THIS SPEC IS OBSOLETE

Spec No: 002-04881

Spec Title: AN204881 - FM3 FAMILY, SENSORLESS BLDC MOTOR

Replaced by: None

AN204881

FM3 Family, Sensorless BLDC Motor

This Application-note describes the usage of the MFT to generate the commutation-signals for a BLDC Motor. These signals are generated with the help of the detection of the BEMF voltage of the motor.

Contents

1 Introduction

This Application-note describes the usage of the MFT to generate the commutation-signals for a BLDC Motor. These signals are generated with the help of the detection of the BEMF voltage of the motor.

Sensor less commutation of the motor is used for cost-sensitive applications like fans or pumps.

This application-note is based on the Fujitsu Starter kits:

- SK-FM3-100PMC
- SK-Power-3P-LV2-MC
- ADA-FM3-100-PMC-MC

The used motor comes from the company DUNKERMOTOREN (BG42X15)

The circuit-diagrams for these evaluation-boards are included the user-manuals of the tools. This application-note shows the principle function of driving a sensor less BLDC motor. In the real application the error-handler has to be included in the application.

2 Sensorless acquisition of BLDC motors

To understand the principle of sensor less acquisition of a BLDC motor drive you have to look about the commutation, done with the signals of hal-sensors

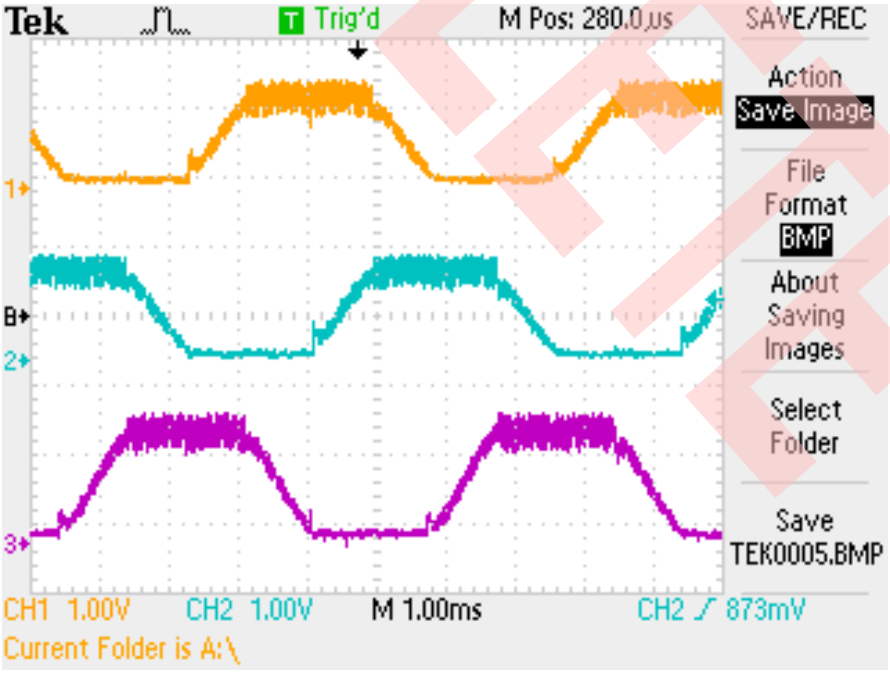
The hal-sensors detect the position of the rotor, which affects the commutation in the right time. This schema is handled in a commutation-table, shown in Table 1.

Table 1. Commutation-table with hall Sensors

Angle	0	60	120	180	240	300
Hal 1	H	H	L	L	L	H
Hal 2	L	L	L	H	H	H
Hal 3	L	H	H	H	L	L
Sector	1	2	3	4	5	6
Phase 1	-	O	+	+	O	-
Phase 2	+	+	O	-	-	O
Phase 3	O	-	-	O	+	+

The commutation is done every 60° of electrical rotating. By using the sensorless acquisition, the information from the hal-sensors is not accessible. The figure shows the voltages of the three phases with the correct commutation, done with the use of hal-sensors. When the motor rotates, the voltages of the phases have a nearly trapezoidal shape.

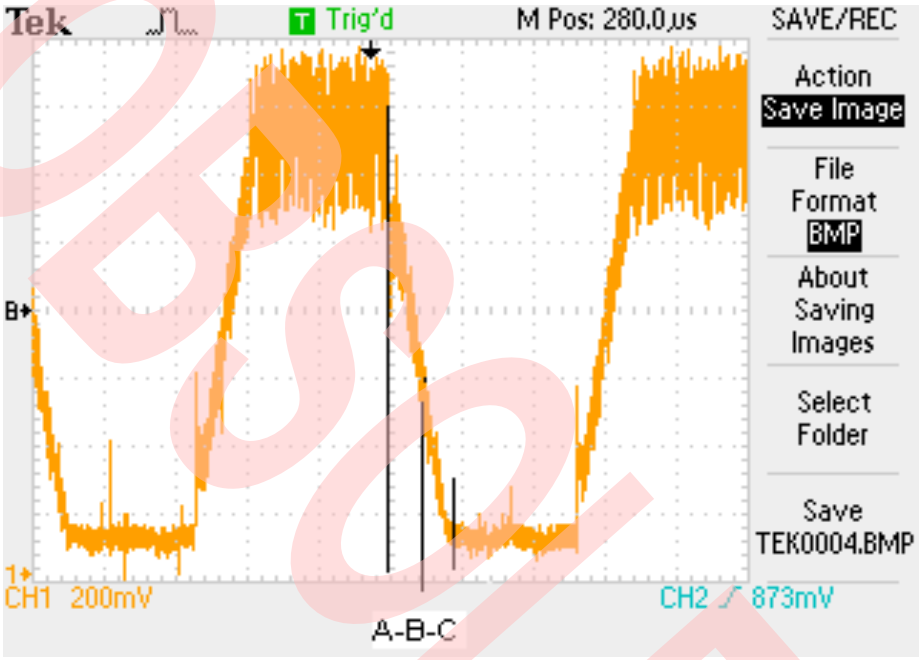
Figure 1. Commutation with hal-sensors



For a sensorless commutation the microcontroller has to measure the phase-voltages and the bus-voltage of the six phase bridge. Every phase has 3 different switching-states. It is possible to connect to the high or low side or to connect it nowhere (tristate). For the detection of the BEMF, the voltage of the phase which is not connected to the high or the low side has to be measured. This phase is marked with the O in the commutation-table. (Table 1)

Looking a little bit closer to this signal, especially the part between the high and the low voltage value, the commutation is periodical visible.

Figure 2. Closer view of one commutation cycle



The marker “A” shows the end of the high sector which is modulated with a PWM. Because of the R/C Filers in the ADC input, there are only the higher ripples of the PWM visible. The C-marker shows the low-side and the beginning of the low-sector.

To calculate the right time for the commutation, the microcontroller has to start a timer at the end of the high phase, represented from marker A. This timer is realized with the Dual-Timer of the FM3. In parallel to the down count of this timer, the voltage of the corresponding phase has to be measured periodically and compared with the voltage with is called virtual zero crossing point. This point is described with the half of the bus-voltage divided with the PWM duty cycle. This point is represented from marker B.

The timer which has been started on marker A is stopped when there is no difference between the value of the virtual zero crossing point and the measured voltage (marker B). A second timer is loaded with the counted value of the first timer and started after that. After the second timer has been expired, the marker C is reached. At this time the microcontroller must start the new commutation phase.

The measurement of the BEMF and the detection of the virtual zero crossing replaces the signals from the hal-sensors.

This method has one drawback in the start phase of the motor. At this time there is no measurement of BEMF possible. To start the motor a rotary field has to be used. This field is generated without any feedback to the motor. The frequency of this field has to be increased until it reaches the maximum needed rotating speed. After this the program has to switch to the above described behavior

3 The Software for sensorless acquisition

The software itself consists of five modules with are described in the following text. (main.c, ISR.c, ADC.c, DualTimer.c, MFT.c)

3.1 The main Function

```
int main(void)
{
    initIrqLevels();                // Setup Level of ISR
    initEXT_INT();                  // Setup external interrupt (Switch)
    initEnable();                   // Enable external interrupt
    initDualTimer();                // Init Timer for BEMF measurement
    MFT_SETUP();                    // Setup Multifunction timer
    initADC1();                     // Init AD Converter

    FM3_MFT1_FRT->TCSA0 &= 0xffbf;  // start FRT0
    Direction=FORWARD;
// Startup-routine for generating the BEMF.
    PosSwitch (6);                  // Switch to sector 6
    for (StartupDelay = 1; StartupDelay < 3000 ; StartupDelay++)
    {}
    PosSwitch (0);
    for (StartupDelay = 1; StartupDelay < 300 ; StartupDelay++)
    {}
// Speed-up ramp to accelerate the motor to the maximum speed
    for (Start=1; Start < 58; Start++)
    {
        for (StartupPOS=1; StartupPOS < 6; StartupPOS++)
        {
         for (StartupDelay=1;StartupDelay<(4500-Start*20);StartupDelay++)
         {}
        PosSwitch (StartupPOS);     // Switch between the Sectors
        }
    }
    PosSwitch (0);          // Switch the output to tristate makes the
                            // tansition between "blind" commutation and
                            // BEMF detection commutation softer.
    for (StartupDelay = 1; StartupDelay < 100 ; StartupDelay++)
    {}
    Startup = 0;            // Startup done, ISR handles the commutation
    while(1);
}
```

In the first sections of the main-function all peripheral parts were initialized. This part is described in the sections of these modules.

After this initialization, the free-run timer of the MFT is started. The parts which are needed for the sensorless acquisition are also running in the background but without function. All of them are interrupt-driven.

For alignment of the motor-rotor the start-up-routine switches the output of the PWM to the sector 6. After a short while (for loop) the output are switched to off. This makes the start-up smoother.

Now the commutation begins without sensing the position of the motor. To make this possible, two for-loops are needed. The first decrease the basis-time between the commutation to accelerate the motor, the second is for the basis-time it selves. Because of the different torque of inertia of the different motors, these values have to be adjusted for every motor.

After the start-up, the start-up value is set to zero. After that the commutation is done automatically in the interrupt service-routine.

3.2 The ISR Function

```
void initIrqLevels(void)
{
    NVIC_SetPriority(EXINT0_7_IRQn, 0x7);    // External I-0 ISR
    NVIC_SetPriority(WFG_IRQn, 0x4);         // DTIF (Emergency Stop)
ISR
    NVIC_SetPriority(ADC0_IRQn, 0x5);        // ADC ISR
    NVIC_SetPriority(DTIM_QDU_IRQn, 0x3);    // DualTimer IRQ
}
void initEnable(void)
{
    NVIC_EnableIRQ(EXINT0_7_IRQn);           // Enable external
Interrupt
    NVIC_EnableIRQ(WFG_IRQn);                // Enable waveform
generator
    NVIC_EnableIRQ(ADC0_IRQn);               // Enable AD converter
    NVIC_EnableIRQ(DTIM_QDU_IRQn);           // Enable DualTimer
}
void initEXT_INT(void)
{
    bFM3_GPIO_DDR5_P0 = 0;                   // Set port as input
    bFM3_GPIO_DDR5_P1 = 0;
    bFM3_GPIO_PFR5_P0 = 0x1;                 // Set Port function T 1
    bFM3_GPIO_PFR5_P1 = 0x1;                 // Set Port function T 2

    FM3_GPIO->EPFR06 |= 0x0003e800;          // Port as interrupt
input
    FM3_EXTI->ENIR = 0;

    bFM3_EXTI_ELVR_LA0 = 1;
    bFM3_EXTI_ELVR_LB0 = 1;                  // T 1, falling edge

    bFM3_EXTI_ELVR_LA1 = 1;
    bFM3_EXTI_ELVR_LB1 = 1;                  // T 2, falling edge

    FM3_EXTI->EICL=0;                        // Reset Interrupt cause
    FM3_EXTI->ENIR=0x00003;
}
```

The initIrqLevels (void) function initializes the priorities of the used interrupts; the init Enable (void) function enables the interrupt globally. The init EXT_INT (void) function setup the input ports for the external interrupt. These Ports are represented from two switches on the SK-FM3-100PMC-Board from Cypress.

```
void INT0_7_Handler(void)
{
    if(bFM3_INTREQ_IRQ04MON_EXTINT1==1)
    {
      //Button 1 pressed
        if(FM3_MFT1_OCU->OCCP0 < 0x170)
        {
            FM3_MFT1_OCU->OCCP0 = FM3_MFT1_OCU->OCCP0 +5;
            FM3_MFT1_OCU->OCCP2 = FM3_MFT1_OCU->OCCP2 +5;
            FM3_MFT1_OCU->OCCP4 = FM3_MFT1_OCU->OCCP4 +5;
        }
    }
    if(bFM3_INTREQ_IRQ04MON_EXTINT0==1)
    {
        if(FM3_MFT1_OCU->OCCP0 > 0x60)
        {
            FM3_MFT1_OCU->OCCP0 = FM3_MFT1_OCU->OCCP0 -5;
            FM3_MFT1_OCU->OCCP2 = FM3_MFT1_OCU->OCCP2 -5;
            FM3_MFT1_OCU->OCCP4 = FM3_MFT1_OCU->OCCP4 -5;
        }
    }

    FM3_EXTI->EICL=0;
}
```

The INT0_7_Handler is the interrupt service-routine for the external interrupts. The FM3 architecture has only one interrupt-line for the external interrupts. Depending on the value of the IRQ04MON Register the program can detect witch interrupt is accessed and decide to call the first or the second if-statement. The function de- or increase the value of the PWM which corresponds to the speed of the motor.

3.4 The ADC Function

```
void initADC1(void)
{
    FM3_GPIO->ADE = 0x00f0;         // AD input enable
    FM3_ADC0->ADCR = 0x08;          // Interrupt enable
    FM3_ADC0->ADSR = 0x40;          // right aligned
    FM3_ADC0->SCCR = 0x12;          // Scan conversion timer start
enable
    FM3_ADC0->SFNS = 0x03;          // Interrupt in the 4th FIFO stage
    FM3_ADC0->SCIS0 = 0xf0;         // Voltage, Phase 1-3, Bus Voltage
    FM3_ADC0->ADST0 = 0x0f;         // AD Sampling Time
    FM3_ADC0->ADCT =  0x02;         // AD Comparison Time
    FM3_ADC0->ADSS0 = 0xf0;         // Sampling Time Select AN07-AN00
    FM3_ADC0->SCTSL = 0x01;         // Starts conversion with the mft
    FM3_ADC0->ADCEN = 0x01;         // Enable ADC

    while (3 != FM3_ADC0->ADCEN); // wait until ADC operation is
enabled
}
```

The function initADC1 initialize the analogue digital converter (ADC). The ADC converts the voltage of the bus and the BEMF-voltage of the three phases. After a complete conversion of the four channels an interrupt is generated. Because of different impedances of the input, the AD sampling and comparison-time has to be adjusted to the customer's application. The AD conversion is triggered from the Free-Run-Timer of the Multifunction-Timer.

```
void ADC0_IRQHandler(void)
{
    Phase_Voltage = FM3_ADC0->SCFDH;            // Voltage of the phase
    BEMF_VoltageR = FM3_ADC0->SCFDH;
    BEMF_VoltageS = FM3_ADC0->SCFDH;
    BEMF_VoltageT = FM3_ADC0->SCFDH;
    bFM3_ADC0_SCCR_SFCLR = 1;                   // Clears FIFO

    //Include the PWM calculation
    limit = (float)(0x1fc / ((float)(0x1fc - (FM3_MFT1_OCU->OCCP0))));
    // Calculate the virtual GND Voltage, depending on the duty cycle
    Virtual_GND_Voltage   = (int)((float)Phase_Voltage / (2*limit));
    // Delay to filter the noise of commutation
    if(FM3_DTIM->TIMER1VALUE < NoiseFilter)
    {
        If(Phase_Voltage>0x150)      // To filter the stop of the motor
        {
            if (BEMFchannel==1)
            {
                if (BEMF_VoltageR > (Virtual_GND_Voltage - deltaCompare))
                if (BEMF_VoltageR < (Virtual_GND_Voltage + deltaCompare))
                {
                    bFM3_DTIM_TIMER1CONTROL_TIMEREN = 0; // Stop DT
                    bFM3_DTIM_TIMER2CONTROL_TIMEREN = 0; // Start the DT
                    // Copy Value of the first Timer to the second Timer
                  FM3_DTIM->TIMER2LOAD =((0xffffffff-FM3_DTIM-
                        >TIMER1VALUE)/5);
                  FM3_DTIM->TIMER1LOAD=0xffffffff;
                  bFM3_DTIM_TIMER2CONTROL_TIMEREN=1;      // Start the DT
                }
            }
            if (BEMFchannel==2)……
            if (BEMFchannel==3)…..

    (FM3_ADC0->ADCR) &= ~0x80;       // clear IRQ flag
    NVIC_ClearPendingIRQ(ADC0_IRQn);
}
}
```

1. The fist lines shows, that the converted values of the AD are copied from the AD Fifo to the corresponding variables. After that the Fifo is cleared. The AD Values has to be compared to the Virtual Ground Voltage to detect the zero crossing of this signals. To calculate this Voltage the “limit” variable is needed to consider the PWM.
2. The next part of the function is only executed, if the Timer1Value is higher than the Noise Filter-value. This check is needed because there a spikes from the commutation in the beginning of the commutation-sequence. The delay prevents the MCU for misinterpreting this signal with the BEMF.
3. After that an additional check is needed to check that the measured value of the Phase_Voltage is higher than a specific Value (0x150, please check according to your design). This is needed to check if there is a power-supply available.
4. By using a block commutation there is always one phase without voltage. With this phase you are measuring the BEMF. To select which is the right one, the BEMFchannel-variable is needed. This Variable is set in the commutation function. There are three different BEMFchannels – this document shows only one because the other reacts in the same way.

5. The following if clauses checks if the Measured BEMF is between plus/minus delta compare of the Virtual_GND_Voltage. The value delta Compare has to be adjusted, according to your design.
6. If these checks are true, the timers are stopped and the Dualtimer2 is loaded with the value of Dualtimer1. After that, the Dualtimer2 is started again and the Dualtimer1 is loaded with the start-value. At the end of the function the IRQ-flag is cleared.

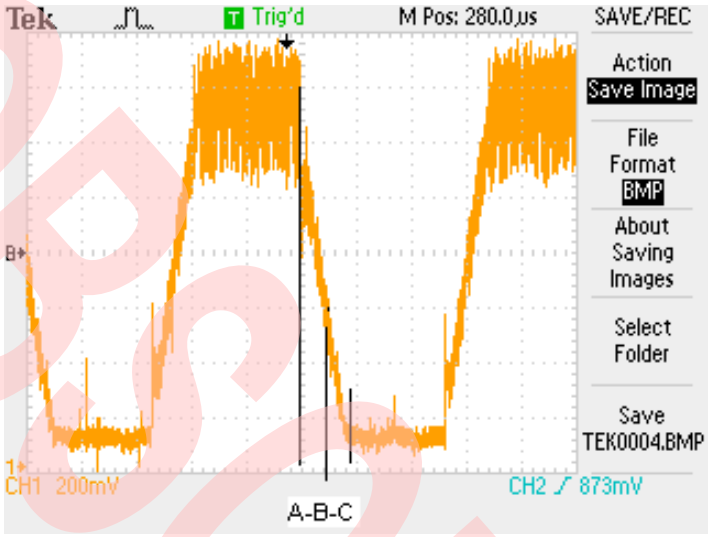
3.5 The Dual Timer Function

```
void initDualTimer (void)
{
    FM3_DTIM->TIMER1LOAD = 0xffffffff;     // Preload Value
    FM3_DTIM->TIMER1CONTROL = 0x63;        // Timer operates in
periodic
                                           // one-shot-mode, 32 bit

    FM3_DTIM->TIMER2LOAD = 0xffffffff;     // Preload Value
    FM3_DTIM->TIMER2CONTROL = 0x63;        // Timer operates in
periodic
                                           // one-shot-mode, 32 bit
}
void DT_Handler (void)
{
    if(bFM3_INTREQ_IRQ06MON_TIMINT0 == 1)
    {
        FM3_DTIM->TIMER1INTCLR = 1;        // Overflow from BEMF
Counter 1
        while(1)
        {
            PosSwitch (0);                 // Shut down PWM!!!
        }
    }
    if(bFM3_INTREQ_IRQ06MON_TIMINT1 == 1)
    {
        //Commutation
        FM3_DTIM->TIMER2INTCLR = 1;           // Clear Interrupt cause
        bFM3_DTIM_TIMER2CONTROL_TIMEREN = 0;  // Disable Timer 2

        if (Startup == 0)                     // Startup done
        {
            if (StartupPOS < 6)               // Switch the sectors
                StartupPOS = StartupPOS + 1;
            else
                StartupPOS = 1;

            PosSwitch (StartupPOS);
        }
```

1. The init Dual Timer function initializes the Dualtimer1 and Dualtimer2 with the start-value 0xffffffff. The Timer operates in periodic one shot mode with a 32bit count base.
2. The DualTimer has only one interrupt-line. For this reason, the Interrupt-cause register has to be read out to check which Timer has generated the interrupt. The fist timer starts at the Value 0xffffffff. If this timer reaches the zero position there was no detection of the VirtualGNDVoltage. It could be possible, that the motor has been stopped. In a real application there might me a more sophisticated error-handler. This funktion only switches the PWM off.

3. If the interrupt is generated from the second channel of the Dual Timer, the timer has reached the zero position. This represents the Position C in the diagram and the commutation has to be done. The PosSwitch-function calls the commutation function to do this.

Figure 3. Closer view of one commutation cycle



3.6 The Multifunction Timer Function

```
void MFT_SETUP (void)
    {
    //Setup IO Ports
    FM3_GPIO->PDOR4 = 0x0;                  // Set output to zero
    FM3_GPIO->PCR4 = 0x0;                   // Don't use Pull ups
    FM3_GPIO->EPFR02 = 0x49221aaa;          // Port input ICU+PPG
output
    FM3_GPIO->DDR4 = 0x3f;                  // Set Port as output
    FM3_GPIO->PFR4 = 0x0f00;                // use as function for ICU
    FM3_MFT1_FRT->TCCP0 = PWM_MAX;          // set compare clear
register
    FM3_MFT1_FRT->TCSA0 = 0x2071;           // enable zero detect IRQ
                                            // Initialize FRT0,up/down
mode
    // INIT OCU
    FM3_MFT1_OCU->OCFS10 = 0x0;             // Connects FRT0 to OCU
    FM3_MFT1_OCU->OCFS32 = 0x0;             // Connects FRT0 to OCU
    FM3_MFT1_OCU->OCFS54 = 0x0;             // Connects FRT0 to OCU
    FM3_MFT1_OCU->OCSA10 = 0xc3;            // Enable OCU 0
    FM3_MFT1_OCU->OCSA32 = 0xc3;            // Enable OCU 0
    FM3_MFT1_OCU->OCSA54 = 0xc3;            // Enable OCU 0
    FM3_MFT1_OCU->OCSC = 0x3f;              // Up/Down-count mode
    FM3_MFT1_OCU->OCCP0 = 0x7e;             // Set to high/2 time
    FM3_MFT1_OCU->OCCP1 = PWM_MAX;          // Set to high
    FM3_MFT1_OCU->OCCP2 = 0x7e;             // Set to high/2 time
    FM3_MFT1_OCU->OCCP3 = PWM_MAX;          // Set to high
    FM3_MFT1_OCU->OCCP4 = 0x7e;             // Set to high/2 time
    FM3_MFT1_OCU->OCCP5 = PWM_MAX;          // Set to high

    FM3_MFT0_WFG->WFSA10 = 0x0000;          // Bypass of the WFG
    FM3_MFT0_WFG->WFSA32 = 0x0000;          // Bypass of the WFG
    FM3_MFT0_WFG->WFSA54 = 0x0000;          // Bypass of the WFG

    FM3_MFT1_ADCMP->ACCP0 = 0;              // trigger on 0 detect
    FM3_MFT1_ADCMP->ACSA = 0x0005;          // Trigger ADC0 by ADTRG
unit
    FM3_MFT1_ADCMP->ACSB = 0x0001;          // Enables Buffer Function

    FM3_MFT1_WFG->NZCL = 0x0009;            // External Fault detect
    }
```

The functions MFT_SETUP setup all parts of the multifunction-timer. For the explanation, please have a look on the comments beside the code.

```
void MFT_WG_IRQHandler(void)
{
    if (bFM3_MFT1_WFG_WFIR_DTIF)
    {
        NVIC_DisableIRQ(WFG_IRQn);              // disable interrupts
        while (1);                              // halt system
    }
}
```

The FM3 has a DTTI-Input included for over-current or -temperature detection. If the microcontroller gets a signal from this Pin, it switches the PWM-Output to a defined status and calls an ISR. In this ISR the customer has to put in the error-handler. In this example a simple error-handler is included which only disables all interrupts and halt the system.

```
void PosSwitch (int POS)
{
    if (Direction==FORWARD)
    {
      if (POS==0)
        SectorSwitch(0);
      if (POS==1)
        SectorSwitch(1);
      if (POS==2)
        SectorSwitch(2);
      if (POS==3)
        SectorSwitch(3);
      if (POS==4)
        SectorSwitch(4);
      if (POS==5)
        SectorSwitch(5);
      if (POS==6)
        SectorSwitch(6);
    }
    if (Direction==BACKWARD)
    {….}

     bFM3_DTIM_TIMER1CONTROL_TIMEREN = 1;
     }
```

The function PosSwitch switches between the 6 sectors of commutation and starts the Timer1 of the Dual Timer.

```
void SectorSwitch (int Sector)
   {
   if (Sector==0)
      {
      bFM3_GPIO_PFR4_P0 = 0;
      bFM3_GPIO_PFR4_P1 = 0;
      bFM3_GPIO_PFR4_P2 = 0;
      bFM3_GPIO_PFR4_P3 = 0;
      bFM3_GPIO_PFR4_P4 = 0;
      bFM3_GPIO_PFR4_P5 = 0;
      }

   if (Sector==1)
   {
      bFM3_GPIO_PFR4_P0 = 0;    // lowside
      bFM3_GPIO_PFR4_P1 = 1;
      bFM3_GPIO_PFR4_P2 = 1;    // highside
      bFM3_GPIO_PFR4_P3 = 0;
      bFM3_GPIO_PFR4_P4 = 0;    // off
      bFM3_GPIO_PFR4_P5 = 0;
      BEMFchannel = 3;          // Channel without Power to detect BEMF
   }
   if (Sector==2)
   {….}
   if (Sector==3)
   {….}
   if (Sector==4)
   {….}
   if (Sector==5)
   {….}
   if (Sector==6)
   {….}
```


The Sector Switch function is the real function to switch the Port off or route it to the Port-Function to make the PWM available.

In the Sector 0, all Pins are routed to the Port which has zero level.

The sector 1 shows a different behavior. The setting in the first two lines represents the setting of the first tree of the inverter bridge.

The settings correspond with the settings in the commutation table

Table 2. Commutation-table without hall Sensors

Angle	0	60	120	180	240	300
Sector	1	2	3	4	5	6
Phase 1	-	O	+	+	O	-
Phase 2	+	+	O	-	-	O
Phase 3	O	-	-	O	+	+

The program shows only Sector 0 and 1. The other sectors correspond with the commutation-table (Table 2).

Document History

Document Title: AN204881 - FM3 Family, Sensorless BLDC Motor

Document Number:002-04881

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	CHRH	09/22/2011	Initial release
*A	5053157	CHRH	12/16/2015	Migrated Spansion Application Note MCU-AN-300404-E-V10 to to Cypress format
*B	5709650	AESATP12	04/26/2017	Updated logo and copyright.
*C	6415018	CHRH	12/18/2018	Obsoleted

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709