Freescale Semiconductor
Application Note

Document Number: AN4558
Rev. 0, 08/2012

3-phase Sensorless BLDC Motor Control Development Kit with MC9S12G128 MCU

by: Josef Kramoliš
Rožnov pod Radhoštem, Czech Republic

1 Introduction

This application note describes the use of the MC9S12G128 microcontroller in BLDC motor control applications. It covers both sensorless and Hall sensor based BLDC motor control techniques.

The MC9S12G128 is an optimized, automotive, 16-bit microcontroller focused on low-cost, high-performance, and low-pin count. It is suited for generic automotive applications requiring CAN or LIN/SAE J2602 communication.

The concept of the sensorless control application is a speed-loop brushless direct current (BLDC) motor drive using a sensorless zero-crossing technique.

Both applications (sensorless and Hall sensor based) use the hardware of the 3-phase Sensorless BLDC Motor Control Development Kit with the MC9S12G128. In addition to the development kit, this application note refers to the following collateral materials, all of which are available at www.freescale.com:

Contents

- *MC9S12G128 Controller Board User Guide*, MC9S12G128MCBUG
- *3-phase Low-voltage Power Stage User Guide*, 3PHLVPSUG
- *MC9S12G Family Reference Manual*, MC9S12GRMV1
- The software that accompanies this document, AN4558SW

2 MC9S12G128 peripherals

Both BLDC motor control applications described in this application note use the following set of MC9S12G128 peripheral modules:

1. Pulse Width Modulator
2. Analog-to-Digital Converter
3. Timer Module
4. Port Integration Module (external pin interrupts)

MC9S12G128 peripheral features related to the BLDC motor control are described in the following sections.

2.1 Pulse-Width Modulator

There is one Pulse-Width Modulator (PWM) module available on the MC9S12G128. The PWM module provides eight PWM channels (PWM0 to PWM7) with an 8-bit resolution. As an alternative, pairs of 8-bit channels can be concatenated to form up to four PWM channels with a 16-bit resolution. Each PWM channel has its own counter that can operate up to the maximum bus clock frequency (25 MHz).

2.1.1 PWM clock select

There are four available clocks: clock A, clock B, clock SA (scaled A), and clock SB (scaled B). These four clocks are derived from the bus clock.

Both applications described further in this document use the maximal bus clock frequency of 25 MHz. With respect to the maximal PWM duty cycle resolution, clock A is used for all PWM channels in both applications. Clock A is selected using the PWMCLK register (PWMCLK = 0). Clock A rate is set to the bus clock by writing 0 to the PWMPRCLK[PCKA2-0] register bit field.

2.1.2 PWM enable

Each PWM channel has an enable bit (PWMEx) to start its waveform output. When any of the PWMEx bits are set (PWMEx = 1), the associated PWM output is enabled immediately.

Once concatenated mode is enabled, enabling/disabling of the corresponding 16-bit PWM channel is controlled by the low order PWMEx bit (see Section 2.1.4, “PWM 16-bit functions” for more details).

When the channel is disabled (PWMEx = 0), the channel counter stops counting.

2.1.3 Polarity control

The starting polarity of each PWM channel waveform is determined by the associated PPOLx bit in the PWMPOL register. When one of the bits in the PWMPOL register is set, the associated PWM channel output is high at the beginning of the waveform, then goes low when the duty count is reached. Conversely, if the polarity bit is zero, the output starts low and then goes high when the duty count is reached.

2.1.4 PWM 16-bit functions

The PWMCTL register contains four control bits, each of which is used to concatenate a pair of 8-bit channels into one 16-bit channel. Channels 6 and 7 are concatenated with the CON67 bit, channels 4 and 5 are concatenated with the CON45 bit, channels 2 and 3 are concatenated with the CON23 bit, and channels 0 and 1 are concatenated with the CON01 bit.

The double-byte channel counter can be reset by using 16-bit write access or by writing either the low or high order byte of the counter. The double byte channel counter structure is shown in Figure 1.

The low order channel registers control the behavior of the concatenated pair (e.g. channel 7 registers control the behavior of the PWM67 pair). The output of the concatenated pair is routed to the low order channel pin. See Table 1 for the 16-bit concatenation mode summary.

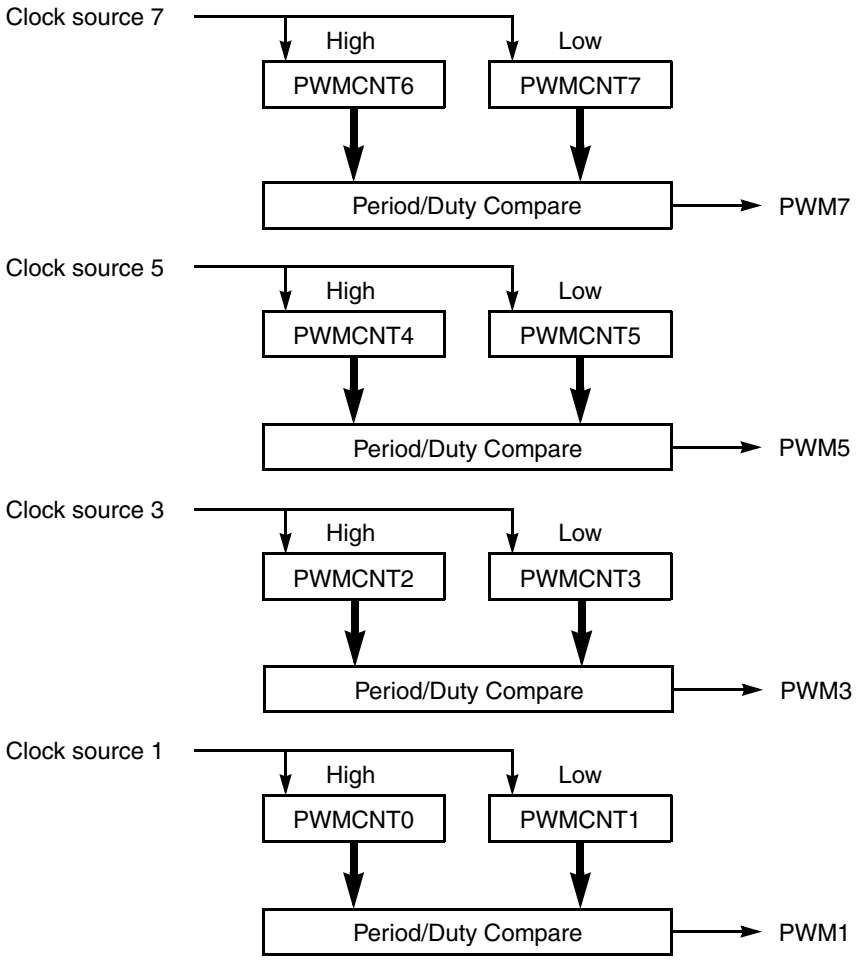


Figure 1. PWM 16-bit mode

Table 1. 16-bit concatenation mode summary

CONxx	PWMEx	PPOLx	PCLKx	CAEx	PWMx Output
CON67	PWME7	PPOL7	PCLK7	CAE5	PWM7
CON45	PWME5	PPOL5	PCLK5	CAE5	PWM5
CON23	PWME3	PPOL3	PCLK3	CAE3	PWM3
CON01	PWME1	PPOL1	PCLK1	CAE1	PWM1

2.1.5 PWM period and duty

This section describes left-aligned PWM generation using the concatenated 16-bit channel.

There are dedicated duty (PWMDTYx) and period (PWMPERx) registers for each channel. Both registers are double buffered so that if they are changed while the channel is enabled, the change will not take effect until the effective PWM period ends, the channel counter is written (counter resets to 0x00), or the channel is disabled (PWMEx = 0).

In left-aligned output mode (PWMCAE[CAEx] = 0), the 16-bit counter operates as an up counter. It is compared to two registers, a duty register and a period register. When the PWM counter matches the duty register, the output flip-flop changes state. A match between the PWM counter and the period register resets the counter and the output flip-flop to the state defined by the polarity control bit PPOLx. Additionally, a loading of the duty and period is performed from the double buffer to the duty and period registers. The counter counts from 0 to the value in the period register – 1.

Both BLDC motor control applications described further in the document use the MC9S12G128 maximal bus clock frequency of 25 MHz and the 20 kHz PWM waveform frequency. The PWMPERx register values for the left aligned output can be calculated as follows:

PWMPERx = channel clock frequency ÷ PWMx period

PWMPERx = 25 MHz ÷ 20 kHz = 1250 = 0x04E2

The PWMDTYx register value for left-aligned PWM, PPOLx = 1 and a 50% duty cycle can be calculated as follows:

PWMDTYx = Duty Cycle × PWMPERx

PWMDTYx = 0.5 ×1250 = 625 = 0x028A

See Figure 2 for the left-aligned PWM example with a 25% duty cycle with an update to a 50% duty cycle in the first PWM period.

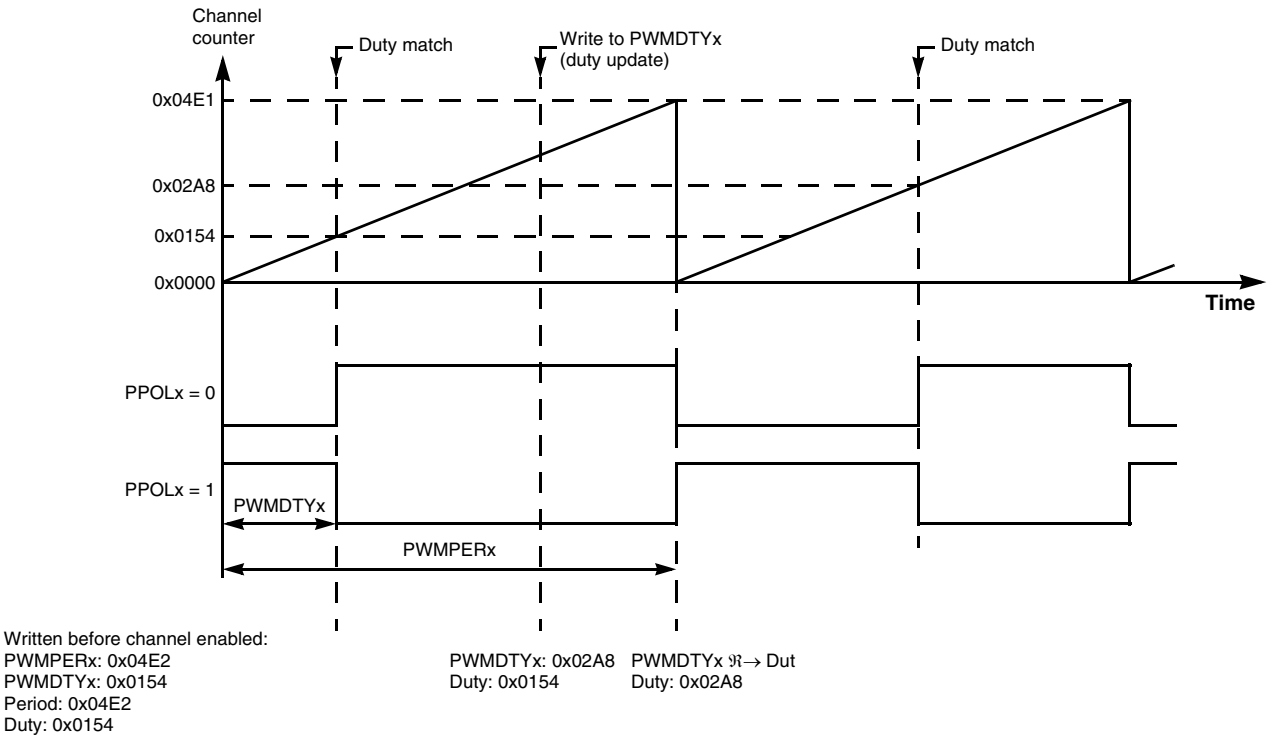


Figure 2. Left-aligned PWM 16-bit example

2.1.6 PWM module initialization example

The following example shows PWM module initialization for both BLDC motor control applications covered in this document. Example 1 corresponds to the PWM and split gate control signals shown in Figure 4 (for details see Section 3.1, “PWM generation”).

Example 1. PWM module initialization

```
#define PWM_MODULO        0x04E2     /* 50us @25MHz bus clock */
#define PWM_DISABLE       0x00
#define PWM_ENABLE        0xAA

PWME = PWM_DISABLE;       /* Disable all PWM channels */

PWMPOL = 0x00;            /* All PWM channels start low */
PWMCLK = 0x00;            /* All channels are clocked from A or B clock source */
PWMPRCLK = 0x00;          /* Clock A & B equal to BUS clock */
PWMCAE = 0x00;            /* PWM1, PWM3, PWM5, PWM7 are left aligned */
PWMCTL = 0xF4;            /* All channels are concatenated to form 16-bit PWMs,
                          counters stop in Freeze mode */
PWMCLKAB = 0x00;          /* Clock A is the source clock for all channels */

PWMPER01 = PWM_MODULO;    /* Set PWM channel 01 period (ADC ext. trigger) */
PWMPER23 = PWM_MODULO;    /* Set PWM channel 23 period (phase A) */
PWMPER45 = PWM_MODULO;    /* Set PWM channel 45 period (phase B) */
PWMPER67 = PWM_MODULO;    /* Set PWM channel 67 period (phase C) */
          ...
          ...
          ...
PWME = PWM_ENABLE;        /* Enable all PWM channels */
```

2.2 ADC module

There is a 16-channel, 10-bit successive approximation Analog-to-Digital Converter (ADC) module available on the MC9S12G128. All voltages to be converted have to be in the range defined by reference voltages applied on the VRH and VRL pins. In the case of the 3-phase Sensorless Motor Control Development Kit with MC9S12G128 MCU, the range is 0 to 3.3 V.

2.2.1 ADC clock control

The ADC module clock is derived from the bus clock. The ADC clock frequency f_{ADCCLK} is limited in the range of 0.25 to 8.0 MHz to meet the ADC specification. To operate in the f_{ADCCLK} range, the bus clock may have to be prescaled. The ADC clock prescaler is configurable via the ATDCTL4[PRS4:0] field. The resulting ADC clock frequency can be calculated by Equation 1:

$$f_{ATDCLK} = \frac{f_{BUS}}{2 \times (PRS + 1)} \qquad \textit{Eqn. 1}$$

To achieve the best ADC performance with respect to the 25 MHz bus clock, and to meet the 8 MHz ADC clock frequency limit, the ATDCTL4[PRS] value needs to be set as follows:

$$PRS \geq \frac{f_{BUS}}{2 \times f_{ATDCLKMAX}} - 1 \qquad \textit{Eqn. 2}$$

$$PRS \geq \frac{25\ \text{MHz}}{2 \times 8\ \text{MHz}} - 1 \Rightarrow PRS \geq 0.5625 \Rightarrow PRS = 1 \qquad \textit{Eqn. 3}$$

$$f_{ATDCLK} = \frac{f_{BUS}}{2 \times (PRS + 1)} = \frac{25\ \text{MHz}}{2 \times (1 + 1)} = 6.25\ \text{MHz} \qquad \textit{Eqn. 4}$$

2.2.2 External trigger control

The external trigger feature allows you to synchronize ADC conversion to an external event rather than relying only on software to trigger the ADC module when a conversion is about to take place. The external trigger signal input is programmable to be edge- or level-sensitive with polarity control.

Since the BLDC sensorless motor control applications use the PWM output as an ADC external trigger signal (see Section 3.2.1, “Back-EMF voltage sensing”), the external trigger is configured with rising edge sensitivity using ATDCTL2[ETRIGLE] = 0 and ATDCTL2[ETRIGP] = 1.

The external trigger ADC input is selectable using ATDCTL1[ETRIGSEL] and ATDCTL1[ETRIGCH3:0]. See Table 2 for configuration options. The BLDC sensorless application uses input ETRIG1.

Table 2. External trigger input select coding

ETRIGSEL	ETRIGCH3	ETRIGCH2	ETRIGCH1	ETRIGCH0	External trigger source
0	0	0	0	0	AN0
0	0	0	0	1	AN1
0	0	0	1	0	AN2
0	0	0	1	1	AN3
0	0	1	0	0	AN4
0	0	1	0	1	AN5
0	0	1	1	0	AN6
0	0	1	1	1	AN7
0	1	0	0	0	AN8
0	1	0	0	1	AN9
0	1	0	1	0	AN10
0	1	0	1	1	AN11
0	1	1	0	0	AN12
0	1	1	0	1	AN13
0	1	1	1	0	AN14
0	1	1	1	1	AN15

Table 2. External trigger input select coding (continued)

ETRIGSEL	ETRIGCH3	ETRIGCH2	ETRIGCH1	ETRIGCH0	External trigger source
1	0	0	0	0	ETRIG0
1	0	0	0	1	ETRIG1
1	0	0	1	0	ETRIG2
1	0	0	1	1	ETRIG3
1	0	1	X	X	Reserved
1	1	X	X	X	Reserved

Once ATDCTL2[ETRIGE] is set, ATDCTL5 must be written to enable the servicing of the external trigger events by the ADC.

2.2.3 Sample and conversion control

The ADC Sample and Hold machine controls the storage and charge of the sample capacitor to the level of the analog signal at the selected ADC input channel. The sampling time (in the number of ADC clock cycles) is configurable using the ATDCTL4[SMP2:0] register bit field. To obtain the best performance of the ADC module, the sample time is set to 4 ATD clock cycles (ATDCTLS4[SMP2:0] = 0).

The ADC resolution is selectable by the ATDCTL1[SRES1:0] register bit field. The left or right result justification in the 16-bit result register can be configured with the ATDCTL3[DJM] bit. 10-bit resolution is suitable for both BLDC motor control applications described further in the document.

The total conversion time of the channel (10-bit resolution) in ADC clock cycles can be calculated as follows:

$$N_{CONV10} = N_{DIS} + N_{SAMPLE} + 15 \qquad \textit{Eqn. 5}$$

In this equation, N_{DIS} is equal to the two ADC clock cycles required to discharge the sampling capacitor if set by ATDCTL1[SMP_DIS]. N_{SAMPLE} is the sample time in ADC clock cycles. Since the capacitor discharge feature is not used in the application, the resulting conversion time is equal to 19 ADC clock cycles (3.04 µs at f_{ATDCLK} = 6.25 MHz).

The ATDCTL3[S8C,S4C,S2C,S1C] fields control the number of conversions within one conversion sequence. The value of the field corresponds with the number of conversions per sequence. If the value is set to 0, the maximal number of 16 conversions is performed in sequence.

The ATDCTL5[MULT] register bit selects whether the ADC module samples only from a single analog input channel for an entire conversion sequence (MULT = 0) or samples across channels (MULT = 1). The analog input channel is selected by channel selection code in the ATDCTL5[CD,CC,CB,CA] register bit field. If MULT is 1, the ADC samples the channel selected by ATDCTL5[CD,CC,CB,CA] first. Subsequent channels sampled in the sequence are determined by incrementing the channel selection code or by wrapping around to channel 0. The last channel converted before wraparound to channel 0 is selected by the ATDCTL0[WRAP3:0] register bit field.

The ATDCTL5[SCAN] bit selects whether conversion sequences are performed continuously (SCAN = 1) or only once (SCAN = 0).

If the external triggered conversion is not enabled (ATDCTL2[ETRIGE] = 0), the conversion sequence can be triggered by software by writing to the ATDCTL5 register. This aborts the current conversion sequence and starts a new one. Start of conversion means the beginning of the sampling phase.

The completion of the conversion sequence is signalized by the ATDSTAT[SCF] flag. The conversion sequence results are then available in the ATDDRx registers. The result of the first conversion appears in the first result register (ATDDR0), the second result in the second result register (ATDDR1), and so on.

2.2.4 Interrupts

If the ATDCTL2[ASCIE] bit is set, the ADC Sequence Complete interrupt may be invoked whenever the SCF is set. To properly service the ADC Sequence Complete interrupt, the SCF flag has to be cleared in the interrupt service routine. This can be achieved by writing 1 to SCF, writing to ATDCTL5 (start of a new conversion sequence), or, if ATDCTL[AFFC] is set (ADC fast flag clearing), by reading the result register.

2.2.5 ADC module initialization examples

Example 2 shows ADC module initialization for both BLDC motor control applications covered in this document.

Example 2. ADC module initialization

```
ATDCTL0 = 0x0F;     /* Wrap around channel 15 */
ATDCTL1 = 0xA1;     /* ETRIG1 selected as external trig. source, 10-bit resolution */
ATDCTL2 = 0x00;     /* External trigger disabled, sequence complete interrupt disabled */
ATDCTL3 = 0x0B;     /* Left justified, single conversion seq. length, FIFO off,
                    freeze immediately */
ATDCTL4 = 0x01;     /* 4 ADC clock sample time, 6.25MHz ADC clock @BUSclk=25MHz
                    (4+15 ADC clocks sampling+conversion) */
ATDSTAT0 = 0xB0;    /* Clear status register 0 flags */
ATDCMPE = 0x00;     /* ATD compare disabled on all channels */
ATDDIEN = 0x0000;   /* Digital input buffer disabled on all channel pins */
```

Example 3 and Example 4 show the initialization of the ADC externally triggered conversion sequence and start of the software controlled conversion.

Example 3. Initialization of externally triggered conversion sequence

```
#define ADC_S_MASK          0x78
#define ATDSTAT0_SCF_MASK   0x80
#define ADC_MULT_MODE       0x10

void ADC_EnableTrigSeq(uint8_t u8channel, uint8_t u8length)
{
      ATDCTL3 &= ~ADC_S_MASK;               /* Reset conversion sequence length */
      ATDCTL3 |= ((u8length & 0x0F) << 3);  /* Set sequence length */
      ATDSTAT0 = ATDSTAT0_SCF_MASK;         /* Clear sequence complete flag */
      ATDCTL2 = 0x0E;                       /* External trigger enabled (rising edge),
                                            sequence complete interrupt enabled */
      ATDCTL5 = (ADC_MULT_MODE | u8channel);
}
```

Example 4. Start of software controlled conversion

```
#define ADC_S_MASK         0x78
#define ATDSTAT0_SCF_MASK  0x80

void ADC_StartSingleConversion(uint8_t u8channel)
{
      ATDCTL2 = 0x00;                 /* Disable external trigger and interrupts */
      ATDCTL3 &= ~ADC_S_MASK;         /* Reset conversion sequence length */
      ATDCTL3_S1C = 1;                /* Set conversion sequence length to 1 */
      ATDSTAT0 = ATDSTAT0_SCF_MASK;   /* Clear sequence complete flag */
      ATDCTL5 = u8channel;            /* Start conversion */
}
```

2.3 Timer module

The 8-channel Timer Module (TIM) with the shared up-counting 16-bit counter is available on the MC9S12G128. The timer counter counts a prescaled bus clock (up to 25 MHz).

2.3.1 Prescaler

The TIM module contains two selectable prescalers. The first prescaler divides the bus clock by 1, 2, 4, 8, 16, 32, 64, or 128. The prescaler divisor factor can be configured using the TSCR2[PR2:0] bit field value. The second prescaler divides the bus clock by 1, 2, 3, 4,...255, or 256. Its prescaler division factor is configurable by the PTPSR register bits. The TSCR1[PRNT] bit provides the selection between these two prescalers. If cleared, only the first prescaler is enabled. If set, only the second prescaler is enabled.

The timer counter is enabled by setting the TSCR1[TEN] bit.

2.3.2 Input capture

Clearing the TIOS[IOSx] bit configures channel x as an input capture channel. When an active edge occurs on the pin of an input capture channel, the timer transfers the value of the 16-bit TIM counter into the 16-bit register TCx. The edge detector circuit can be configured with the TCTL3/TCTL4[EDGGxB:EDGxA] bits, as shown in Table 3.

Table 3. Edge detector circuit configuration

EDGxB	EDGxA	Configuration
0	0	Capture disabled
0	1	Capture on rising edges only
1	0	Capture on falling edges only
1	1	Capture on any edge (rising or falling)

An input capture on channel x sets the TFLG1[CxF] flag. The TIE[CxI] bit enables the CxF flag to generate interrupt requests. The timer module must remain enabled (TSCR1[TEN] = 1) while clearing CxF (writing one to clear).

2.3.3 Output compare

Setting the TIOS[IOSx] bit configures channel x as an output compare channel. When the timer counter reaches the value in the 16-bit TCx register, the timer can set, clear, or toggle the channel pin. The compare result output action can be configured using the TCTL1/TCTL2[OMx:OLx] register bits, as shown in Table 4.

Table 4. Compare result output action

OMx	OLx	Action
0	0	No output compare action on the timer output pin
0	1	Toggle channel x pin output level
1	0	Clear channel x pin output level to zero
1	1	Set channel x pin output level to one

The TIE[CxI] bit enables the CxF flag to generate interrupt requests. The timer module must remain enabled (TSCR1[TEN] = 1) while clearing CxF (writing one to clear).

2.3.4 TIM module initialization example

Example 5 shows TIM module initialization for the sensorless BLDC motor control application. The application uses channel 0 in input capture mode. Channels 1 and 2 operate in output compare mode. Channel 1 is configured to generate an interrupt request on a compare event. The bus clock is divided by the factor of 25 to achieve a 1 MHz counter clock. Input parameters define the compare values for channels 1 and 2.

Example 5. TIM module initialization

```
void TIM_Init(uint16_t u16tc1, uint16_t u16tc2)
{
      TIOS = 0x06;      /* Channels 1 & 2 act as output compare */
      TSCR1 = 0x68;     /* Disable timer in wait and freeze modes, precision timer enabled */
      TCTL1 = 0x00;     /* No output compare action on IOC[7..4] pins */
      TCTL2 = 0x00;     /* No output compare action on IOC[3..0] pins */
      TCTL3 = 0x00;     /* Capture disabled on IOC[7..4] pins */
      TCTL4 = 0x02;     /* Capture disabled on IOC[3..1], enabled on IOC0 (falling edge) */
      TIE = 0x02;       /* Channel 1 output compare interrupt enabled */

      TC1 = u16tc1;     /* Channel 1 output compare counter value */
      TC2 = u16tc2;     /* Channel 2 output compare counter value */

      OCPD = 0xFF;      /* All output compare pins disconnected */
      PTPSR = 0x18;     /* Timer prescale factor 25. TIM running at 1MHz @BUSclk=25MHz */
      TFLG1 = 0xFF;     /* Clear all channels OC/IC event flags */

      TSCR1_TEN = 1;    /* Enable TIM */
}
```

2.4 Pin interrupts

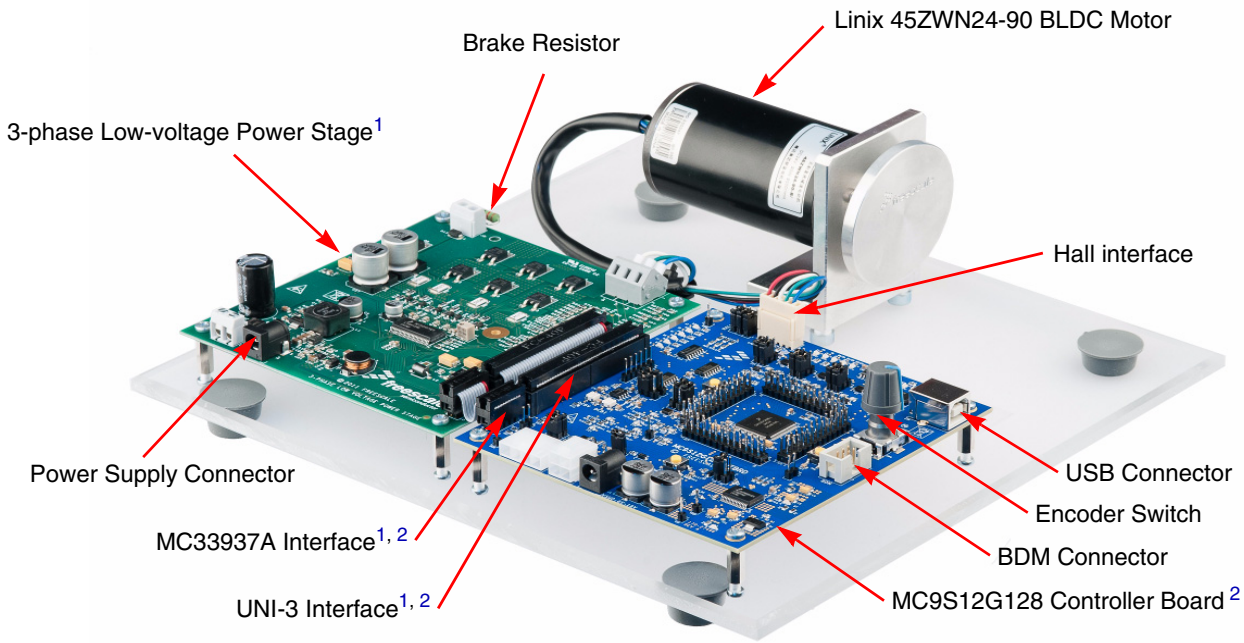
Ports P, J, and AD offer pin interrupt capability covered by the Port Integration Module (PIM). The related interrupt enable (PIE) bit in the PIEP, PIEJ, PIE0AD, and PIE1AD registers, as well as the sensitivity to rising or falling edges (PPS) in the PPSP, PPSJ, PPS0AD, and PPS1AD registers, can be individually configured on a per-pin basis. All bits/pins in a port share the same interrupt vector. Interrupts can be used with the pins configured as inputs or outputs.

An interrupt is generated when a port interrupt flag (PIF) and its corresponding port interrupt enable (PIE) bit in the PIFP, PIFJ, PIF0AD, or PIF1AD registers are both set. Writing one to the corresponding PIF bit clears the flag.

A digital filter on each pin prevents short pulses from generating an interrupt. A valid edge on an input is detected if 4 consecutive samples of a passive level are followed by 4 consecutive samples of an active level. Otherwise, the sampling logic is restarted. An input sample rate is equal to the bus clock.

3 BLDC sensorless motor control application using the MC9S12G128

This section provides an example of how to use the PWM, ADC, and TIM module in a BLDC sensorless motor control application. This application example is compatible with the hardware of the 3-phase Sensorless BLDC Motor Control Development Kit with MC9S12G128 MCU.



[1] For further information, refer to the *3-phase Low-voltage Power Stage User Guide,* 3PHLVPSUG.

[2] For further information, refer to the *MC9S12G128 Controller Board User Guide*, MC9S12G128MCBUG.

Figure 3. 3-phase BLDC Motor Control Development Kit with MC9S12G128 MCU

3.1 PWM generation

To benefit from the maximal PWM resolution, all PWM channels are configured in left-aligned PWM generation mode using concatenated 16-bit channels. Since the BLDC motor control requires six PWM driving signals, additional logic gates are provided on the MC9S12G128 Controller Board, as shown in Figure 4.

The 16-bit PWM channels PWM3, PWM5, and PWM7 generate high-side driving signals for all three motor phases. The low-side PWM driving signals are derived from the high-side PWM signals and can be forced to logic 0 by the MCU general-purpose outputs PA0, PA1, and PA2. These additional AND gates allow the BLDC motor power stage to be driven with only three PWM channels. This solution, however, disallows the MC9S12G128's ability to control dead time with the PWM module. The MC33937A FET predriver on the 3-phase Low-voltage Power Stage is responsible for the dead time control. For further information, see Section 3.1.1, "Dead time control".

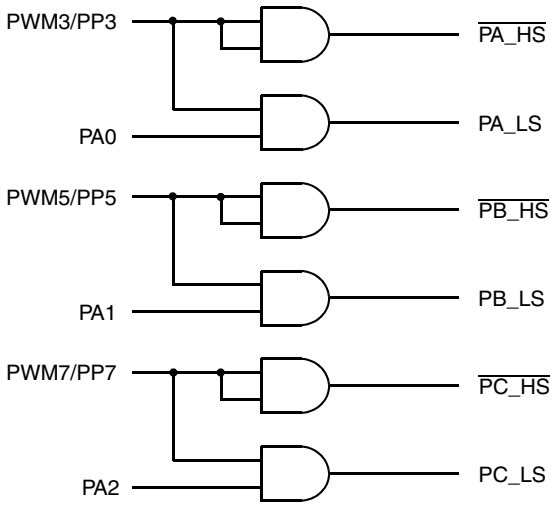


Figure 4. High-side and low-side PWM signal split gates

Figure 5 shows the 3-phase H-bridge with high-side and low-side switches. Figure 6 shows the PWM generated waveform with six commutation vectors including the low-side gating signals applied to the AND gates and phase voltages. The PWM polarity corresponds with the MC33937A PWM input polarity specification.

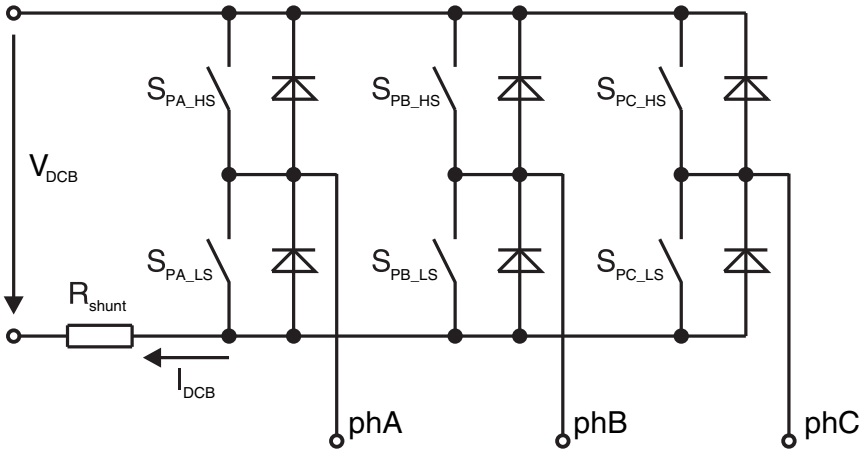


Figure 5. 3-phase H-bridge

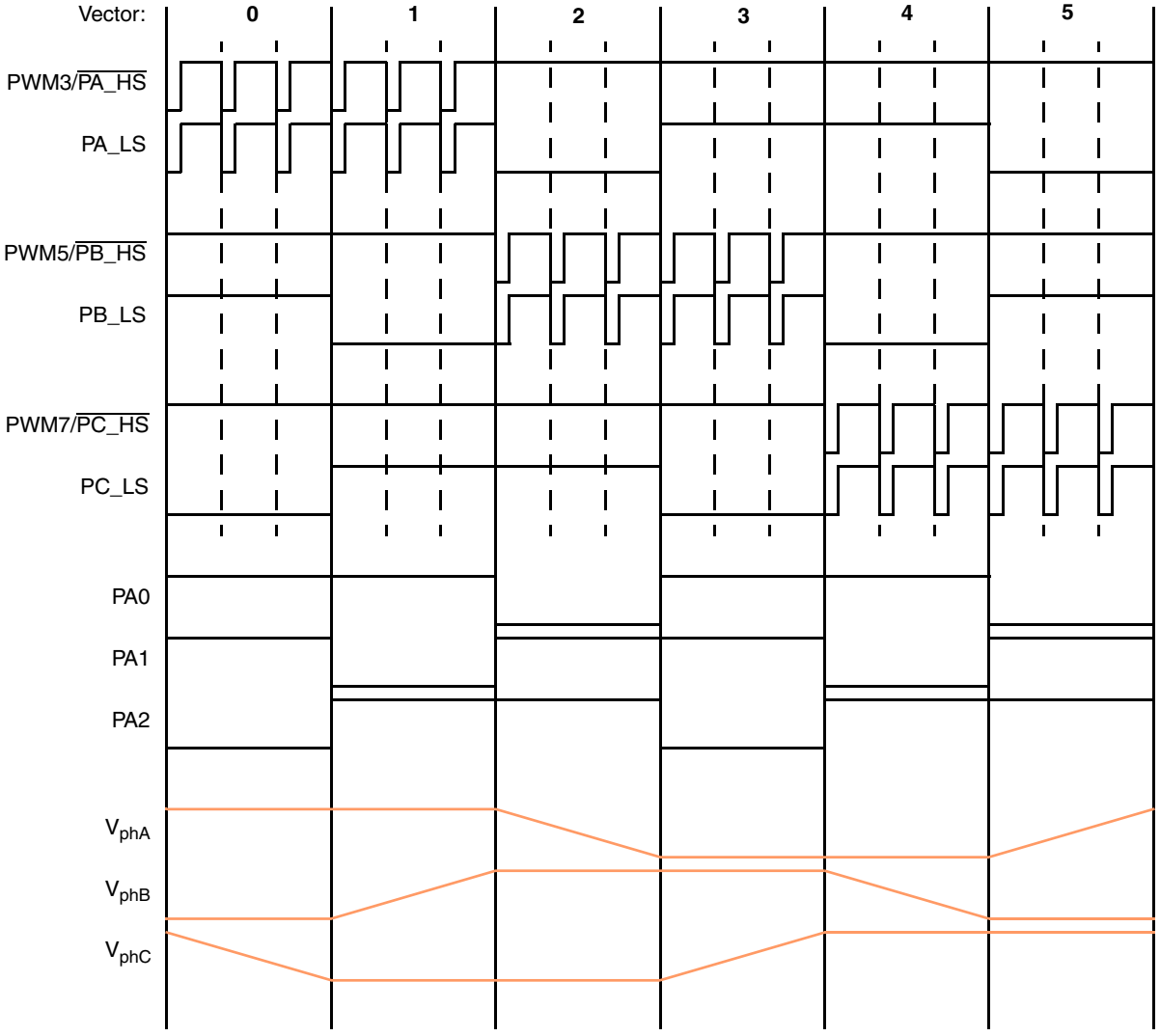


Figure 6. PWM generation waveforms

During one commutation period, one motor phase is powered by a complementary PWM signal driving both the high-side and low-side MOSFET transistors, while the second phase is grounded by the low-side transistor which is on during the whole commutation period. The third phase is left unpowered and is used for the back electromotive force (Back-EMF) signal measurement by the ADC for the zero-cross detection. The six-step commutation sequence is summarized in Table 5.

Table 5. Clockwise direction commutation sequence

Rotor position	Vector number	Switch closed		Commutation vector		
				Phase A	Phase B	Phase C
0–60°	0	S_{PA_HS}	S_{PB_LS}	$+V_{DBC}$	$-V_{DBC}$	NC
60–120°	1	S_{PA_HS}	S_{PC_LS}	$+V_{DBC}$	NC	$-V_{DBC}$
120–180°	2	S_{PB_HS}	S_{PC_LS}	NC	$+V_{DBC}$	$-V_{DBC}$
180–240°	3	S_{PB_HS}	S_{PA_LS}	$-V_{DBC}$	$+V_{DBC}$	NC
240–300°	4	S_{PC_HS}	S_{PA_LS}	$-V_{DBC}$	NC	$+V_{DBC}$
300–360°	5	S_{PC_HS}	S_{PB_LS}	NC	$-V_{DBC}$	$+V_{DBC}$

Since the actual position of the rotor is not known before the motor rotation starts, the rotor needs to be aligned into a defined position. During the alignment process, phase C is connected to the positive DC bus voltage, while phases A and B are connected to the negative DC bus voltage. The alignment time depends on the mechanical constant of the motor including the load. Once aligned, the six-step commutation process may be started by applying commutation vector 0 (ensures a proper variation of the stator field angle). See Figure 17 and Figure 18 for an illustration.

Table 6. Alignment vector

Switch closed			Alignment vector		
			Phase A	Phase B	Phase C
S_{PA_LS}	S_{PB_LS}	S_{PC_HS}	$-V_{DBC}$	$-V_{DBC}$	$+V_{DBC}$

Example 6 shows the commutation vector function for vector 0. The example corresponds to the PWM and split gate control signals shown in Figure 4.

NOTE

During the execution of the commutation vector function, the PWM outputs need to be temporarily disabled (switched to a constant logic high). Once a PWM channel is disabled, the corresponding output pin is controlled by the PIM module. Therefore, the PWM output pins driving the high-side switches need to be pre-configured as outputs with a high output logic level in the corresponding Data Direction Register and Port Data Register in the PIM initialization.

Example 6. Commutation vector 0 function

```
/* Phase A complementary PWM (zero dead time) */
/* Phase B TOP off, BOTTOM on */
/* Phase C TOP off, BOTTOM off */

#define PWM_DISABLE      0x00
#define PWM_LS_RESET     0xF8
#define PWM_DISABLE      0x00
#define PWM_LS_SECT0     0x82
#define PWM_HS_SECT0     0x03

PWME = PWM_DISABLE;      /* Disable all high-side PWMs (driven logic high by PIM) */
PORTA &= PWM_LS_RESET;   /* Disable low-side PWMs */

PWMCNT1 = 0x0000;        /* Reset PWM1 counter (ADC trigger signal) */
PWMCNT3 = 0x0000;        /* Reset PWM3 counter (phase A high-side PWM) */

PORTA |= PWM_LS_SECT0;   /* Enable phase A & B low-side switches to be
                         driven by high-side PWM signals */
PWME |= PWM_HS_SECT0;    /* Enable phase A high-side PWM and ADC trigger PWM */
```

3.1.1 Dead time control

Since the complementary PWM dead time control cannot be handled by the MCU due to the limited number of 16-bit PWM channels, the dead time is controlled either by the MC33937A FET pre-driver on the 3-phase Low-voltage Power Stage or by the RC cell with diode present on the MC9S12G128 Controller Board.

The MC33937A FET pre-driver applies the dead time to the MOSFET gate driving signals $\overline{\text{Px\_HS}}$ and Px_LS. The dead time is configurable by the DEADTIME macro value in the MC33937_routines.h file (please refer to the AN4558SW). The MC33937A is configurable via SPI. Figure 7 shows MC33937A handled dead time generation.

RC cells provide generation of the $\overline{\text{Px\_HS}}$ and Px_LS signals with the dead time already applied, so the $\overline{\text{Px\_HS}}$ and Px_LS look the same as signals Px_HS_G and Px_LS_G in Figure 7. See the MC9S12G128 Controller Board User Manual for jumper configuration.

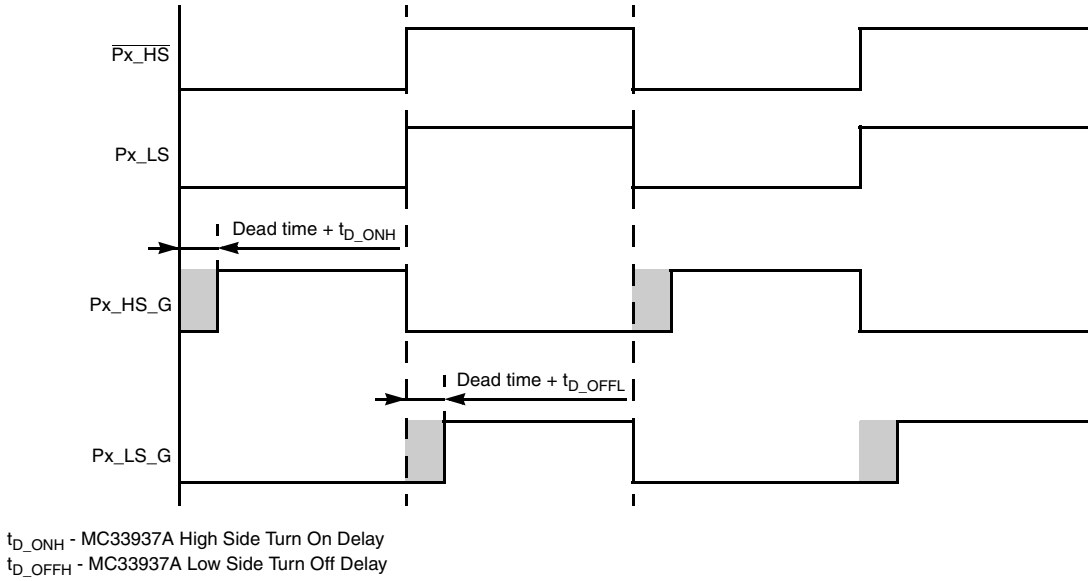


Figure 7. MC33937A dead time control

3.2 Back-EMF processing

For the sensorless BLDC application, it is necessary to sense the following parameters during the application run:

- DC bus voltage
- Phase A, B, and C Back-EMF voltages
- DC bus current for current and torque limiting

The commutation time and period are calculated from the first two parameters measured.

3.2.1 Back-EMF voltage sensing

In sensorless applications with a BLDC motor, only two phases are powered during each commutation period. The third phase is always unpowered. Because the motor shaft is rotating and there are permanent magnets on the rotor producing a magnetic field, there is a Back-EMF voltage generated on the phase winding during the rotor motion. This voltage can be measured on the currently unpowered phase by the ADC module. The actual rotor position can be evaluated from this information. Back-EMF voltage can be positive or negative depending on the actual rotor position, therefore it is necessary to measure this voltage when the other two motor phases are powered. The measured Back-EMF voltage can then be evaluated as:

- Positive when Back-EMF voltage > DC bus voltage ÷ 2
- Negative when Back-EMF voltage < DC bus voltage ÷ 2

NOTE

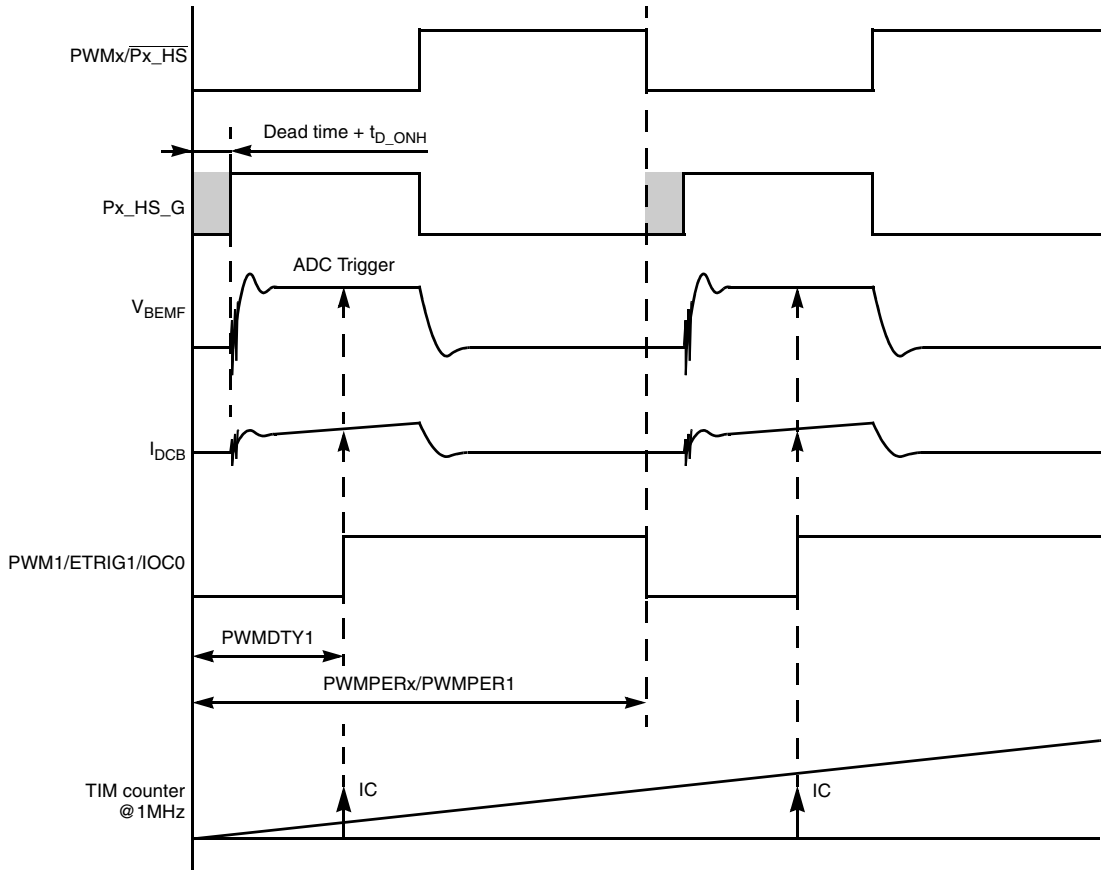
It is important to know that if the Back-EMF voltage generated by the motor is zero (measured Back-EMF voltage is equal to DC bus voltage ÷ 2 during the active PWM pulse on the output) then the motor shaft is just in the middle of two commutations. A principle of the sensorless motor control application is simply to find the Back-EMF voltage zero-cross. The time of the next commutation can be then easily calculated from the times of the last two zero-cross events.

An ADC sample point needs to be set for Back-EMF sensing. The ADC sample point depends on the parameters of the application: PWM dead time, analog circuitry delay, power component switching noise, motor coil transient due to PWM switching, and motor parameters. The ADC sample point can be precisely synchronized from the PWM module. The fourth 16-bit PWM channel (PWM1) is used as the ADC trigger signal. The PWM1 output is internally linked to the ADC ETRIG1 input. Channel PWM1 operates in the 20 kHz left-aligned output PWM mode with the same polarity as PWM7/PWM5/PWM3. The duty cycle of the channel PWM1 (register PWMDTY1) defines the delay of the Back-EMF sample point from the start of the PWM period (active rising edge).

The Back-EMF voltage is measured once every PWM period, that is once every 50 µs in the case of a 20 kHz PWM frequency.

The PWM1 signal is also externally fed into the TIM IOC0 input pin. The TIM counter value is captured by the TIM channel 0 on the PWM1 rising edge. Once the zero-cross event is detected, the last captured value of the TIM counter is the time stamp of the zero-cross event (last ADC trigger).

Figure 8 shows the timing relation between the PWM1 trigger signal, PWM7/PWM5/PWM3 outputs, related MOSFET transistor gate driving signal generated by the MC33937A, Back-EMF voltage, DC bus current, and the TIM counter input capture events.



t_{D_ONH} - MC33937A High Side Turn On Delay
IC - TIM channel 0 input capture

Figure 8. Back-EMF voltage measurement

The external ADC trigger signal (PWM1) triggers the conversion sequence of the following voltages:

1. Back-EMF voltage
2. DC bus current (amplified DC bus shunt voltage)

When these channels are converted, the ADC Sequence Complete interrupt is invoked. The interrupt service routine then reads the Back-EMF and DC bus current conversion results and starts the DC bus voltage conversion (external trigger disabled). Once the DC bus voltage conversion is complete, the external trigger is re-enabled.

NOTE

During short duty cycles, the PWM pulse is not wide enough to get the Back-EMF conversion and DC bus current sampling done during the pulse duration (640 ns sample time, 3.04 µs sample & conversion time). Because the Back-EMF voltage needs to be converted in each PWM cycle to detect a zero-cross event, the DC bus current conversion result may have to be ignored during short duty cycles (DC bus current conversion out of the PWM pulse or in a transient will return an inappropriate result).

3.2.2 Zero-cross processing

As described in Section 3.2.1, “Back-EMF voltage sensing”, the Back-EMF is sensed every 50 µs. Therefore, the zero-cross event can be found with a 50 µs precision. This is very inaccurate for high-speed applications. For example, if the commutation period takes only 200 µs, then there are only four periods during the commutation period, and a 50 µs precision in finding a zero-cross event is inaccurate. In that case, the interpolation between the two Back-EMF ADC samples can be used to calculate the zero-cross event time more precisely. One Back-EMF voltage ADC value just before the zero-cross event, and one Back-EMF voltage ADC value just after the zero-cross event are needed to precisely recalculate the zero-cross event time. See Figure 9 and Figure 10. This interpolation technique is implemented in the accompanying software, AN4558SW. A different calculation is used for rising and falling Back-EMF (commutation sector dependent) ADC samples interpolation.

3.2.2.1 Rising Back-EMF interpolation

Figure 9 illustrates the shape of the falling Back-EMF voltage.

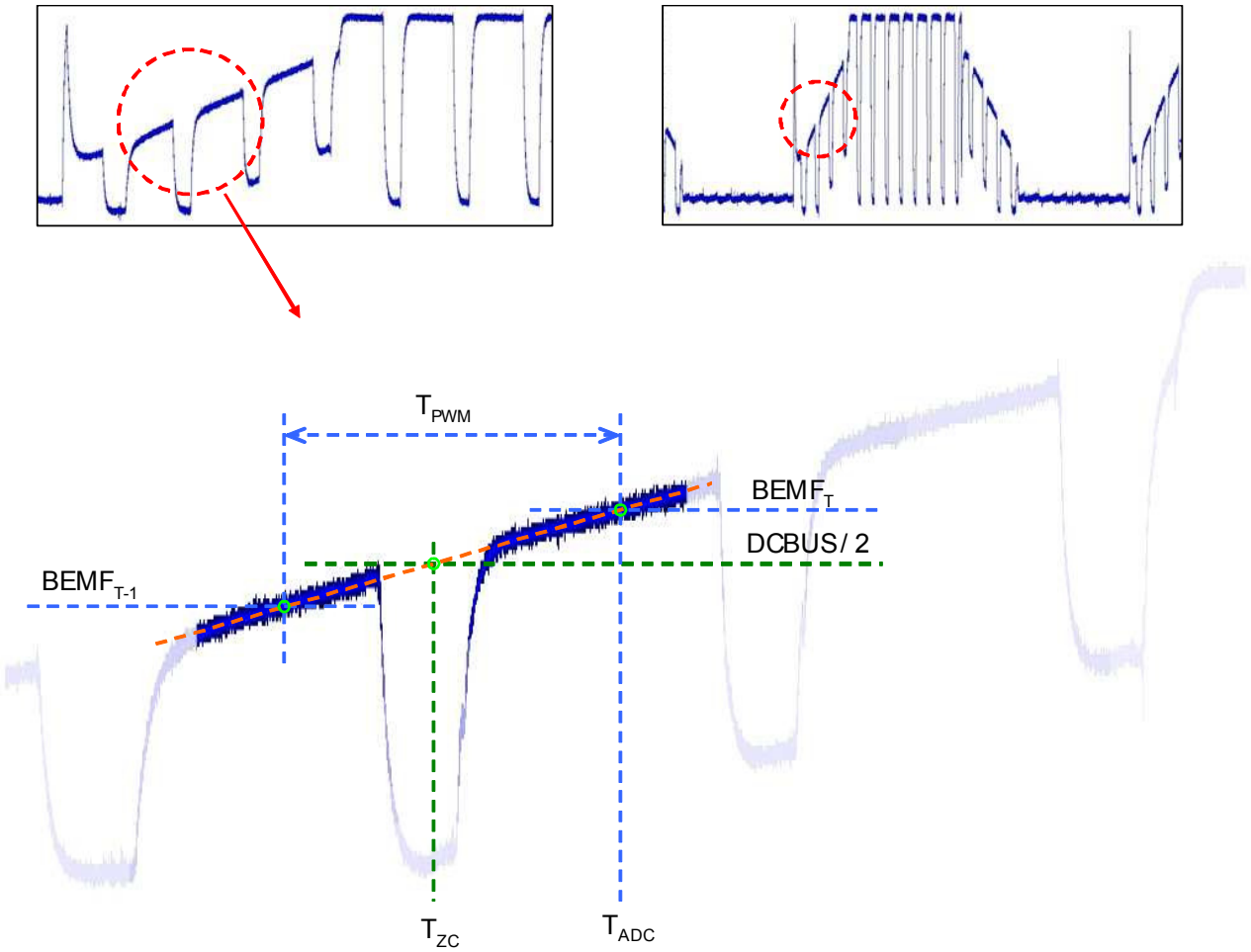


Figure 9. Rising Back-EMF voltage

The zero-cross event can be calculated using the rising Back-EMF interpolation used in Equation 6:

$$T_{ZC} = T_{ADC} - \frac{BEMF_T - DCBUS/2}{BEMF_T - BEMF_{T-1}} \times T_{PWM} \qquad \textit{Eqn. 6}$$

where:

- T_{ADC} — Time of the actual ADC sample
- $BEMF_T$ — Measured Back-EMF voltage in current commutation period
- $BEMF_{T-1}$ — Measured Back-EMF voltage in the previous commutation period
- DCBUS — Actual DC bus voltage
- T_{PWM} — Period of the PWM, equal to the time between the $BEMF_T$ and $BEMF_{T-1}$ ADC samples
- T_{ZC} — Recalculated time of the zero-cross event

3.2.2.2 Falling Back-EMF interpolation

Figure 10 illustrates the shape of the falling Back-EMF voltage.

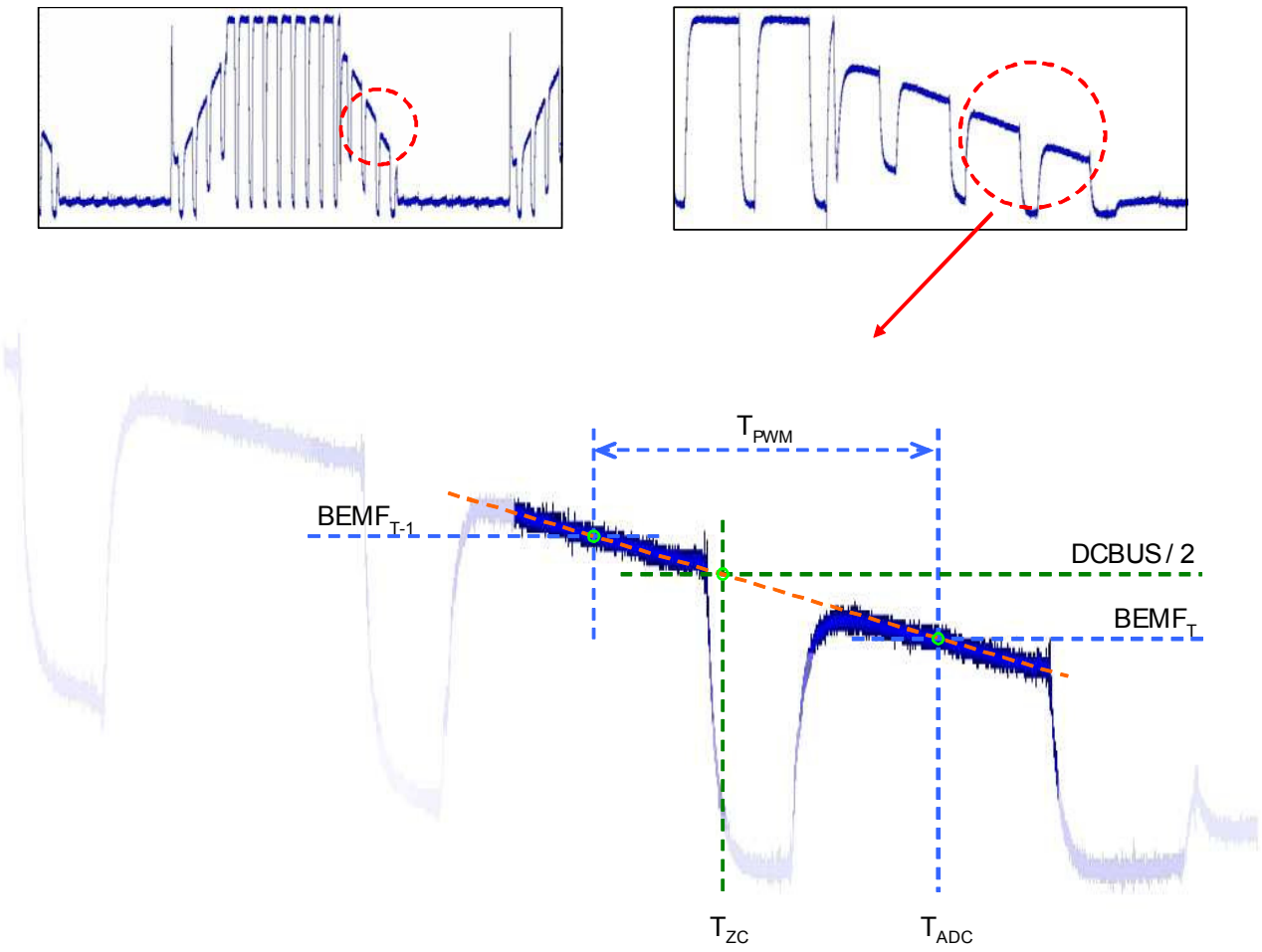


Figure 10. Falling Back-EMF voltage

The zero-cross event can be calculated using the falling Back-EMF interpolation used in Equation 7:

$$T_{ZC} = T_{ADC} - \frac{DCBUS/2 - BEMF_T}{BEMF_{T-1} - BEMF_T} \times T_{PWM} \qquad \textit{Eqn. 7}$$

The example code in Example 7 can be used for the rising and falling interpolation of two ADC samples to calculate a precise value of the zero-cross event time. The interpolation is performed in the ADC Sequence Complete interrupt within the zero-cross detection function.

Example 7. Rising and falling Back-EMF interpolation

```
timeOldBackEmf = timeBackEmf;       /* Save time of the old Back-EMF sample */
timeBackEmf = TC0;                  /* Save time of the actual Back-EMF sample */

/* Calculate actual Back-EMF voltage */
bemfVoltage = ADCResults.BEMFVoltage - (u_dc_bus_filt >> 1);

/* Mul_F16(): Signed 16-bit fractional multiplication without saturation (F16*F16 = F16) */
/* Div_F16(): Signed 16-bit fractional division without saturation (F16/F16 = F16) */

/* Rising Back-EMF zero-cross detection with interpolation */
int16_t delta;
if(bemfVoltage >= 0)
{
      delta = bemfVoltage - bemfVoltageOld;
      if(delta > bemfVoltage)
      {
          timeBackEmf -= Mul_F16(Div_F16(bemfVoltage, delta), (timeBackEmf - timeOldBackEmf));
      }
      else
      {
          timeBackEmf -= ((timeBackEmf - timeOldBackEmf) >> 1);
      }

      lastTimeZC = timeZC;
      timeZC = timeBackEmf;
}

/* Falling Back-EMF zero-cross detection with interpolation */
int16_t delta;
if(bemfVoltage <= 0)
{
      delta = bemfVoltage - bemfVoltageOld;
      if(delta < bemfVoltage)
      {
          timeBackEmf -= Mul_F16(Div_F16(bemfVoltage, delta), (timeBackEmf - timeOldBackEmf));
      }
      else
      {
          timeBackEmf -= ((timeBackEmf - timeOldBackEmf) >> 1);
      }

      lastTimeZC = timeZC;
      timeZC = timeBackEmf;
}
```

3.3 Commutation

Commutation is an event in which the motor phases are re-switched to GND, DC bus voltage, or disconnected. This depends on the actual rotor position and direction of rotation. See Figure 11.

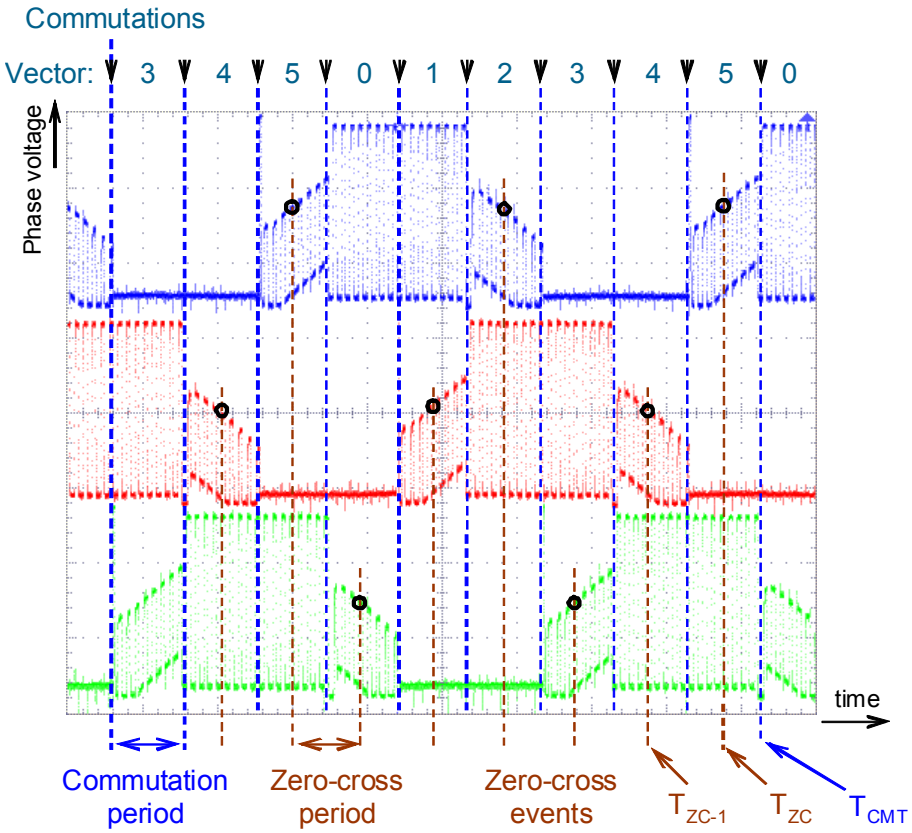


Figure 11. Commutation and zero-cross events

During the normal run of the application, the time of the next commutation is calculated from the zero-cross event time stamps using Equation 8.

$$T_{CMT} = T_{ZC} + \text{AdvanceAngle} \times (T_{ZC} - T_{ZC-1}) \qquad \textit{Eqn. 8}$$

where:

- T_{ZC} — Time of the actual zero-cross event (captured in the TIM TC0 register)
- T_{ZC-1} — Time of the previous zero-cross event
- AdvanceAngle — Constant in the range 0.3 to 0.5 (depending on the motor parameters)
- T_{CMT} — Calculated time of the next commutation

The constant parameter AdvanceAngle reflects the motor parameters, the application response delay, and motor load status. In an ideal case, the value should rise up to 0.5. In a real application, the value is usually in the range of 0.3 to 0.45 and is almost constant. The exact value is tuned during the application development.

Commutation events are interrupt driven. TIM channel 1 operates in the output capture mode with no output compare action on the timer output pin. TIM TC1 register is used to generate a commutation interrupt at the desired time. The value is calculated by Equation 8 and written into the compare register once the zero-cross event is detected in the ADC Sequence Complete Interrupt service routine.

Example 8. Commutation event time calculation function

```
actualPeriodZC = (actualPeriodZC + (timeZC - lastTimeZC)) >> 1;

/* Mul_F16(): Signed 16-bit fractional multiplication without saturation (F16*F16 = F16) */
NextCmtPeriod = Mul_F16(actualPeriodZC, advanceAngle);
TC1 = timeZC + NextCmtPeriod;
```

3.4 Application flow

This section describes the software design of the BLDC sensorless algorithm. Figure 12 shows the system block diagram.

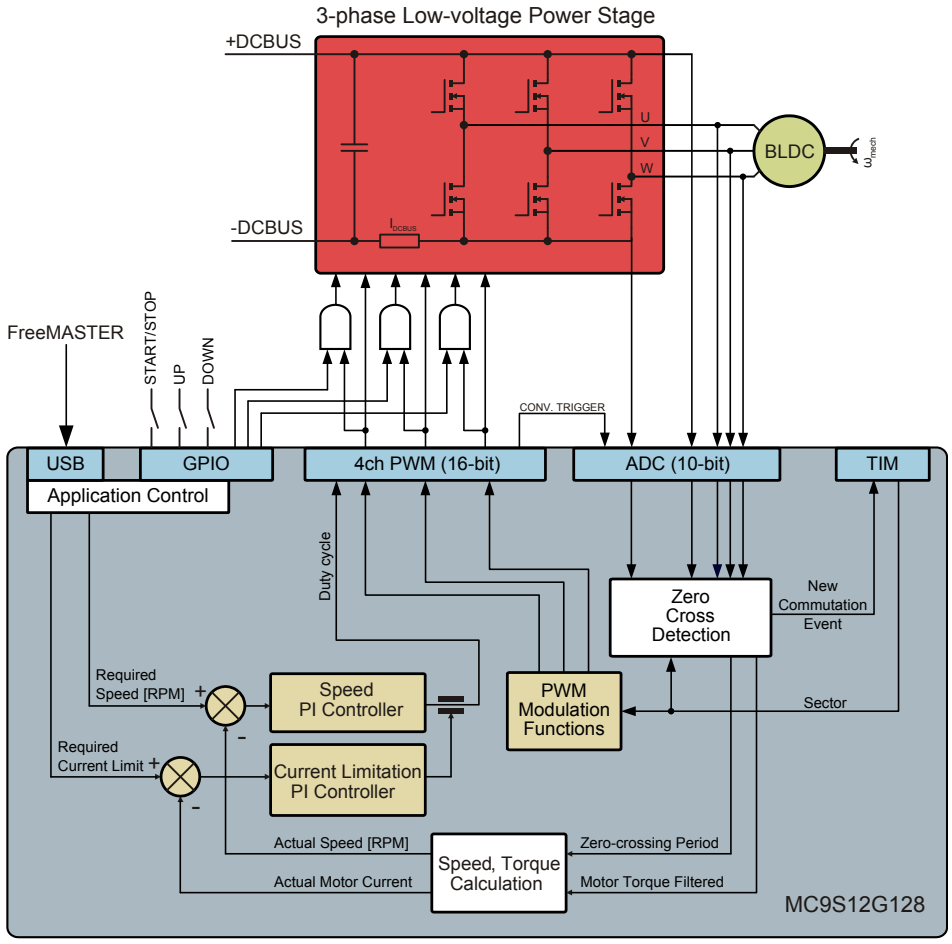


Figure 12. System block diagram

The application is optimized to use the MC9S12G128 on-chip peripherals to achieve a small S12 core load. The ADC, PWM, and TIM modules are set up to achieve deterministic sampling of analog quantities and a precise commutation of the stator field.

The application state machine function, including the speed and current proportional integral (PI) controller functions, is run in the main endless loop. The application is also driven by two interrupt sources:

- ADC Sequence Complete interrupt (every 50 µs)
- TIM channel 1 output compare interrupt (commutation interrupt)

All application tasks apart from the commutation and ADC interrupt are executed in the order shown in the application flow charts in Figure 13 and Figure 14.

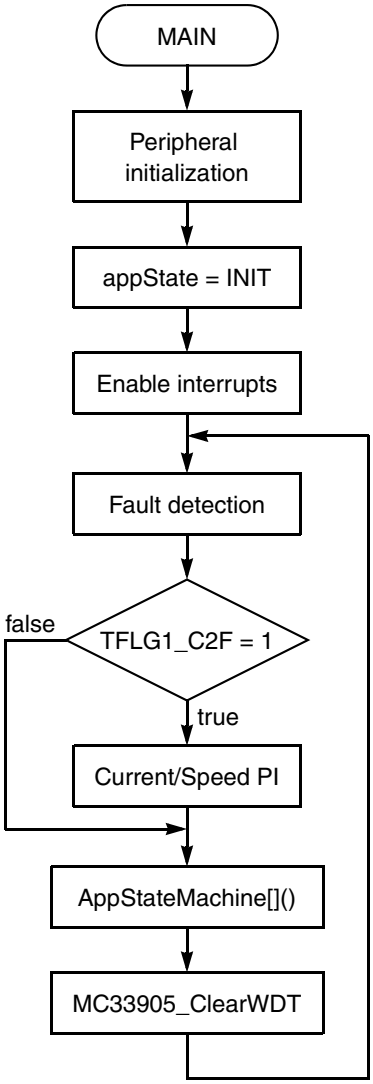


Figure 13. Main task flow diagram

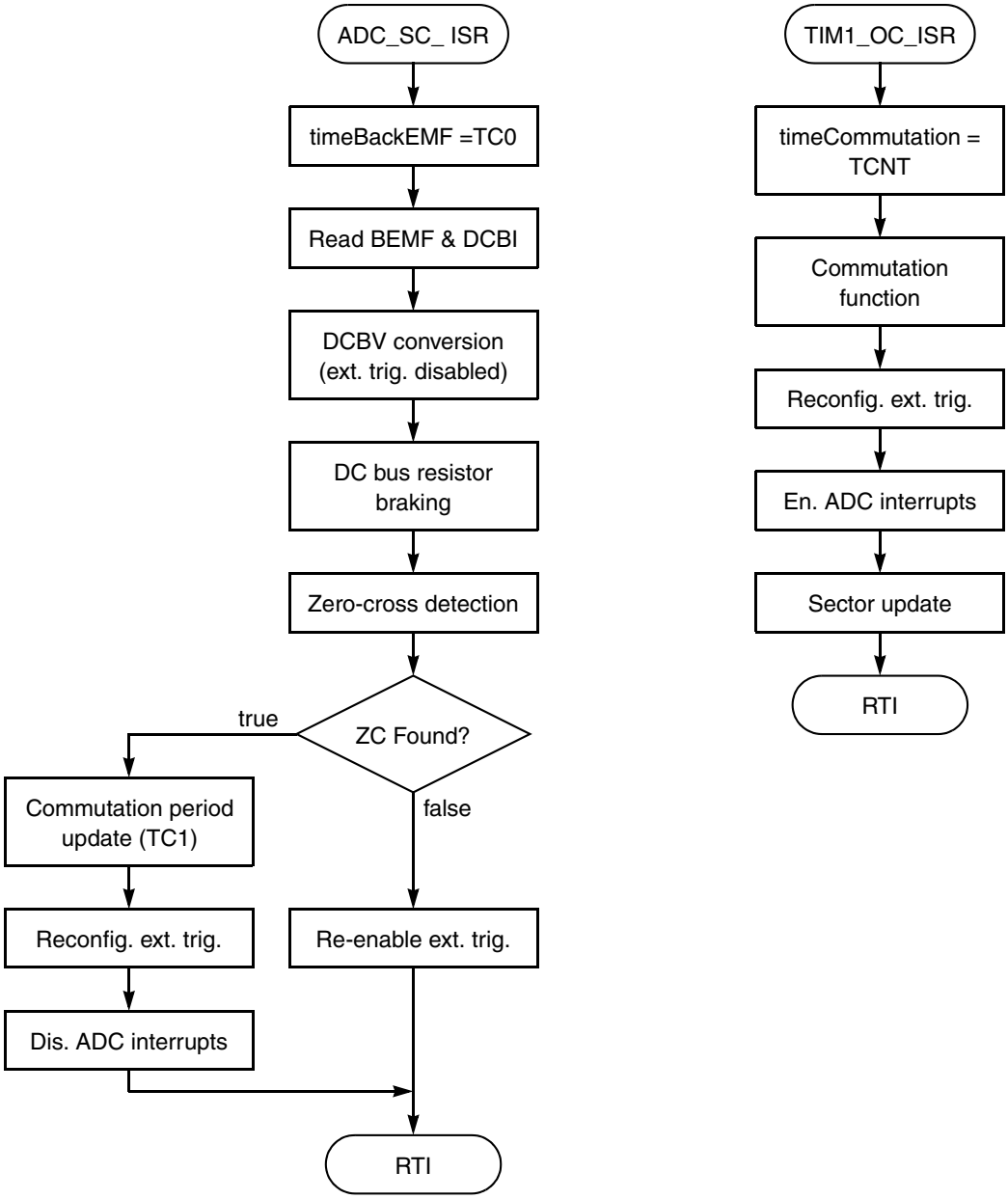


Figure 14. Interrupt service routine flow charts

The commutation interrupt (TIM channel 1 output compare interrupt) has the highest priority in the application to ensure precise commutation timing. Since a commutation event is not synchronous to the ADC external trigger, the commutation interrupt may be delayed by the ADC Sequence Complete interrupt service routine execution (interrupt nesting not enabled). To avoid this situation, ADC interrupts are disabled once the zero-cross event is detected. The external trigger is reconfigured to trigger the DC bus current and voltage conversion sequence (the ATDSTAT0[SCF] flag has to be polled in the main endless loop to read the DC bus current and voltage conversion results). The ADC interrupts are re-enabled again in the commutation interrupt. The external trigger is reconfigured to trigger again the Back-EMF and DC bus current conversion sequence.

3.4.1 Speed evaluation and control

The speed is calculated every 2 ms in the main loop (the TFLG[C2F] flag polled in the main endless loop, TIM channel periodic output compare set to 1 ms). The zero-cross detection function provides the actual commutation period duration for each commutation event. The TIM counter clock is set to 1 MHz. The period of one electrical revolution in seconds is given by Equation 9.

$$T_{elrev} = \frac{T \times N}{f_{TIM}} \qquad \textit{Eqn. 9}$$

where:

- T — Commutation period in TIM counter ticks
- N — Number of commutations of one electrical revolution
- f_{TIM} — TIM counter clock frequency
- T_{elrev} — Calculated period of one electrical revolution in seconds

To calculate the period of one mechanical revolution in seconds, the electrical revolution period needs to be multiplied by the number of motor pole-pairs:

$$T_{mechrev} = T_{elrev} \times p = \frac{T \times N \times p}{f_{TIM}} \qquad \textit{Eqn. 10}$$

where:

- p — Number of pole-pairs

Mechanical speed in revolutions per minute can be calculated using Equation 11.

$$RPM = \frac{60}{T_{mechrev}} = \frac{60 \times f_{TIM}}{T \times N \times p} \qquad \textit{Eqn. 11}$$

The mechanical speed constant Equation 12 can be obtained by grouping the constant parameters from Equation 11.

$$c = \frac{60 \times f_{TIM}}{N \times p} = \frac{60 \times 1 \times 10^6}{6 \times 4} = 2.5 \times 10^6 \qquad \textit{Eqn. 12}$$

Finally, the speed of the Linix 45ZWN24-90 motor with 4 pole-pairs can be calculated using Equation 13.

$$RPM = \frac{c}{T} = \frac{2.5 \times 10^6}{T} \qquad \textit{Eqn. 13}$$

The motor speed PI controller is called every 2 ms in the main endless loop. The speed PI controller input is the difference between the required and actual speeds. The PI controller output is the PWM duty cycle limited in the range of 0.1 to 0.9.

3.4.2 Current limitation

The motor current limitation PI controller is called every 1 ms in the main endless loop. The current limitation PI controller input is the difference between the maximal DC bus current (current limit) and the average DC bus current sampled during six consecutive zero-cross events. The PI controller output is the PWM duty cycle limited in the range of 0.1 to 0.9.

If the current limiting PI controller output is lower than the speed PI output, the PWM duty cycle is controlled (limited) by the current limiting PI controller. See Figure 15 for the speed control and current limitation block diagram.

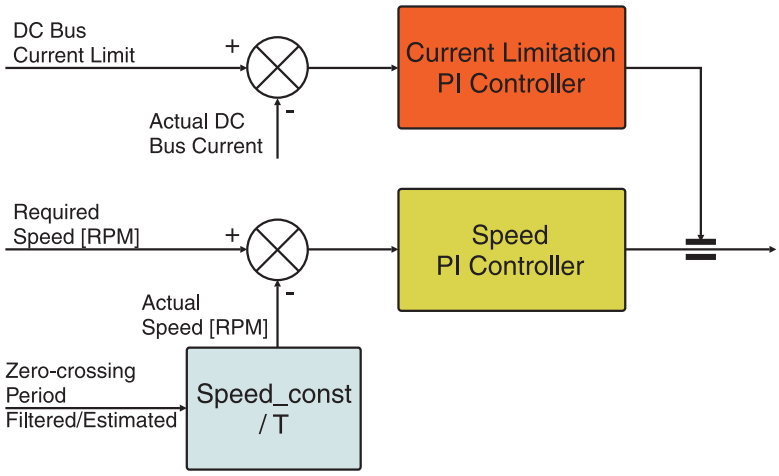


Figure 15. Speed control with current limitation

3.4.3 State machine

The application state machine is implemented using a one-dimensional array of pointers to state functions, called AppStateMachine[]. The index of the array specifies the pointer to the related application state function. The application state machine consists of the following six states selected using the index variable *appState* value. The application states are listed in the Table 7. Possible state transitions are shown in Figure 16.

Table 7. Application state machine

AppState	Application state	Description
0	INIT	The INIT state provides the initial configuration of the PWM duty cycle, commutation period, ADC external triggering, and DC bus current offset calibration. The state machine then transitions to the STOP state.
1	ALIGNMENT	In the ALIGNMENT state, the alignment vector (Table 6) is applied to the stator to set the rotor to the defined position as described in Section 3.1, “PWM generation”. The duration of the alignment state and the duty cycle applied during the state are defined by the ALIGNMENT_TIME and ALIGNMENT_DUTY_CYCLE macro values accessible in the S12G128_appconfig.h header file (see AN4558SW). The state machine then transitions to the START state.

Table 7. Application state machine (continued)

AppState	Application state	Description
2	START	In the START state, the motor commutation is controlled in an open-loop without any rotor position feedback. The initial commutation period is controlled by the START_FIRST_PERIOD macro value. Motor acceleration (commutation period multiplier < 1) is set by the START_CMT_ACCELERATION macro value. The number of commutations in the START state and the duty cycle incremental step is defined by START_CMT_CNT and START_DUTY_CYCLE_INC. All macro values are accessible in the S12G128_appconfig.h header file (see AN4558SW). The aim of the START state is to achieve an RPM where the zero-cross event can be reliably detected (Back-EMF high enough). Once the defined number of commutations is performed, the state machine transitions to the RUN state.
3	RUN	In the RUN state, the BLDC motor is controlled in the closed-loop by the sensorless algorithm (Back-EMF sensing with zero-cross detection). Speed control and current limitation are performed as described in Section 3.4.1, “Speed evaluation and control” and Section 3.4.2, “Current limitation”. The transition to the INIT state is done by setting the *appSwitchState* variable to 0.
4	STOP	In the STOP state, the motor is stopped and prepared to start running. Transition to the ALIGNMENT state is performed once the *appSwitchState* variable is set to 1.
5	FAULT	The fault detection function is executed in the main endless loop, detecting DC bus undervoltage, DC bus overvoltage, DC bus overcurrent, and MC33937A faults. When any of the faults are detected, the state machine automatically transitions to the FAULT state. The PWM outputs are set to the safe state. To exit the FAULT state, all fault sources must be removed and the *faultSwitchClear* variable has to be set to 1 to clear the fault latch. The state machine then automatically transitions to the INIT state.

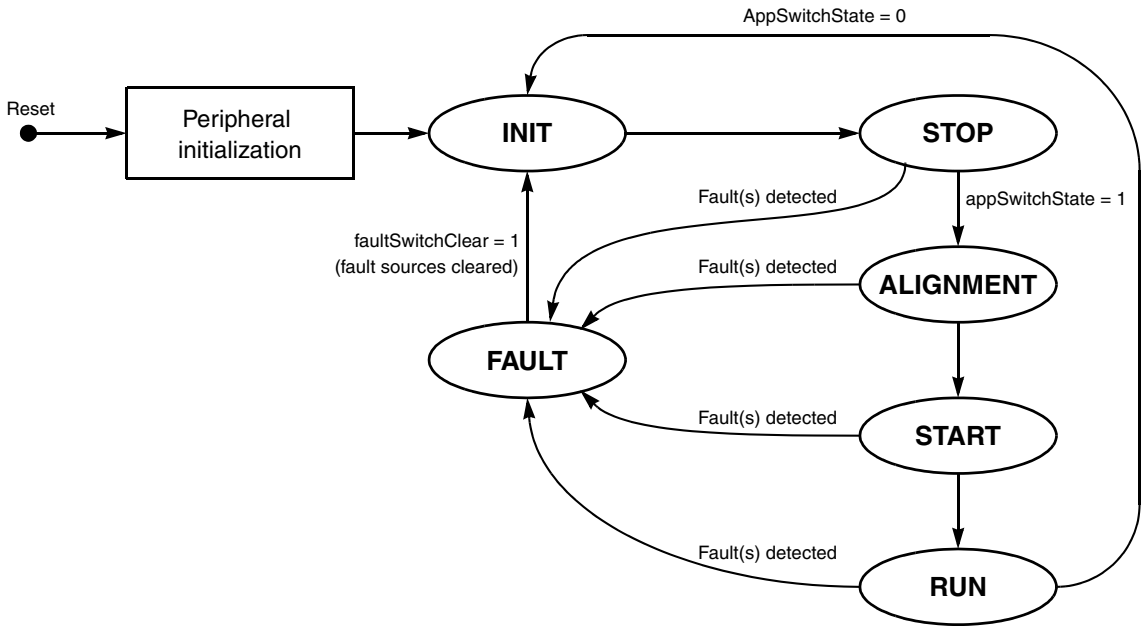


Figure 16. Sensorless motor control application state diagram

4 BLDC Hall sensor based motor control application using the MC9S12G128

This section provides an example of how to use the PWM, ADC, TIM, and PIM module in the BLDC Hall sensor based motor control application. This application example is compatible with the hardware of the 3-phase Sensorless BLDC Motor Control Development Kit with MC9S12G128 MCU.

4.1 Commutation table definition

In this sensor-based BLDC motor application, the Hall sensors incorporated in the motor are used as external position sensors to sense the actual rotor position. Hall sensors detect the rotor flux, so their actual state is not influenced by stator current. The Hall effect sensor outputs in the BLDC motor divide the electrical revolution into three equal sections of 120° electrical. In this so-called 120° configuration, the Hall states 111 and 000 never occur. Hall sensors are aligned phase-to-phase to the Back-EMF voltage.

The process of defining the commutation table has the following steps:

1. Defining the Hall sensor pattern for each commutation sector
2. Defining the commutation vector for each Hall sensor pattern

4.1.1 Defining the Hall sensor pattern

The Hall sensor pattern corresponding to the BLDC motor commutation sector needs to be defined. This can be done by supplying all the three phases with a combination of positive and negative voltage that causes the motor to move to the given sector and then observing the Hall sensor signals when the motor settles.

The following steps describe a simple method that can be used to define the Hall sensor pattern:

1. Mark all the motor phases as A, B, and C and the Hall sensor outputs as H1, H2, H3, in the desired order.
2. Set the current limit of the power supply to 20–30% of the nominal motor current.
3. Choose the direction of the motor rotation to clockwise/counterclockwise.
4. Connect one of the three motor phases to the positive terminal as shown in the first row of Table 8.
5. Connect the remaining two motor phases to the negative terminal.

Table 8 illustrates the required phases to be powered for the desired sector and the example of sensor output for the BLDC motor (Linix 45ZWN24-90) supplied with the 3-phase Sensorless BLDC Motor Control Development Kit with MC9S12G128 MCU. Corresponding sectors are shown in Figure 17.

Table 8. Linix 45ZWN24-90 Hall sensor pattern

Powered phase			Sector	Hall sensor output		
Phase A	Phase B	Phase C		Hall_2	Hall_1	Hall_0
$+V_{DBC}$	$-V_{DBC}$	$-V_{DBC}$	I	0	1	0
$+V_{DBC}$	$+V_{DBC}$	$-V_{DBC}$	II	1	1	0
$-V_{DBC}$	$+V_{DBC}$	$-V_{DBC}$	III	1	0	0
$-V_{DBC}$	$+V_{DBC}$	$+V_{DBC}$	IV	1	0	1
$-V_{DBC}$	$-V_{DBC}$	$+V_{DBC}$	V	0	0	1
$+V_{DBC}$	$-V_{DBC}$	$+V_{DBC}$	VI	0	1	1

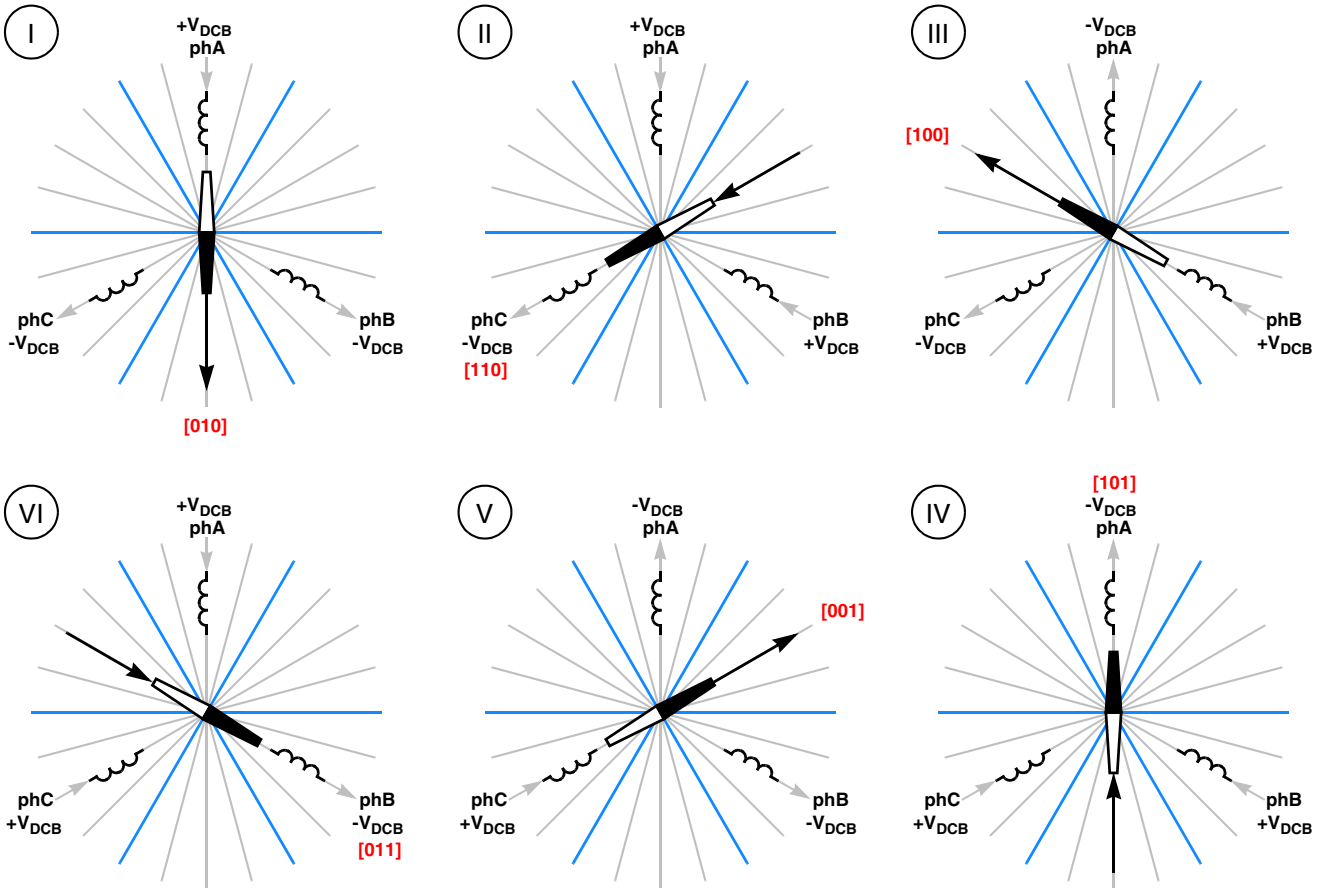


Figure 17. Commutation sector definition

4.1.2 Definition of the commutation vectors

When the Hall sensor patterns are defined, the corresponding commutation vectors can also be defined. For each of the sectors I–VI, the corresponding commutation vectors 0–5 are defined. Table 9 shows the vectors for the clockwise rotation direction that ensure the variation of real angle from 60° to 120°. See

Table 9 for all combinations. A graphical representation of the commutation vectors for the clockwise rotation direction with highlighted angle variation is shown in Figure 18.

Table 9. Linix 45ZWN24-90 clockwise direction commutation sequence

Commutation Vector			Vector	Hall sensor pattern definition			Decimal result
Phase A	Phase B	Phase C		Hall_2	Hall_1	Hall_0	
$+V_{DBC}$	$-V_{DBC}$	NC	0	0	0	1	1
$+V_{DBC}$	NC	$-V_{DBC}$	1	0	1	1	3
NC	$+V_{DBC}$	$-V_{DBC}$	2	0	1	0	2
$-V_{DBC}$	$+V_{DBC}$	NC	3	1	1	0	6
$-V_{DBC}$	NC	$+V_{DBC}$	4	1	0	0	4
NC	$-V_{DBC}$	$+V_{DBC}$	5	1	0	1	5

Table 10. Linix 45ZWN24-90 counterclockwise direction commutation sequence

Commutation Vector			Vector	Hall sensor pattern definition			Decimal result
Phase A	Phase B	Phase C		Hall_2	Hall_1	Hall_0	
$-V_{DBC}$	$+V_{DBC}$	NC	3	0	0	1	1
NC	$+V_{DBC}$	$-V_{DBC}$	2	1	0	1	5
$+V_{DBC}$	NC	$-V_{DBC}$	1	1	0	0	4
$+V_{DBC}$	$-V_{DBC}$	NC	0	1	1	0	6
NC	$-V_{DBC}$	$+V_{DBC}$	5	0	1	0	2
$-V_{DBC}$	NC	$+V_{DBC}$	4	0	1	1	3

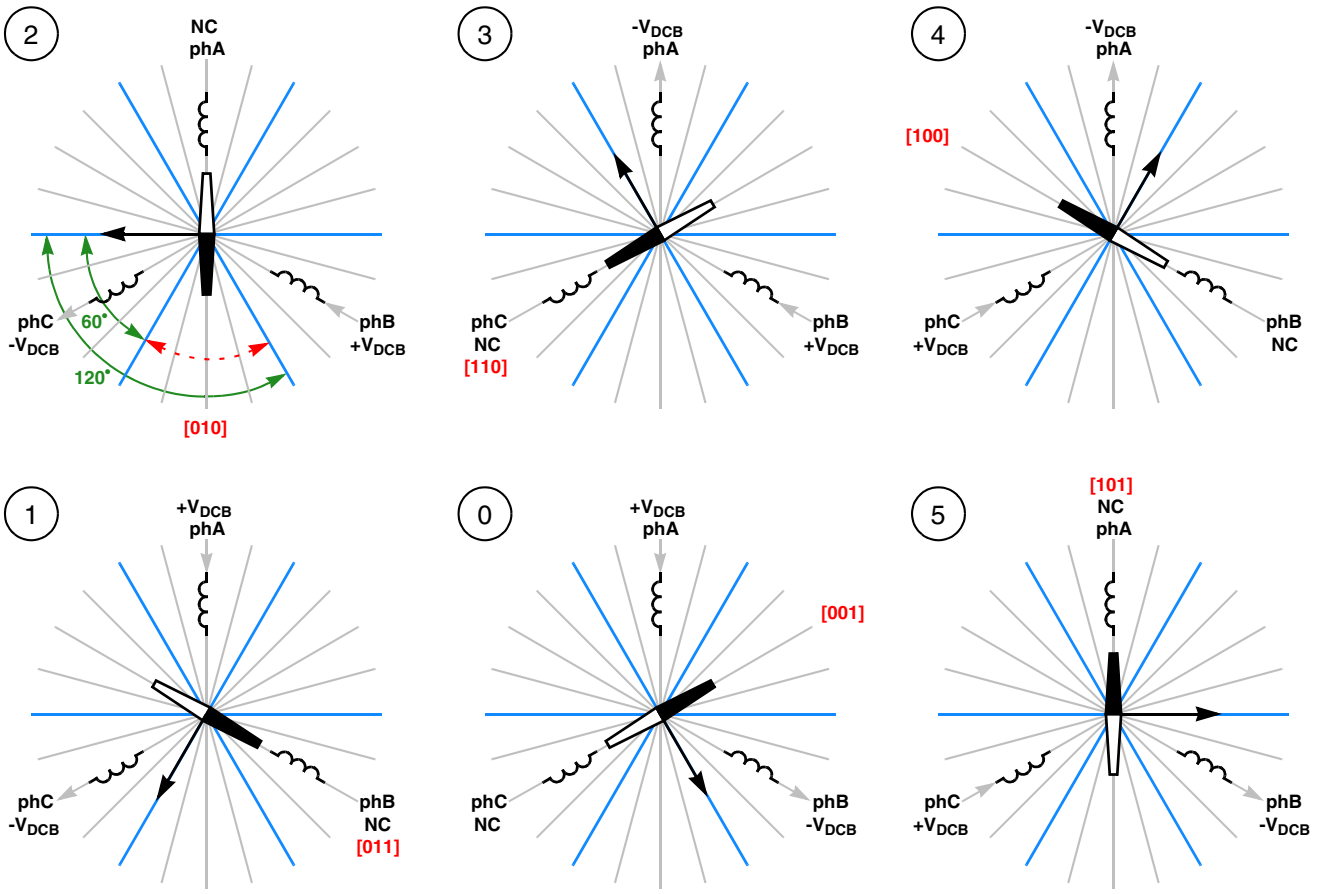


Figure 18. Commutation vector definition for clockwise rotation direction

Based on the rotation direction, each commutation vector is assigned to the Hall sensor pattern as shown in Table 9 and Table 10. The Hall sensor patterns 0 [000] and 7 [111] indicate a fault because such combinations are not available in normal operation. Such a state can be caused either by a disconnected Hall sensor interface, or by a malfunction. In these cases, the PWM outputs need to be switched to the off state in order to protect the drive.

Figure 19 shows the PWM waveforms with split gates signals, Hall sensor output signals, and phase voltages.

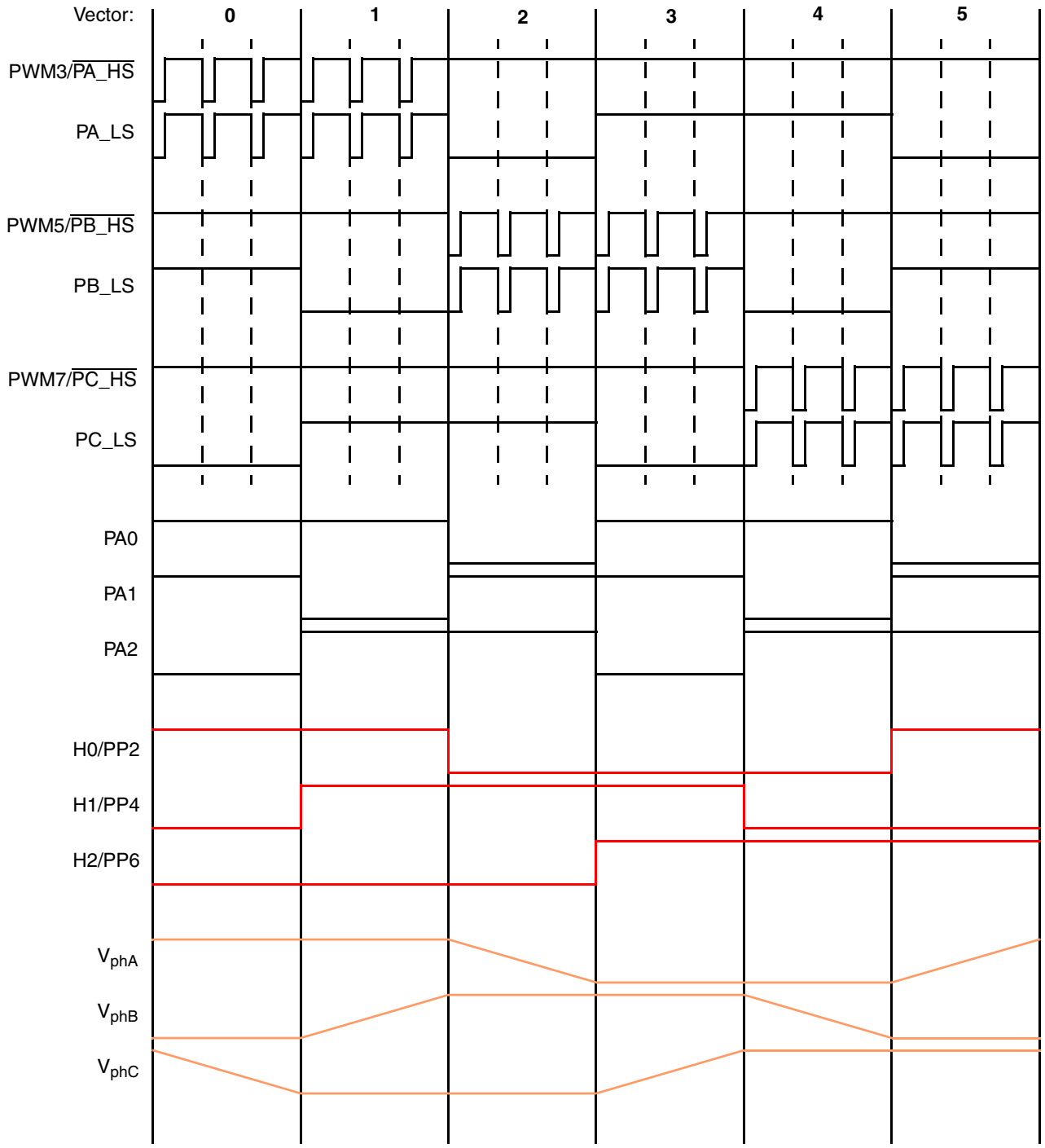


Figure 19. PWM generation waveforms with Hall sensor outputs (clockwise direction)

4.2 Commutation

The commutation process is interrupt driven. The Hall sensor outputs are fed to the inputs configured to detect active edges and invoke an external interrupt. This feature is handled by the PIM module (see Section 2.4, “Pin interrupts”).

Pins PP2, PP4, and PP6 are used to detect edges of the Hall sensor H0, H1, H2 output signals on the MC9S12G128 Controller Board. Since only one type of input active edge can be configured, the active edge polarity needs to be configured for each input before starting the motor and reconfigured after each Hall output transition (based on the actual Hall sensor pattern, as depicted in Figure 19).

Once an active edge is detected on one of the inputs, the interrupt is generated. The Hall sensor output needs to be read and the corresponding commutation vector needs to be applied based on the rotation direction (see Table 9 or Table 10).

NOTE

As the optimal solution, the Hall sensor patterns can be detected by the TIM channel input capture feature (three TIM channels configured in input capture mode, sensitive to both falling and rising edges). This solution eliminates the need for an active edge reconfiguration as in the external interrupt solution. The TIM solution is, however, not natively supported on the MC9S12G128 Controller Board (it requires a manual wiring between the Hall sensor interface and the MCU pin headers).

4.3 Application flow

The application flow of the main task is identical to the sensorless BLDC motor application (see Figure 13). The application state machine function, including the speed and current control loops, is executed in the main endless loop. The application is also driven by two interrupt sources:

- ADC Sequence Complete interrupt (every 50 µs)
- Port P interrupt (commutation interrupt)

The ADC external trigger is configured in a similar way as in the sensorless application. Only the DC bus voltage and DC bus current are converted in the conversion sequence triggered by the PWM1 rising edge. Conversion results are read in the ADC Sequence Complete interrupt.

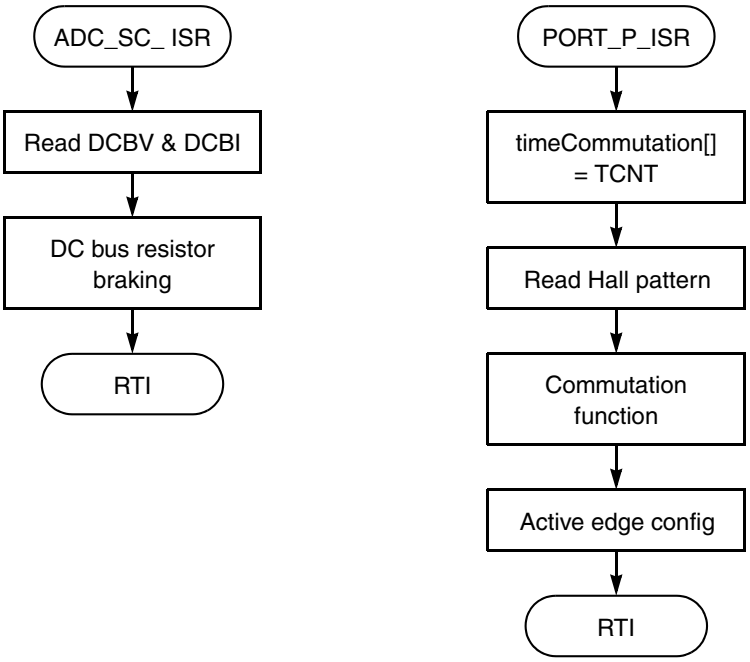


Figure 20. Interrupt service routine flow charts

4.3.1 Speed evaluation and control

The speed is calculated every 2 ms in the main loop (the TFLG[C0F] flag polled in the main endless loop, TIM channel periodic output compare set to 1 ms). The commutation function calculates the actual commutation period based on the free-running TIM counter value. The TIM counter clock is set to 1 MHz. The calculation of the actual speed and speed control are done according to the equations in Section 3.4.1, "Speed evaluation and control".

The motor speed PI controller is called every 2 ms in the main endless loop. The speed PI controller input is the difference between the required and actual speeds. The PI controller output is the PWM duty cycle limited in the range of 0.1 to 0.9.

4.3.2 Current limitation

The motor current limitation PI controller is called every 1 ms in the main endless loop. The current limitation PI controller input is the difference between the maximal DC bus current (current limit) and the average DC bus current sampled during the PWM cycle before each of the six consecutive commutations. The PI controller output is the PWM duty cycle limited in the range of 0.1 to 0.9.

If the current limiting PI controller output is lower than the speed PI output, the PWM duty cycle is controlled (limited) by the current limiting PI controller. See Figure 15 for the speed control and current limitation block diagram.

4.3.3 State machine

The application state machine is implemented as in the sensorless BLDC motor control application using a one-dimensional state machine function pointer array. The application state machine consists of the following four states, selected using the index variable *appState* value. The application states are listed in the Table 11. Possible state transitions are shown in Figure 21.

Table 11. Application state machine

AppState	Application state	Description
0	INIT	The INIT state provides the initial configuration of the PWM duty cycle, ADC external triggering, and DC bus current offset calibration. The state machine then transitions to the STOP state.
1	RUN	In the RUN state, the motor commutation is based on the Hall sensor output transitions starting with the initial duty cycle defined by the value of START_DUTY_CYCLE accessible in the S12G128_appconfig.h header file (see AN4558SW). Speed control and current limitation are performed as described in Section 4.3.1, “Speed evaluation and control” and Section 4.3.2, “Current limitation”. The transition to the INIT state is done by setting the *appSwitchState* variable to 0.
2	STOP	In the STOP state, the motor is stopped and prepared to start running. Transition to the RUN state is performed when the *appSwitchState* variable is set to 1.
3	FAULT	The fault detection function is executed in the main endless loop, detecting DC bus undervoltage, DC bus overvoltage, DC bus overcurrent, and MC33937A faults. When any of the faults are detected, the state machine automatically transitions to the FAULT state. The PWM outputs are set to the safe state. To exit the FAULT state, all fault sources must be removed and the *faultSwitchClear* variable has to be set to 1 to clear the fault latch. The state machine then automatically transitions to the INIT state.

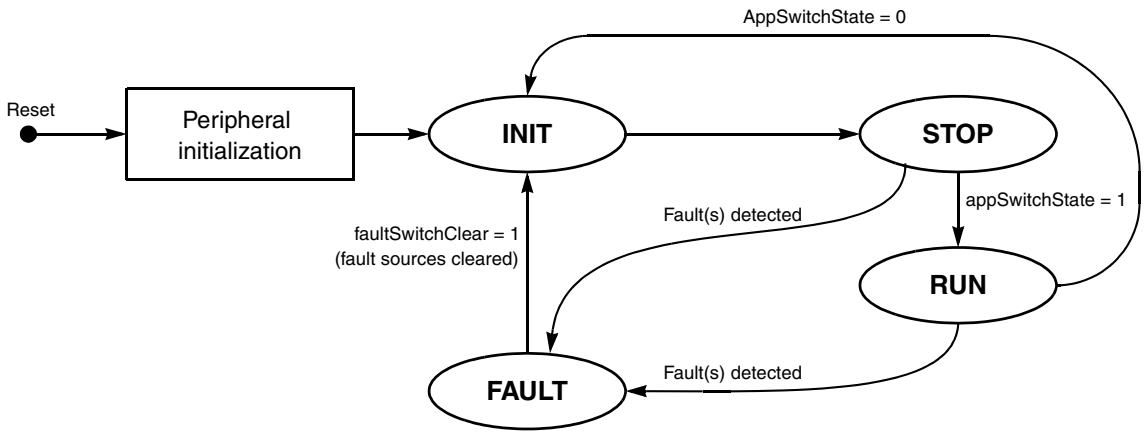


Figure 21. Hall sensor based motor control application state diagram

5 Conclusion

This application note describes the usage of the MC9S12G128 PWM, ADC, TIM, and PIM peripherals in the sensorless and Hall sensor based BLDC motor control applications using the 3-phase Sensorless BLDC Motor Control Development Kit with the MC9S12G128. Both application examples take advantage of the MC9S12G128 peripheral features, such as hardware-handled ADC external triggering to improve timing precision of the Back-EMF voltage sampling with a reduced CPU load. This set of peripherals makes the MC9S12G128 suitable for BLDC motor control applications.

6 References

Document number	Title	Location
MC9S12GRMV1	*MC9S12G Family Reference Manual*	www.freescale.com
MC9S12G128MCBUG	*MC9S12G128 Controller Board User Guide*	www.freescale.com/AutoMCDevKits
3PHLVPSUG	*3-phase Low Voltage Power Stage User Guide*	
—	*3-phase Sensorless BLDC Motor Control Development Kit with MC9S12G128 MCU*	

7 Acronyms

ADC	Analog-to-Digital Converter
Back-EMF	Back Electromotive Force
BDM	Background Debug Module
BLDC	Brushless DC
CAN	Controller Area Network
DC	Direct Current
FET	Field Effect Transistor
ISR	Interrupt Service Routine
LIN	Local Interconnect Network
MCU	Microcontroller Unit
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
PI	Proportional Integral
PIM	Port Integration Module
RPM	Revolutions per Minute
RTI	Return from Interrupt
PWM	Pulse Width Modulation
SAE	Society of Automotive Engineers
SPI	Serial Peripheral Interface
TIM	Timer Module
USB	Universal Serial Bus

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4558
Rev. 0
08/2012