIT = -2000;
PWMA_SM2VAL0 = 0;

PWMA_SM2VAL2 = -((PWMA_SM0VAL1+1)>>1); /* 50% duty cycle */
PWMA_SM2VAL3 = (PWMA_SM0VAL1+1)>>1;    /* 50% duty cycle */

/* dead time = 1us */
PWMA_SM2DTCNT0 = 100;
PWMA_SM2DTCNT1 = 100;


/* enable PWM 1 A, B mask at Fault 1 and Fault 2 inputs */
PWMA_SM2DISMAP0 = (PWMA_SM2DISMAP0_DIS0B | PWMA_SM2DISMAP0_DIS0B)|\
                  (PWMA_SM2DISMAP0_DIS0A | PWMA_SM2DISMAP0_DIS0A);
PWMA_SM2DISMAP1 = 0;

/* Enable output on PWMA_A0, PWMA_A1, PWMA_A2, PWMA_A3 */
PWMA_OUTEN = (PWMA_OUTEN_PWMA_EN_0 | PWMA_OUTEN_PWMA_EN_1 |\
PWMA_OUTEN_PWMA_EN_2 | PWMA_OUTEN_PWMA_EN_3);
/* Enable output on PWMA_B0, PWMA_B1, PWMA_B2, PWMA_B3 */
PWMA_OUTEN |= (PWMA_OUTEN_PWMB_EN_0 | PWMA_OUTEN_PWMB_EN_1 |\
```

```
PWMA_OUTEN_PWMB_EN_2 | PWMA_OUTEN_PWMB_EN_3);

PWMA_MCTRL |= PWMA_MCTRL_CLDOK; /* Clear LDOK bits */
PWMA_MCTRL |= PWMA_MCTRL_LDOK;  /* LDOK */
PWMA_MCTRL |= PWMA_MCTRL_RUN;   /* Enable clock */
ptr->pwmmctrl = PWMA_MCTRL;

/* Module SM0 */
ptr->SM0.pwmsminit = PWMA_SM0INIT;
ptr->SM0.pwmsmval0 = PWMA_SM0VAL0;
ptr->SM0.pwmsmval1 = PWMA_SM0VAL1;
ptr->SM0.pwmsmval2 = PWMA_SM0VAL2;
ptr->SM0.pwmsmval3 = PWMA_SM0VAL3;
ptr->SM0.pwmsmval4 = PWMA_SM0VAL4;
ptr->SM0.pwmsmval5 = PWMA_SM0VAL5;

/* Module SM1 */
ptr->SM1.pwmsminit = PWMA_SM1INIT;
ptr->SM1.pwmsmval0 = PWMA_SM1VAL0;
ptr->SM1.pwmsmval1 = PWMA_SM1VAL1;
ptr->SM1.pwmsmval2 = PWMA_SM1VAL2;
ptr->SM1.pwmsmval3 = PWMA_SM1VAL3;
ptr->SM1.pwmsmval4 = PWMA_SM1VAL4;
ptr->SM1.pwmsmval5 = PWMA_SM1VAL5;

/* Module SM2 */
ptr->SM2.pwmsminit = PWMA_SM2INIT;
ptr->SM2.pwmsmval0 = PWMA_SM2VAL0;
ptr->SM2.pwmsmval1 = PWMA_SM2VAL1;
ptr->SM2.pwmsmval2 = PWMA_SM2VAL2;
ptr->SM2.pwmsmval3 = PWMA_SM2VAL3;
ptr->SM2.pwmsmval4 = PWMA_SM2VAL4;
ptr->SM2.pwmsmval5 = PWMA_SM2VAL5;

/* trigger signal 0 used to synchronize ADC via XBAR */
```

Application example 1 code

```
    PWMA_SM0TCTRL |= (PWMA_SM0TCTRL_OUT_TRIG_EN_0);
}
static void ADC12_Init(void)
{
       /* enable clock to ADC modules */
    SIM_PCE2 |= SIM_PCE2_CYCADC;

    /* ADC registers */
        /* SMODE - triggered parallel, SYNC0 - enabled, End of scan ISR enabled */
    ADC_CTRL1 = 0x1805U;
        /* Simultaneous parallel mode; DIV0 = 0_0100 10MHz at PLL 50MHZ */
    ADC_CTRL2  |= ADC_CTRL2_DIV0_2 | ADC_CTRL2_SIMULT;
        /* SAMPLE3 - ANA3, SAMPLE2 - ANA2,SAMPLE1 - ANA1, SAMPLE0 - ANA0 */
    ADC_CLIST1 = 0x3210U;
        /* SAMPLE7 - ANA7, SAMPLE6 - ANA6,SAMPLE5 - ANA5, SAMPLE4 - ANA4 */
    ADC_CLIST2 = 0x7654U;
        /* SAMPLE11 - ANB3, SAMPLE10 - ANB2,SAMPLE9 - ANB1, SAMPLE8 - ANB0 */
    ADC_CLIST3 = 0xBA98U;
        /* SAMPLE14 - ANB7, SAMPLE14 - ANB6,SAMPLE13 - ANB5, SAMPLE12 - ANB4 */
    ADC_CLIST4 = 0xFEDCU;
        /* enable ADC channels 0to7&8to15 -> ANA0toANA7, ANB0toANB7 */
    ADC_SDIS   = 0x0000U;
        /*  power-up delay set to 26 clocks*/
    ADC_PWR  = 0x01A0U;
        /* DIV1 = 100 */
    ADC_PWR2 = ADC_PWR2_DIV1_2;

        /* Enable End of Scan interrupt - priority 1 */
    INTC_IPR2 |= INTC_IPR2_ADC_CC0_1;
} /* Module SM0 */
    PWMA_SM0INIT = ptr->SM0.pwmsminit;
    PWMA_SM0VAL0 = ptr->SM0.pwmsmval0;
    PWMA_SM0VAL1 = ptr->SM0.pwmsmval1;
    PWMA_SM0VAL2 = ptr->SM0.pwmsmval2;
    PWMA_SM0VAL3 = ptr->SM0.pwmsmval3;
```

```
    PWMA_SM0VAL4 = ptr->SM0.pwmsmval4;
    PWMA_SM0VAL5 = ptr->SM0.pwmsmval5;


    /* Module SM1 */
    PWMA_SM1INIT = ptr->SM1.pwmsminit;
    PWMA_SM1VAL0 = ptr->SM1.pwmsmval0;
    PWMA_SM1VAL1 = ptr->SM1.pwmsmval1;
    PWMA_SM1VAL2 = ptr->SM1.pwmsmval2;
    PWMA_SM1VAL3 = ptr->SM1.pwmsmval3;
    PWMA_SM1VAL4 = ptr->SM1.pwmsmval4;
    PWMA_SM1VAL5 = ptr->SM1.pwmsmval5;

    /* Module SM2 */
    PWMA_SM2INIT = ptr->SM2.pwmsminit;
    PWMA_SM2VAL0 = ptr->SM2.pwmsmval0;
    PWMA_SM2VAL1 = ptr->SM2.pwmsmval1;
    PWMA_SM2VAL2 = ptr->SM2.pwmsmval2;
    PWMA_SM2VAL3 = ptr->SM2.pwmsmval3;
    PWMA_SM2VAL4 = ptr->SM2.pwmsmval4;
    PWMA_SM2VAL5 = ptr->SM2.pwmsmval5;

    /* Set LDOK LDOK0 for SM0,1,2 update,  LDOK3 for SM3 update*/
    PWMA_MCTRL |= PWMA_MCTRL_LDOK_0;
}

void PWM_Clear_Faults(PWMA_REGS *ptr)
{
    PWMA_FSTS0 |= PWMA_FSTS0_FFLAG;
    PWMA_FSTS1 |= PWMA_FSTS1_FFLAG;
}
```

7.7 Main function and initializations

In our example, the position is periodically read in the software main loop, but the position can be read from any interrupt subroutine (for example, from TimeBaseISR):

```
thetaKElectrical = ENC_PositionGet(&encElPosParam);
thetaKMechanical = ENC_PositionGet(&encMechPosParam);
```

The initialization and main software loop is below:

```
void main (void)
{
    GPIOA_Init()
    GPIOB_Init();
    GPIOC_Init();
    GPIOE_Init()
    XBARA_Init();
    ADC12_Init();
    PWM_A_Init(&udtPWMAreg);

    while(1)
    {
        if (uwPWM_Update)
        {
            PWM_A_Update(&udtPWMAreg);
            /* update PWM duty cycles according to udtPWMAreg */
            PWM_A_Update(&udtPWMAreg);
            uwPWM_Update = 0;
        }

        if (uwPWM_ClearFaults)
        {
            PWM_Clear_Faults(&udtPWMAreg);
            uwPWM_ClearFaults = 0;
        }
        /* check over-current fault flag */
    ((PWMA_FSTS0&PWMA_FSTS0_FFLAG_0)!=0)? (uwOverCurrentHWFault = 1) : \
                                          (uwOverCurrentHWFault = 0);
        /* check over-voltage fault flag */
    ((PWMA_FSTS0&PWMA_FSTS0_FFLAG_1)!=0)? (uwOverVoltageHWFault = 1) : \
                                          (uwOverVoltageHWFault = 0);
```

```
    }
}
```

7.8 Time base interrupt subroutine

```
#pragma interrupt alignsp
void ADC12_EOS_ISR(void)
{
    /* read ADC samples from channels 0&8 */
    udtADCresults.adc_result0 = ADC_RSLT0;
    udtADCresults.adc_result1 = ADC_RSLT1;
    udtADCresults.adc_result2 = ADC_RSLT2;
    udtADCresults.adc_result3 = ADC_RSLT3;
    udtADCresults.adc_result4 = ADC_RSLT4;
    udtADCresults.adc_result5 = ADC_RSLT5;
    udtADCresults.adc_result6 = ADC_RSLT6;
    udtADCresults.adc_result7 = ADC_RSLT7;
    udtADCresults.adc_result8 = ADC_RSLT8;
    udtADCresults.adc_result9 = ADC_RSLT9;
    udtADCresults.adc_result10 = ADC_RSLT10;
    udtADCresults.adc_result11 = ADC_RSLT11;
    udtADCresults.adc_result12 = ADC_RSLT12;
    udtADCresults.adc_result13 = ADC_RSLT13;
    udtADCresults.adc_result14 = ADC_RSLT14;
    udtADCresults.adc_result15 = ADC_RSLT15;

    /* Clear interrupt request flag */
    ADC_STAT |= ADC_STAT_EOSI0;
}
```

8 Acronym definitions

Table 1. Acronym definitions

ADC	Analogue-to-Digital Converter
AOI	Crossbar And/Or/Invert Module
CW	CodeWarrior

DSC	Digital Signal Controller
FOC	Field Oriented Control
GPIO	General Port Input Output
ISR	Interrupt Service Routine
PWM	Pulse-Width Modulation
SIM	System Integration Module
Motor control	In this application note, this means a process that controls an electrical motor such as a BLDC PMSM, AC-induction, etc.
XBAR	Cross-Bar Switch
XBARA	Cross-Bar Switch A
XBARB	Cross-Bar Switch B

9 References

MC56F827xx Reference Manual (MC56F827XXRM)

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4828
Rev. 0
12/2013