Freescale Semiconductor
Application Note

Document Number: AN4410
Rev. 0, 11/2011

FlexTimer and ADC Synchronization for Field Oriented Control on Kinetis

by: Pavel Rech
Systems Application Engineer, Microcontroller Solutions Group

Contents

1 Introduction

This application note describes the synchronization between the FlexTimer (FTM) and analog-to-digital converter (ADC) for the Kinetis K40 microcontroller (MCU). This feature is often used for precision measurement in motor control applications.

Field-oriented control is one of the most widespread methods of motor control for both synchronous and induction machines. For all kinds of feedback control, precise and quick measurement is important because it determines the accuracy of the controlled quantities and, in this way, the whole application control. Even though the ADC can measure precisely and quickly, the wrong data can still be obtained. This is because the accuracy depends on the instance of sensing and whether the value which is to be measured is average, effective, or instantaneous.

This application note consists of an introduction, a short peripherals description, and an example which sets out the module settings and how it works. The diagrams and measured waveforms are included for clarity and to make it easier to understand the synchronization process.

2 Synchronization principle

It is a well-known fact that there would be no control without measurement. For a precise and fast control, it is necessary to have the actual data before feedback system intervention. Particularly, a periodic signal must be measured at the right time instance, otherwise, an incorrect value may be obtained. Evidently, there is a requirement for quality sensing. The FTM and ADC synchronization, together with quality ADCs and sensors, is an example of a system that provides the precise electrical quantity measurements.

There are other benefits gained from this suggested solution:

- The measurement is not influenced by disturbances and interferences from the switching of the power semiconductors. In other words, the synchronization helps to filter the measured currents, a process known as anti-aliasing.
- Secondly, an average value can be obtained. If the measurement is made in the middle of the pulse length of a pulse width modulation (PWM), an average value of the current is obtained without any additional calculation. See Figure 1. This principle works only on the condition that the motor electrical time constant must be many times higher than the switching period. Therefore, the motor current waveform will be almost linear during the pulse of the PWM.
- The final benefit is that all processing, until the ADC interrupt routine starts, can be done by hardware only, without computing power.

In specific cases, the FTM and ADC synchronization can be the only way to measure something. The discontinuous current, which forms only a part of the PWM period, can be taken as an example.

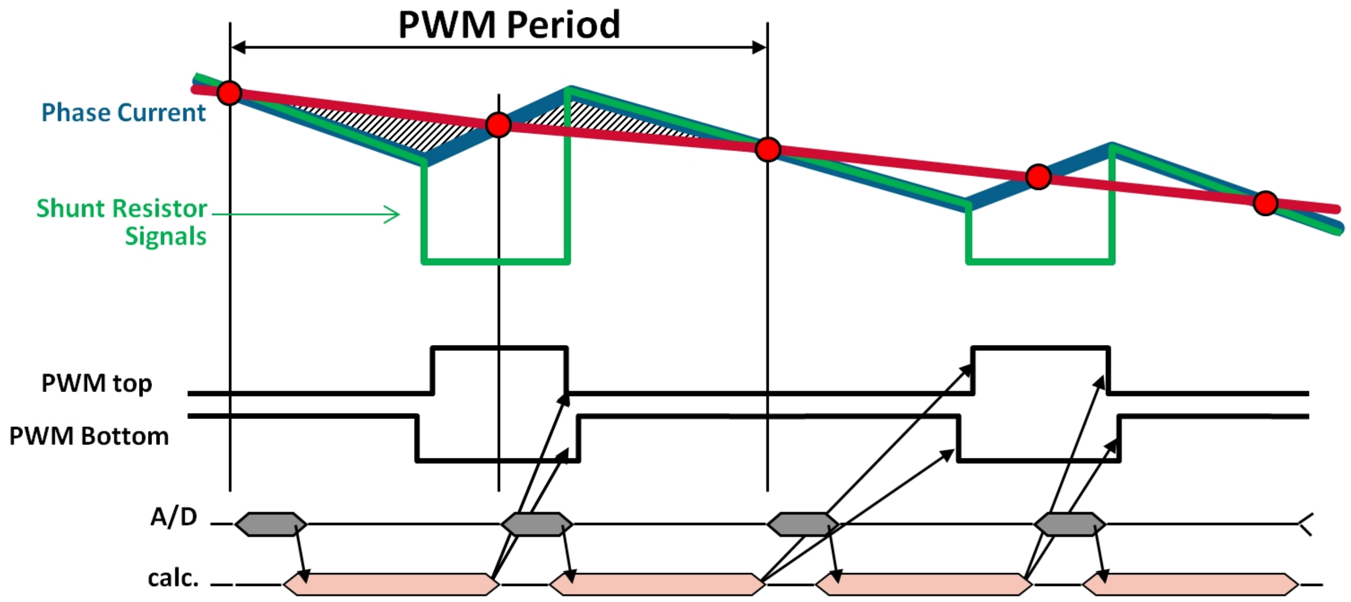

Figure 1. Synchronization principle

It depends on the determination of the application, or, on the type of current sensing, as to which instance, or how many instances during a PWM period, are convenient to measure. Accordingly, the right approach must be chosen. Typically, one or two pairs of current samples are needed. Basically, there are two main possibilities of implementing the FTM and ADC synchronization:

- Without the programmable delay block (PDB): If the PDB is not used, the ADC can be triggered whenever any channel match event occurs, or when the FTM counter is equal to the Counter Initial Value register.
- With the PDB: If the PDB is used, the sampling can be delayed by a value defined by the PDB. It is allowed to trigger at an arbitrary point or two points, because the ADC includes two result registers. See Figure 2.

3 Peripherals

The approach of using the FTM and ADC synchronization with the PDB is preferred, especially in the field-oriented control applications, because it has many advantages. Figure 2 simply shows the peripheral interconnection of these three blocks.

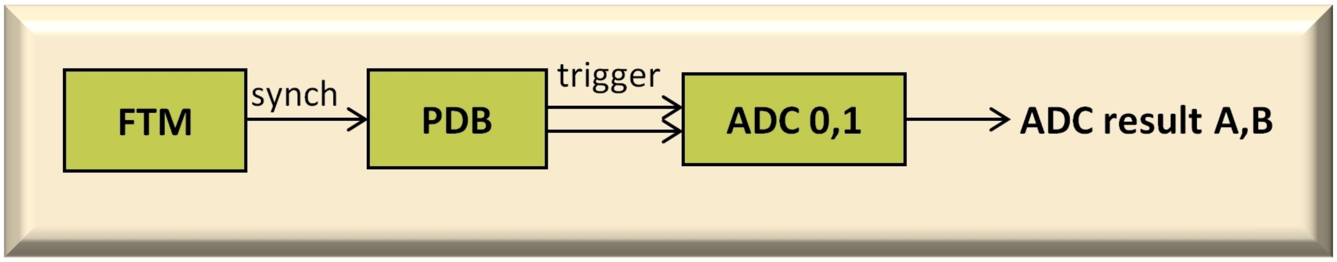

Figure 2. Peripheral interconnection

3.1 FTM

The FTM is a timer designed especially for the motor control and power management applications. The Kinetis K40 MCU has three FlexTimer modules. The first module is an 8-channel timer designed mainly for the generation of PWM signals and all operations connected with it such as deadtime insertion, fault control, enhanced triggering functionality, and initialization and polarity control. The second and third FTMs have only 2-channel timers and are determined for quadrature decoding, or the other purposes.

The basic strategies of the FTM and ADC synchronization depend on the way the PWM signals are generated. It is important to emphasize that for having the possibility of synchronization, the combine mode PWM must be used. The edge-aligned, or center-aligned PWM modes are not supported. In combine mode, the edge-aligned, center-aligned, or asymmetrical PWM can also be generated, therefore, there are no restrictions imposed by the requirement of FTM ADC synchronization. The FTM can generate triggers derived from various events which occur when the FTM counter equals any Value Register or Initial Register. It is possible to set up only one or all events together.

3.2 PDB

The PDB is a timer that provides the controllable timing delays for various peripherals. PDB has two ADC channel outputs and each channel has two pre-trigger outputs. The pre-triggers are assigned to trigger the corresponding ADC channel when PDB counter reach the PDB pre/trigger value. Taking into account that each ADC has two result registers, two trigger instants are allowed for each ADC, during the switching period.

3.3 ADC

ADC can convert up to four pairs of differential or 24 single-ended analog inputs for differential 16-, 13-, 11-, or 9-bit modes and for single-ended 16-, 12-, 10-, or 8-bit modes. The converter is based on linear successive approximation algorithm. For the synchronization it is recommended that:

- Hardware trigger must be chosen
- The Single mode must be set up because in the Continuous Conversion (COCO) mode, only the first sample would be triggered.
- To finish the ADC interrupt routine, the Conversion Complete (COCO) flag must be cleared, or, all the ADC result registers must be read.

.

4 Example

The following subsections briefly describe the ADC-FTM synchronization using PDB, with the help of an example.

4.1 Task

To demonstrate the FTM and ADC synchronization, the following task was chosen:

Propose the algorithm as a part of the field-oriented control of a permanent magnet synchronous motor (PMSM), which synchronizes the ADC with the FTM through the PDB in such a way that two phase currents will be measured simultaneously in the middle of a PWM pulse when the bottom semiconductors from the full bridge are turned on. What code lines should be added if two phase currents are to be measured twice per PWM period?

4.2 Timing

Figure 3 and Figure 4 show the synchronization timing principle for one and two ADC instances, respectively. The center-aligned PWM was used because it is requested for PMSM control applications. The trigger from the FlexTimer is generated when the FlexTimer counter is equal to the Counter Initial Value register. The PDB does not generate the delay because the trigger is generated in the middle of the pulse of the center PWM. Both the ADCs are triggered from the PDB pre-triggers. After finishing the ADC conversion, the program jumps to the ADC interrupt routine. For measurement twice per PWM period, the delay generated by the PDB must be used for the second pair of samples. The setting for this case is marked in the code notes given in the sections PDB initialization , ADC initialization, and Other settings and the ADC interrupt routine.

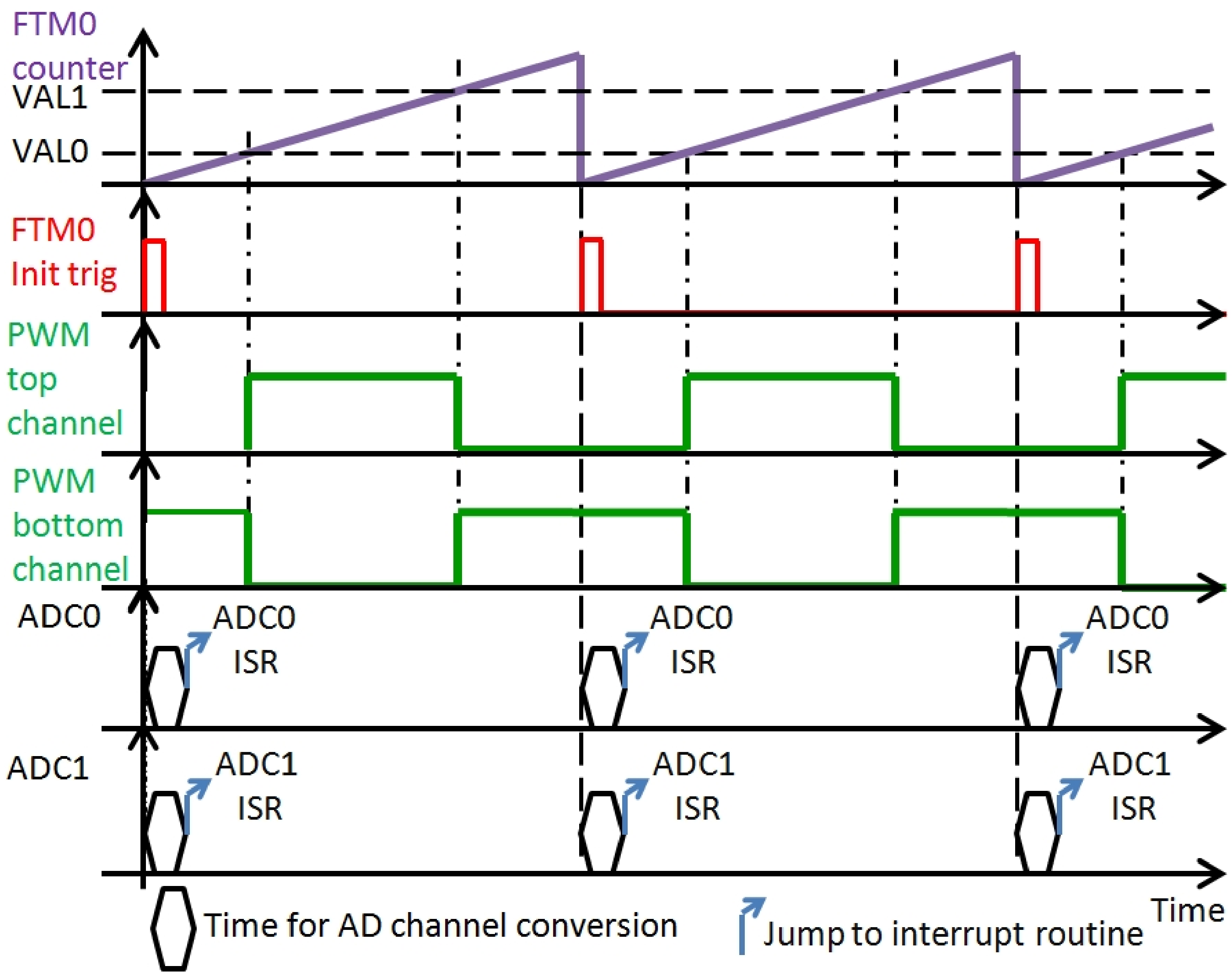

Figure 3. Timing for one ADC instance

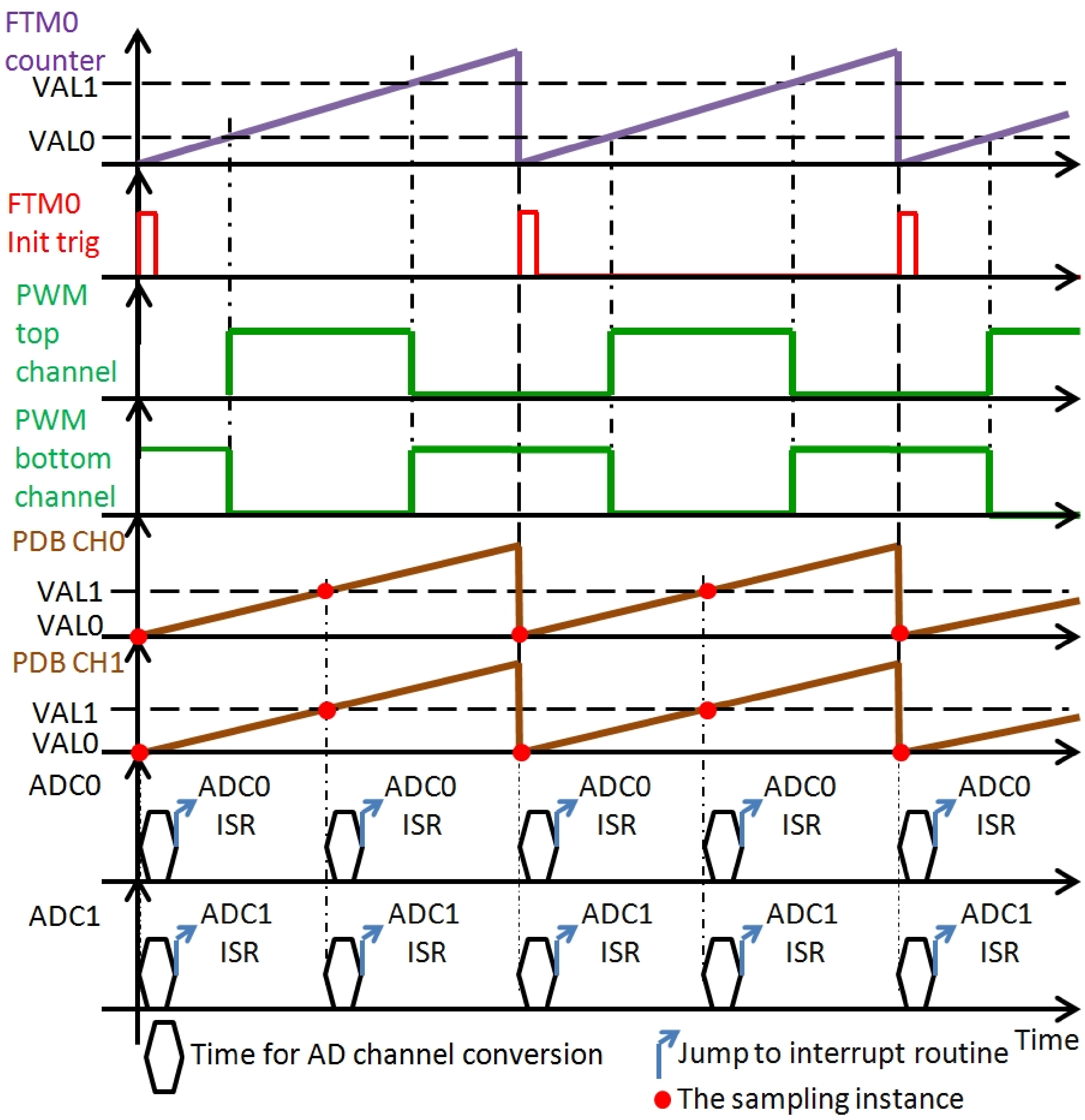

Figure 4. Timing for two ADC instances

4.3 Periperal initialization

The code example including initialization of the FTM, PDB, and ADC is given in the following three subsections. The direct synchronization of the ADCs from the FTM0 is noted in the code given below.

4.3.1 FTM initialization

```
FTM0_MODE |= FTM_MODE_WPDIS_MASK; /* Disable write protection */
/* FAULTM = 1 - Fault control is enabled for all channels,
FTMEN = 1 - all registers are available for use with no restrictions.*/
FTM0_MODE |= FTM_MODE_FAULTM_MASK | FTM_MODE_FTMEN_MASK;
/* setting for Center Aligned PWM in Combine Mode *
FTM0_MOD = MODULO/2 - 1;  /* Set PWM frequency; MODULO = 25 MHz/10kHz=2500 */
FTM0_CNTIN = -MODULO/2;   /* CTNMAX = 1 - PWM update at counter in max. value */
FTM0_SYNC |= FTM_SYNC_CNTMAX_MASK;
/* SWSYNC = 1 - set PWM value update. This bit is cleared automatically */
FTM0_SYNC |= FTM_SYNC_SWSYNC_MASK;
/* Disable all channels outputs using the OUTPUT MASK feature.*/
FTM0_OUTMASK = FTM_OUTMASK_CH5OM_MASK | FTM_OUTMASK_CH4OM_MASK
             | FTM_OUTMASK_CH3OM_MASK | FTM_OUTMASK_CH2OM_MASK
             | FTM_OUTMASK_CH1OM_MASK | FTM_OUTMASK_CH0OM_MASK;
/* COMBINE = 1 - combine mode set, COMP = 1 - complementary PWM set,
DTEN = 1 - deadtime enabled, SYNCEN = 1 - PWM update synchronization enabled,
FAULTEN = 1 - fault control enabled */
FTM0_COMBINE =  FTM_COMBINE_SYNCEN0_MASK | FTM_COMBINE_DTEN0_MASK
              | FTM_COMBINE_COMP0_MASK   | FTM_COMBINE_COMBINE0_MASK
              | FTM_COMBINE_SYNCEN1_MASK | FTM_COMBINE_DTEN1_MASK
              | FTM_COMBINE_COMP1_MASK   | FTM_COMBINE_COMBINE1_MASK
              | FTM_COMBINE_SYNCEN2_MASK | FTM_COMBINE_DTEN2_MASK
              | FTM_COMBINE_COMP2_MASK   | FTM_COMBINE_COMBINE2_MASK;
/* Dead time = 2 us for 25 MHz core clock 2us/40ns */
FTM0_DEADTIME = 50;
/* Initial setting of value registers to 50 % of duty cycle */
FTM0_C0V = -MODULO/4;
FTM0_C1V =  MODULO/4;
FTM0_C2V = -MODULO/4;
FTM0_C3V =  MODULO/4;
FTM0_C4V = -MODULO/4;
FTM0_C5V =  MODULO/4;
/* ELSnB:ELSnA = 1:0 Set channel mode to generate positive PWM */
FTM0_C0SC |= FTM_CnSC_ELSB_MASK ;
FTM0_C1SC |= FTM_CnSC_ELSB_MASK ;
FTM0_C2SC |= FTM_CnSC_ELSB_MASK ;
FTM0_C3SC |= FTM_CnSC_ELSB_MASK ;
FTM0_C4SC |= FTM_CnSC_ELSB_MASK ;
FTM0_C5SC |= FTM_CnSC_ELSB_MASK ;
/* Enables the loading of the MOD, CNTIN, and CV registers with the values of their write
buffers. */
FTM0_PWMLOAD = FTM_PWMLOAD_LDOK_MASK ;
/* Enables the generation of the trigger when the FTM counter is equal to the CNTIN
register. */
FTM0_EXTTRIG |= FTM_EXTTRIG_INITTRIGEN_MASK;
FTM0_MODE |= FTM_MODE_INIT_MASK;
/* Default ADC trigger source is PDB. In case you need ADC triggered directly from FTM add
next two lines. ADC0ALTTRGEN = 1 - alternate trigger for ADC, ADC1TRGSEL = 0x8 - FTM0
trigger*/
//SIM_SOPT7 |= SIM_SOPT7_ADC0TRGSEL (0x8) | SIM_SOPT7_ADC0ALTTRGEN_MASK; /*FTM0 triggers
ADC0*/
//SIM_SOPT7 |= SIM_SOPT7_ADC1TRGSEL (0x8) | SIM_SOPT7_ADC1ALTTRGEN_MASK; /*FTM0 triggers
ADC0*/
Set system clock as source for FTM0 (CLKS[1:0] = 01) */
/* Set system clock as source for FTM0 (CLKS[1:0] = 01) */
FTM0_SC |= FTM_SC_CLKS(1);
/* PORTs for FTM0 initialization */
PORTC_PCR1 = PORT_PCR_MUX(4); /* FTM0 CH0 */
PORTC_PCR2 = PORT_PCR_MUX(4); /* FTM0 CH1 */
PORTC_PCR3 = PORT_PCR_MUX(4); /* FTM0 CH2 */
PORTC_PCR4 = PORT_PCR_MUX(4); /* FTM0 CH3 */
PORTD_PCR4 = PORT_PCR_MUX(4); /* FTM0 CH4 */
PORTD_PCR5 = PORT_PCR_MUX(4); /* FTM0 CH5 */
```

4.3.2 PDB initialization

```
/* EN = 0x01 | EN = 0x00 - first two pre/triggers are enabled
   TOS = 0x01 | TOS = 0x00 - channels assert when the counter reaches the channel
   delay register plus one peripheral clock cycle after PDB trigger  */
PDB0_CH0C1 |= PDB_C1_EN(0x1) | PDB_C1_TOS(0x1);
/* delete next line if the two-time measurement per PWM period is not required */
PDB0_CH0C1 |= PDB_C1_EN(0x2) | PDB_C1_TOS(0x2);
PDB0_CH1C1 |= PDB_C1_EN(0x1) | PDB_C1_TOS(0x1);
/* delete next line if the two-time measurement per PWM period is not required */
PDB0_CH1C1 |= PDB_C1_EN(0x2) | PDB_C1_TOS(0x2);
/* set the delay value for the channel's corresponding pre-triggers */
PDB0_CH0DLY0 = 0;    /* corresponding ADC0_RA */
PDB0_CH1DLY0 = 0;    /* corresponding ADC1_RA */
/* delete next two lines if the two-time measurement per PWM period is not required */
PDB0_CH0DLY1 = 1250; /* corresponding ADC0_RB */
PDB0_CH1DLY1 = 1250; /* corresponding ADC1_RB */
/* PDBEN = 1 - PDB enabled, TRGSEL = 0x8 - FTM0 is a trigger source for PDB,
   LDOK - writing 1 to this bit updates the internal registers*/
PDB0_SC |= PDB_SC_PDBEN_MASK | PDB_SC_TRGSEL(0x8)
        |  PDB_SC_LDOK_MASK;
```

4.3.3 ADC initialization

```
/* MODE = 0x3 - single-ended 16-bit conversion   */
ADC0_CFG1 |= ADC_CFG1_MODE(0x3);
ADC1_CFG1 |= ADC_CFG1_MODE(0x3);
/* ADHSC = 1 - very high speed operation */
ADC0_CFG2 |= ADC_CFG2_ADHSC_MASK ;
ADC1_CFG2 |= ADC_CFG2_ADHSC_MASK ;
/* ADTRG = 1 - hardware trigger selected */
ADC0_SC2  |= ADC_SC2_ADTRG_MASK ;
ADC1_SC2  |= ADC_SC2_ADTRG_MASK ;
/* AIEN = 1 - conversion complete interrupt enabled, ADCH = 0x9 - Channel 9 is selected as
SE input */
ADC0_SC1A = ADC_SC1_AIEN_MASK | ADC_SC1_ADCH(0x9);  /* corresponding ADC0_RA */
ADC1_SC1A = ADC_SC1_AIEN_MASK | ADC_SC1_ADCH(0x1);  /* corresponding ADC1_RA */
/* delete next two lines if the two-time measurement per PWM period is not required */
ADC0_SC1B = ADC_SC1_AIEN_MASK | ADC_SC1_ADCH(0x9);  /* corresponding ADC0_RB */
ADC1_SC1B = ADC_SC1_AIEN_MASK | ADC_SC1_ADCH(0x1);  /* corresponding ADC1_RB */
```

4.4 Other settings and the ADC interrupt routine

```
/* enable clock for peripherals */
SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK | SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTC_MASK
          |  SIM_SCGC5_PORTD_MASK | SIM_SCGC5_PORTE_MASK ;
SIM_SCGC6 |= SIM_SCGC6_FTM0_MASK  | SIM_SCGC6_PDB_MASK   | SIM_SCGC6_ADC0_MASK;
SIM_SCGC3 |= SIM_SCGC3_ADC1_MASK ;  /* enable interrupt ADC0, ADC1 */
NVICISER1 = 0x06000000;
/* Enable PWM outputs of FTM0 and */
FTM0_OUTMASK = 0;
/* Enables the loading of the MOD, CNTIN, and CV registers with the values of their write
buffers. */
FTM0_PWMLOAD = FTM_PWMLOAD_LDOK_MASK;
void ADC0_IRQHandler(void)
{ /* toggle PTC18 to see when ADC interrupt is called */
  GPIOC_PTOR = 1UL << 18;
  /* Save the results and clear the COCO flag */
  if ( ADC0_SC1A & ADC_SC1_COCO_MASK )
    result0RA = (unsigned short) ADC0_RA;
  /* delete next two lines if the two-time measurement per PWM period is not required */
  else if ( ADC0_SC1B & ADC_SC1_COCO_MASK )
```

```
    result0RB = (unsigned short) ADC0_RB;
}
void ADC1_IRQHandler(void)
{ /* toggle PTC18 to see when ADC interrupt is called */
  GPIOC_PTOR = 1UL << 18;
  /* Save the results and clear the COCO flag */
  if ( ADC1_SC1A & ADC_SC1_COCO_MASK )
  result1RA = (unsigned short) ADC1_RA;
  /* delete next two lines if the two-time measurement per PWM period is not required */
  else if ( ADC1_SC1B & ADC_SC1_COCO_MASK )
  result1RB = (unsigned short) ADC1_RB;
```

4.5 Measured waveforms

Both variants of the task were measured. The pin from the GPIO port C18 toggles after entering into the ADC routines. That means the rising edge corresponds to the ADC0 routine, and the falling edge corresponds to the ADC1 routine. Figure 5 demonstrates the pair of phase currents sensed when the button power components are turned on. Figure 6 demonstrates the two pairs of phase currents sensed per PWM period. In the following figures, Channel A is the PWM, shown as the upper red segment, and Channel B is the pin toggling after entering the ADC routine, shown as the lower blue segment.

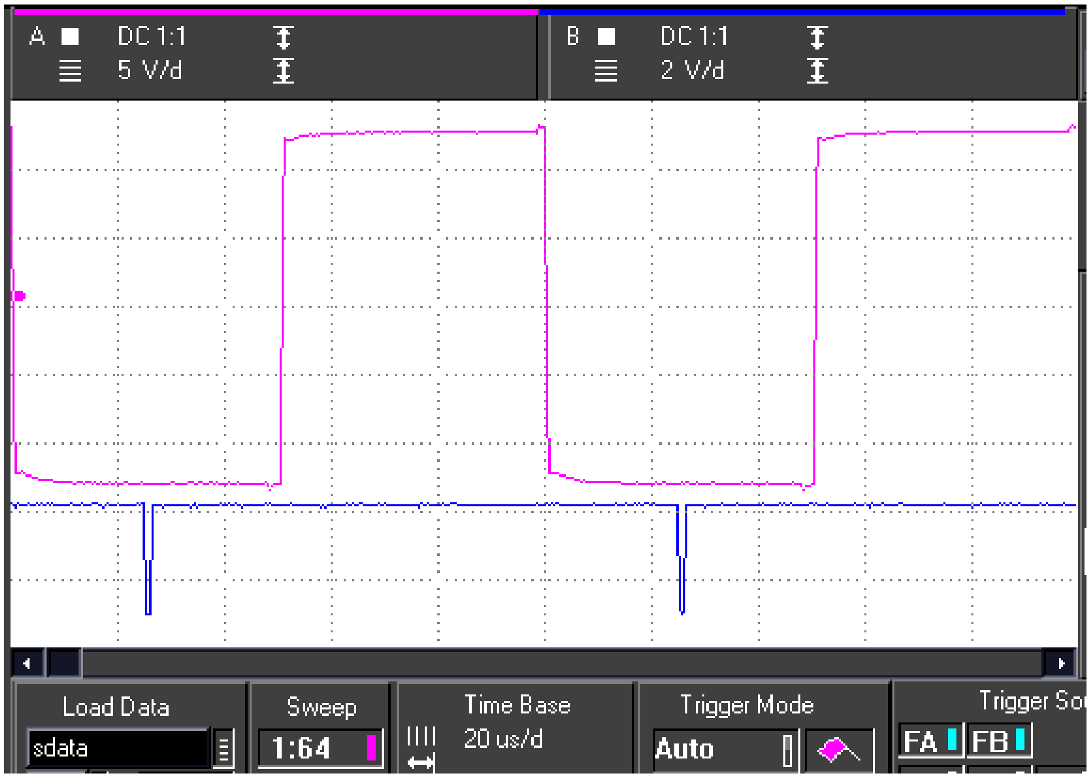

Figure 5. Measured waveforms: One pair per PWM.

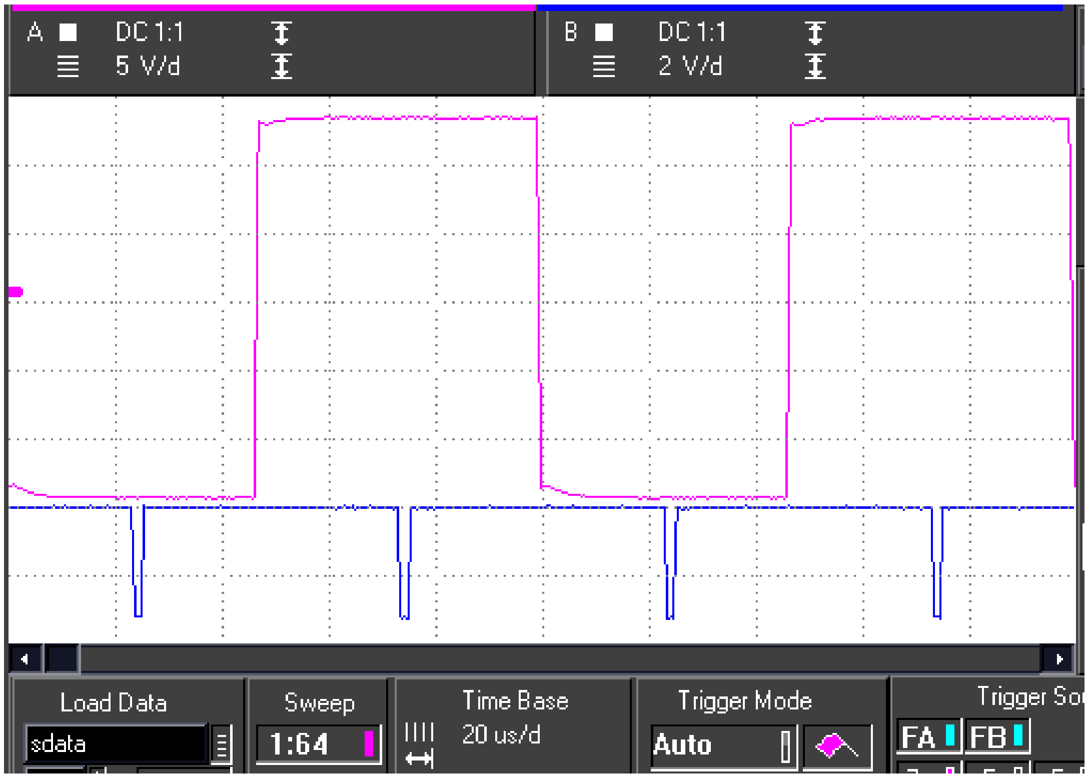

Figure 6. Measured waveforms: Two pairs per PWM

5 References

- K40 Sub-Family Reference Manual, available at http://www.freescale.com
- Using FlexTimer in ACIM/PMSM Motor Control Applications, available at http://www.freescale.com
- PMSM Vector Control with Quadrature Encoder on Kinetis Design Reference Manual, available at http://www.freescale.com

Document Number: AN4410
Rev. 0, 11/2011