AN1728

Making Low-Distortion Motor Waveforms with the MC68HC708MP16

By David Wilson
Freescale Field Applications
Milwaukee, WI

Introduction

As an aspiring[1] musician, I sometimes add a little distortion to my guitar amp to make the sound more interesting or to cover up my mistakes. However, when dealing with a drive connected to a motor, the best distortion is *no* distortion. As Dr. Ned Mohan put it, the goal is harmony, *not* harmonics! In the case of AC (alternating current) induction motors, the desire is to achieve the cleanest sine waves possible coming out of the drive.

But first, the bad news. The 6-transistor inverter topology commonly used in most voltage sourced inverters requires that a "dead-time" must be inserted between the turn off of one transistor in a half-bridge and the turn-on of its complementary device. Otherwise, both transistors in a half-bridge may be on momentarily at the same time, which can have nasty consequences for a drive. As a result of inserting this dead-time, a distortion is introduced in the output voltage and current waveforms when the inverter is driving an inductive load such as a motor.

1. In this case, the word implies a desire to achieve a particular goal which is far, far away.

The good news is that this distortion can be corrected satisfactorily in most situations. By using a LEM sensor or other current sensing device, correction waveforms can be generated which are synchronous to the motor phase currents and applied to the PWM (pulse width modulation) signals.

The great news is that Freescale has developed a patent-pending, sensorless technique to accomplish this without the need for current sensors and has integrated this feature into the MC68HC708MP16 microcontroller. The hope is that, for the first time, the benefits of distortion correction can be brought to the arena of low-cost motor control applications which cannot afford expensive current sensing techniques.

The Problem

To better understand the effects of this distortion, it is necessary to review the equations governing the modulation process.

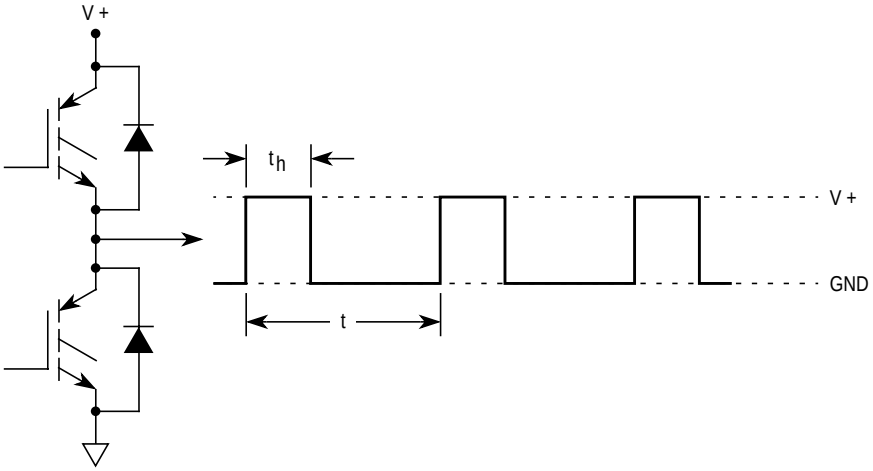


Figure 1. PWM Output of a Typical Half-Bridge

Referring to **Figure 1**, if we assume that the dead-time is zero and sinusoidal modulation is employed, the averaged or filtered output for unity power supply voltage is given as:

$$\overline{V_{o1}(t)} = \frac{t_h(t)}{T} = \frac{1}{2} + \frac{M}{2} \sin(\omega_o t + \theta) \qquad \text{Equation 1}$$

where:

$\overline{V_{o1}(t)}$ = the averaged phase 1 output voltage

$t_h(t)$ = high time of the PWM signal

T = the PWM period

M = the modulation index (from 0 to 1)

The analysis presented here can be applied to other modulation waveforms as well. Solving for the high-time, we obtain:

$$t_h(t) = \frac{T}{2} + \frac{TM}{2} \sin(\omega_o t + \theta) \qquad \text{Equation 2}$$

If we could find a way to keep dead-time equal to zero, I could conclude with equation 2 and complete this application note here. However, as already stated, dead-time is a necessary evil to prevent shoot-through current. **Figure 2** illustrates a half-bridge circuit composed of IGBTs which are attempting to generate a desired PWM output waveform with 50% duty cycle. With no dead-time, this could be achieved easily by turning on the top transistor for half of the cycle and turning on the bottom transistor for the remainder of the cycle. However, with dead-time inserted, the on-times of both the top and bottom transistors are shortened evenly, so that neither the top nor bottom PWM signal corresponds to a 50% duty cycle.

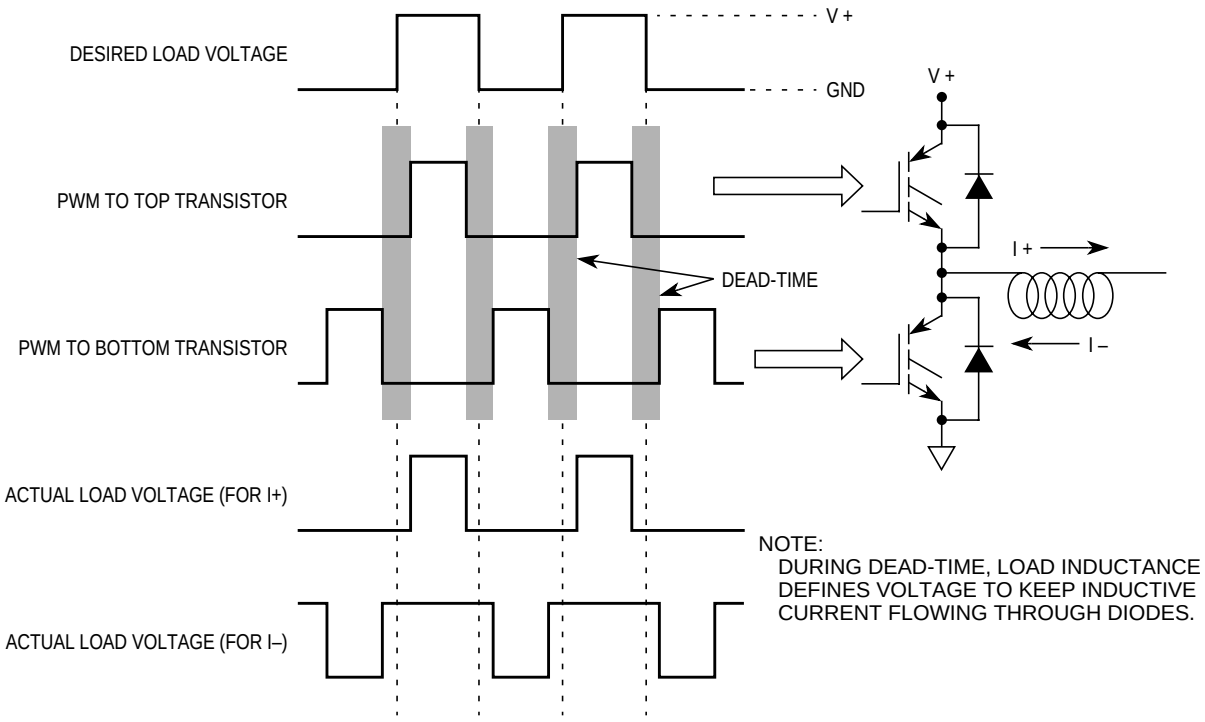


Figure 2. Distortion Created from Load Inductance

What actually happens to the output voltage during dead-time, when both transistors are off and the inverter is in a high-impedance state? The answer is: It depends on what the load characteristics are.

What if the load has inductive properties, like a motor winding? Rule No. 1 about inductors: If current is flowing in an inductor and an attempt to interrupt that current is made, the inductor will fight back. It immediately turns into a voltage source and will achieve whatever voltage is necessary to protect its current flow. (Without the commutation diodes, the inductor would jack up its voltage so high the transistors would blow out.) Therefore, if current is flowing out of the inverter and both transistors are turned off, the inductor voltage seen by the half-bridge will go negative to keep its current flowing out of the inverter, as illustrated in the second to last waveform of **Figure 2**. The bottom waveform shows the condition when current is flowing into the inverter. Now when both

transistors are turned off, the inductor voltage will go positive in an attempt to keep its current flowing into the inverter.

As a result of this inductive action during the dead-time interval, the pulse width of the output voltage will be affected. It will either be larger or smaller than desired (depending on the current polarity) by an amount equal to one dead-time interval. This in turn causes an offset in the average output voltage. Once the voltage waveform has been affected, rest assured that the current waveform will be distorted as well.

Based upon the above discussion, we can now rewrite equation 2 to include a first order approximation of the distortion created by having dead-time inserted:

$$t_h(t) = \frac{T}{2} + \frac{TM}{2} \sin(\omega_o t + \theta) - sgn(i_1)\mathrm{DT} \qquad \text{Equation 3}$$

where:

$sgn(i_1)$ = the polarity of the line current for phase 1

DT = the dead-time

Dividing both sides of equation 3 by the PWM period, T, we can once again solve for the averaged output voltage. The difference between equation 4 and equation 1 is that the effects of the dead-time distortion are now included.

$$\overline{V_{o1}(t)} = \frac{1}{2} + \frac{M}{2} \sin(\omega_o t + \theta) - \frac{sgn(i_1)\mathrm{DT}}{T} \qquad \text{Equation 4}$$

Equation 4 suggests that the distortion of the phase voltage can be approximated as a bipolar squarewave which is synchronized to the phase current signal. **Figure 3** confirms this by showing an actual phase voltage waveform obtained from a Freescale hybrid power module. A pure sine wave reference is given to show the effects of the distortion.

NOTE: *To avoid any potential confusion, notice that the two waveforms in **Figure 3** were taken consecutively and are not aligned in time.*

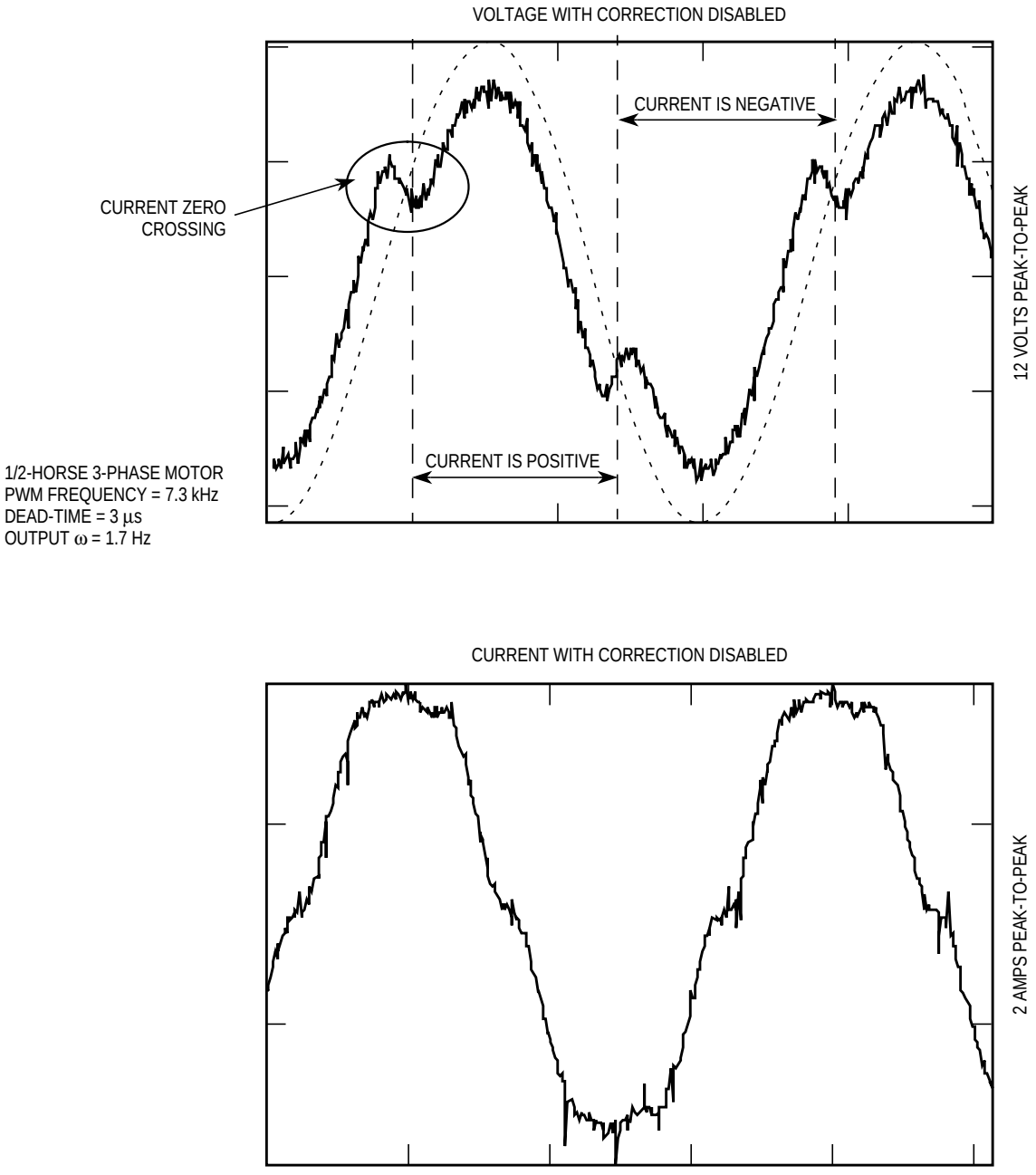


Figure 3. Voltage and Current Distortion Due to Dead-Time

To understand the shape of the current waveform, consideration of the interactions of all three phases of distortion is necessary. Assume that the angular separation of each line current is 120 degrees. Since the distortion voltage is 180 degrees out of phase with the current for each phase, then each of the distortion waveforms is also separated by 120 degrees, as illustrated in **Figure 4**.

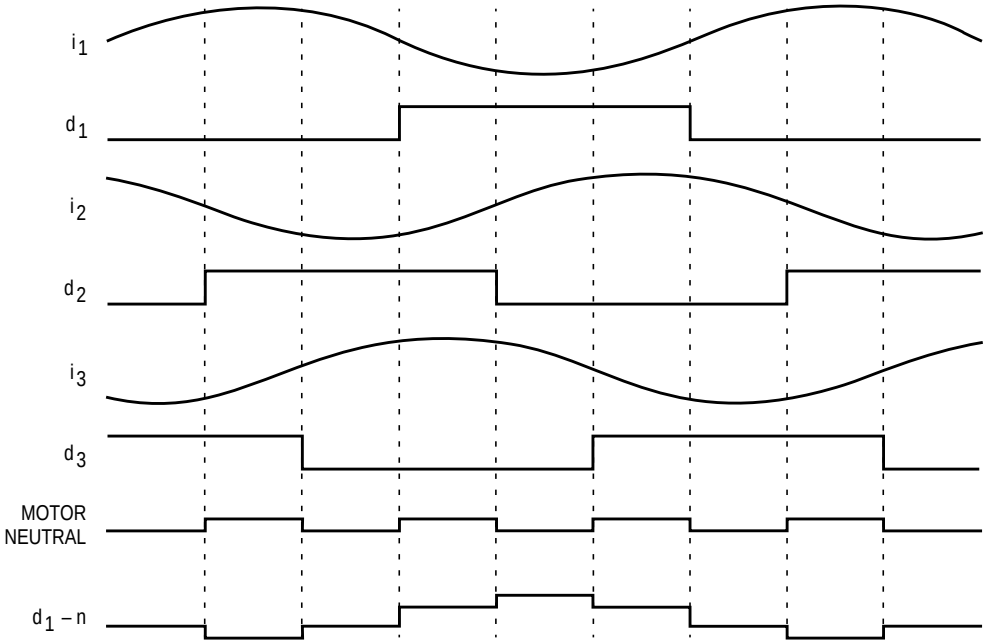


Figure 4. Calculated Distortion Waveforms

Assuming the motor load is linear, we can use superposition and analyze the system response to the distortion alone (for instance, modulation signal equals zero). Under this condition, if we assume that the motor load is balanced, we can average the three distortion voltages to obtain the motor neutral voltage, *n.* Unlike 3-phase sine waves, the distortion waveforms do not result in a neutral voltage of zero over time. Instead, it is a square wave which transitions every 60 degrees.

For the sake of our analysis, let's assume that the motor is a 3-phase *Y* connected load (although the results apply equally to Δ connected loads). By subtracting *n* from d_1, we obtain the distortion voltage which actually is impressed across the phase 1 leg of the load, which is shown

as the bottom waveform of **Figure 4**. This voltage sets up a current in that phase which is a function of the load impedance. When the original sine wave current is added back in by superposition, the result is a current waveform whose peaks are clipped, as seen in **Figure 3**. In fact, under certain conditions when the distortion waveform is large with respect to the modulation waveform, the distortion can actually cause the current to dip at its peaks.

Inquiring minds also will wonder about the mysterious flat spots of the current waveform at the zero crossings in **Figure 3**. Since the source of this distortion is linked intimately with the proposed correction technique, the flat spots are covered in detail in the next section.

From a review of equation 4, **Figure 3**, and **Figure 4**, several characteristics about dead-time distortion are worth mentioning:

1. The amplitude of the distortion per unit of bus voltage is equal to the ratio of the dead-time to the PWM period. Since dead-time is a system parameter which is usually fixed in accordance with the switching characteristics of the power devices, the problem is usually associated with higher PWM frequencies. In some cases, an engineer will specify a higher PWM frequency to minimize THD (total harmonic distortion) without realizing that the distortion from dead-time actually gets worse.
2. The voltage distortion causes a current distortion, and the net result is torque pulsations felt on the motor shaft. Several customers have reported that, under certain conditions, this also translates into stability problems between the motor and drive.
3. Unlike the modulation signal, which is purposefully impressed upon the motor windings, the amplitude of the distortion is *not* affected by the modulation index. This means that the problem, when viewed from a signal-to-noise vantage point, is most severe when the applied motor voltage is small. On an AC (alternating current) induction motor, this occurs when the motor RPM is also small, and the momentum of the rotor cannot smooth out the torque pulsations, making them even more apparent.
4. The distortion is synchronous with the motor current and 180 degrees out of phase with it. At low frequencies, this effect

combines with the stator resistor losses to further reduce the motor torque. Overmodulation in the form of a voltage boost can be used to mitigate this problem. However, the torque pulsations from the distortion are still present.

5. The previous analysis is based on the presupposition that the distortion waveform has a perfect rectangular wave shape. Discussion in the next section shows that variances from this premise have an impact on the correction technique.

The Solution

Since the distortion effect from dead-time can be fairly well characterized, it stands to reason that the cure should be equally straightforward. With the exception of a few second-order effects, this is true. Since the output waveform has a distortion, and the characteristics of that distortion can be approximated closely, the answer is to "counter-modulate" the original PWM signal, essentially to provide noise cancellation. In other words, superimpose a correction signal on top of the modulation signal in the processor to exactly cancel the output distortion. Equation 5 is obtained by modifying equation 3 (which defines the high time of the output waveform in the presence of distortion) to include a correction term to counteract the distortion. Since the distortion signal resembles a square wave, the correction term is also a square wave. Since the distortion signal is synchronized to the current waveform for that phase, the correction term must also be synchronized to the same current waveform.

$$t_h(t) = \frac{T}{2} + \underbrace{\frac{TM}{2}\sin(\omega_o t + \theta)}_{\text{signal}} \underbrace{- sgn(i_1)DT}_{\text{– distortion}} + \underbrace{sgn(i_1)DT}_{\text{+ correction}} \qquad \text{Equation 5}$$

Another way to view the correction process is illustrated in **Figure 5**. Recall from **Figure 2** that the dead-time was balanced between the top and bottom PWM signals. For that particular example, 50% duty cycle was desired. With the insertion of dead-time, the top and bottom PWM signals had their on-times reduced by an equal amount to something less than 50%.

What would happen if we chose not to split the dead-time evenly? **Figure 5** shows the case where all the dead-time is completely relegated to the "recessive" PWM signal. In other words, because the load inductor drives the phase voltage based on its current polarity, the turn-on of one of the transistors is redundant since the inductor drives the voltage in that direction anyway. The problem is identifying which PWM signal (top or bottom) is recessive since it changes depending on the output current polarity. It is, therefore, necessary to have knowledge of the output current polarity for each phase.

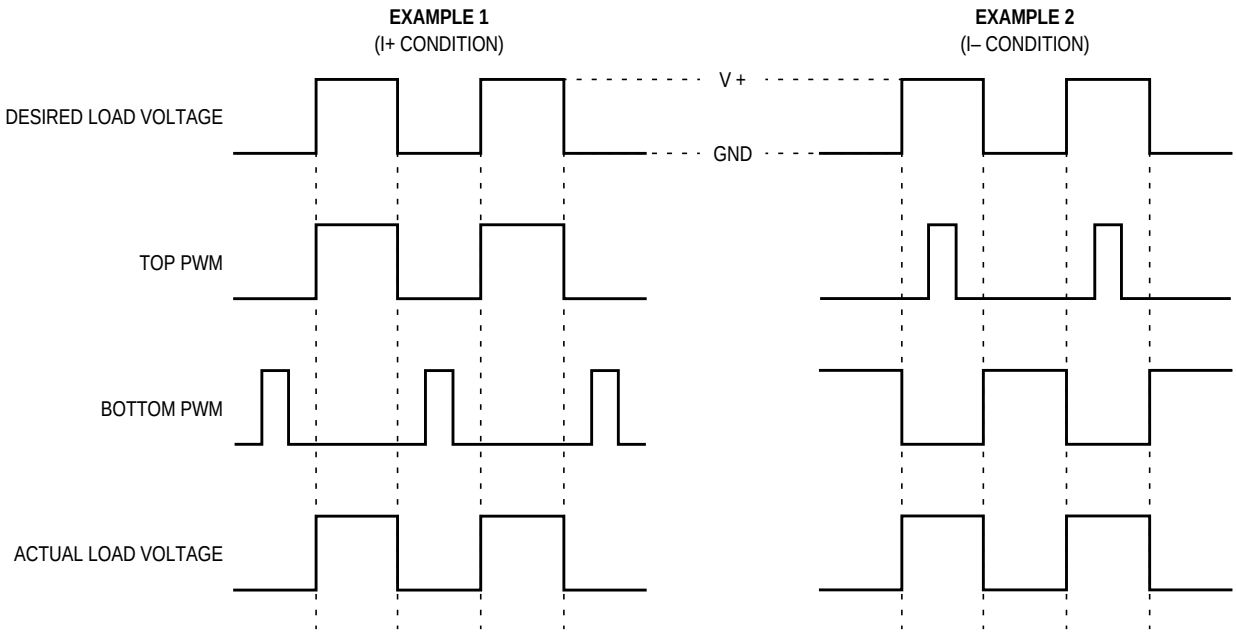


Figure 5. Distortion Correction Accomplished by Redistribution of Dead-Time

The 68HC708MP16 always uses a single PWM register to derive the top and bottom signals for each transistor in a half-bridge and automatically inserts a programmable dead-time in these signals. With no distortion correction applied, the register chosen for each half-bridge driving a motor phase does not change during the course of operation and defaults to an odd numbered PWM value register. When correction is enabled, the PWM module toggles between *two* PWM registers for each motor phase; one when the current polarity for that phase is positive and the other when it is negative. The waveforms of **Figure 5** are obtained by programming one register with the desired pulse width *plus* the dead-time and the other register with the desired pulse width *minus* the dead-time. This is also illustrated in **Figure 6** which shows the voltage waveforms of **Figure 3** plus two additional waveforms which represent the biased voltages in the two PWM registers. In this case, correction is obtained by toggling to register 1 to generate the output waveform when the current is positive and register 2 when the current is negative.

In actuality, to obtain an output pulse width delta of plus or minus one dead-time on the 68HC708MP16 when using center-aligned mode, the correction value should be plus or minus *half* of the value in the dead-time register. This is because the PWM resolution in center-aligned mode is half that of edge-aligned mode.

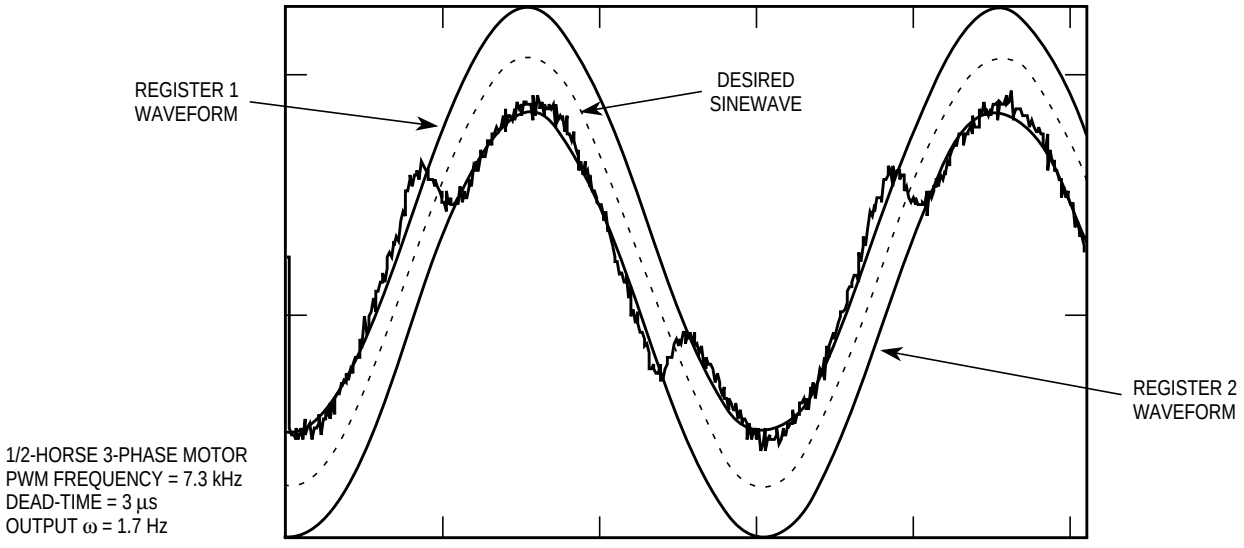
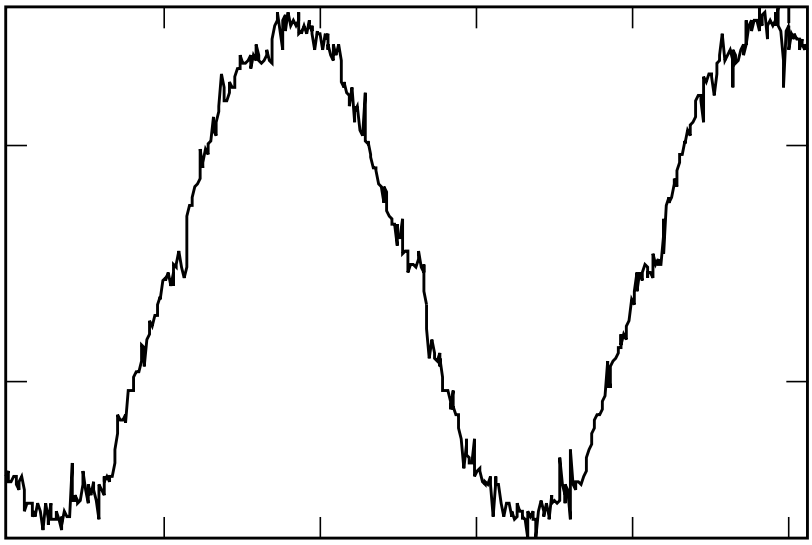


Figure 6. Offset PWM Values in Register 1 and Register 2

One economical way to measure the current polarity for each phase is to use the current polarity sense inputs on the 68HC708MP16. These pins are used to monitor the three PWM voltage waveforms supplied to the motor. Each input is sampled during the dead-time for that particular waveform. If the input is high, the current polarity is determined to be negative and vice versa. Then this information is used to toggle automatically between one of two PWM registers, as discussed above.

When the distortion is corrected in this manner, the current waveform of **Figure 7** is obtained. The modulation index is scaled so that the current peak amplitude matches that of **Figure 3**. Otherwise, the same modulation index results in a 50% increase in the peak amplitude, which demonstrates the severity of the distortion. Although most of the distortion is gone, some distortion still exists at the zero-crossings which causes torque pulsations that can be detected on the motor shaft. To eliminate this distortion, we need to go beyond our first order explanation of the distortion source to understand what is happening to the system during the current zero-crossings.



1/2-HORSE 3-PHASE MOTOR
PWM FREQUENCY = 7.3 kHz
DEAD-TIME = 3 µs
OUTPUT ω = 1.7 Hz

Figure 7. Partially Corrected Current Waveform

Up to now, we have assumed that the distortion waveform is a perfect rectangular wave shape, and, therefore, has consistent rising and falling edges. Related to this premise, we also have implied that the inverter output voltages are either high or low at any given time, including the dead-time intervals. (At this point, you may be suspicious that these assumptions are about to be modified; in which case, your insight serves you well.)

Figure 8 shows the voltage waveforms out of one half-bridge of the inverter under various current conditions. As can be seen, our assumptions are true when the current amplitude is high. However, under low current conditions, which occur near the zero crossings, the inductor is less aggressive in snapping the voltage high or low, presumably due to parasitic capacitance in the motor and drive which can support the inductor's low current flow. Prior to the dead-time interval, if the output voltage already was driven in the direction that the inductor would drive it anyway based on the current polarity, this phenomenon is not observable. The problem occurs when the other transistor was previously on and the inductor must drive the output voltage through the full power supply range during the dead-time interval.

The net effect is that the voltage waveshape does not transition instantaneously between waveforms (3) and (6) when the current polarity changes. Instead, a softer transition occurs which takes the "edge" off of the distortion waveform.

The next question to address is how this effect causes the current to flatten out at the zero crossings.

Let's examine the case of positive inductor current which is decreasing so that the output voltage changes from waveform (3) to waveform (4) of **Figure 8**. As stated earlier, the polarity of the distortion voltage is such that it opposes the polarity of the current. However, as the voltage transitions from waveform (3) to waveform (4), the Δv of the distortion (in the lighter shaded dead-time region) is positive, which contributes to a positive Δi. In other words, the Δv results in negative feedback which opposes the current decreasing further. As a result, the current magnitude is regulated, and the zero crossing is delayed.

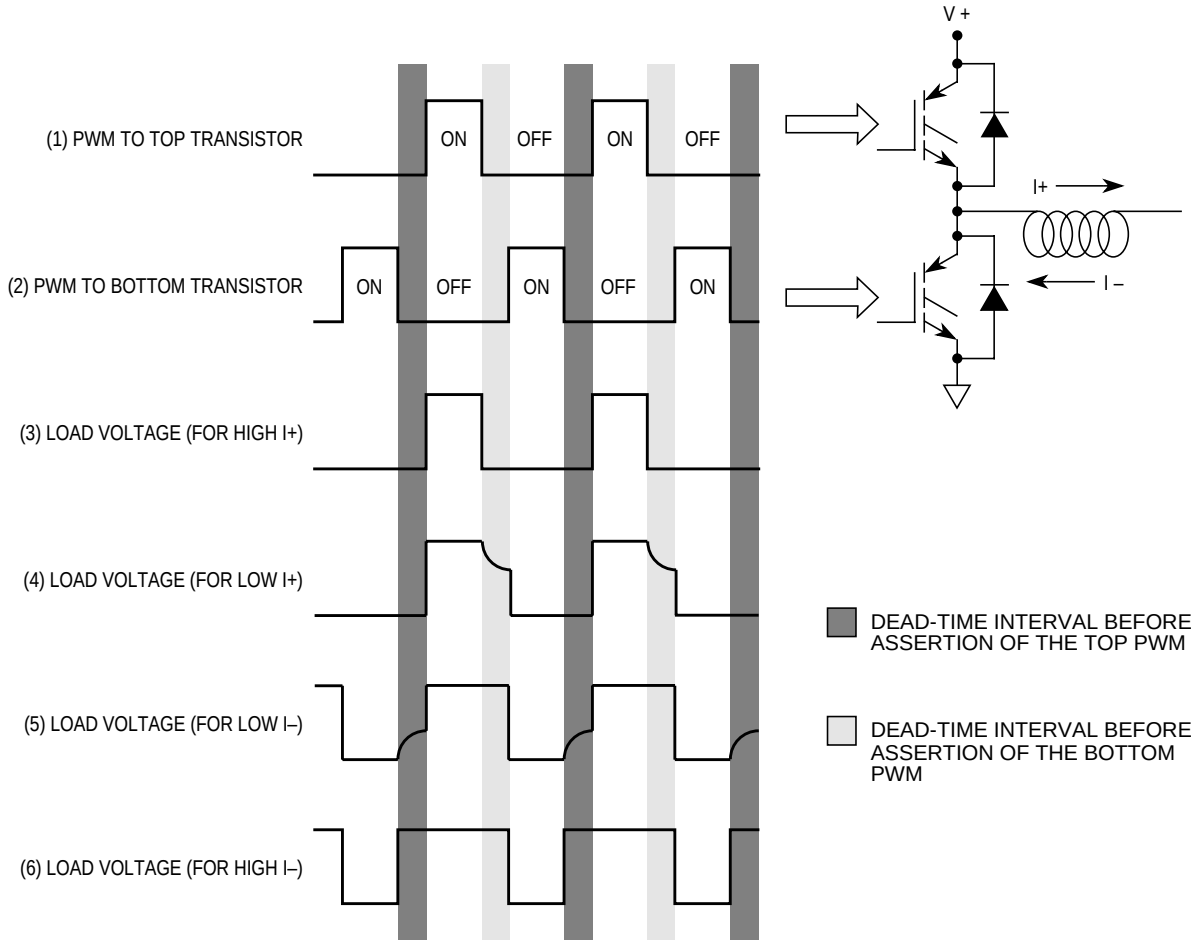


Figure 8. Output Voltage Waveforms under Various Current Conditions

Waveform (5) in **Figure 8** shows the condition of the output voltage shortly after the current zero crossing finally occurs. As the current continues to go more negative, the voltage transitions from waveform (5) to waveform (6). The Δv of the distortion (in the darker shaded dead-time region) is still positive, which continues to oppose the negative trend of the current. As a result, the current remains flattened until waveform (6) is achieved, at which point the Δv in the distortion is zero.

The progression of the waveforms in **Figure 8** correlates with negative trending current, which occurs on the negative slope of the current sine wave. After the current reaches its negative peak and starts positive again, the waveforms of **Figure 8** occur in reverse order from (6) to (3).

The current flattening occurs once again, this time with the negative Δv of the distortion opposing the positive trend of the current.

Applying the previous discussion to the current waveform of **Figure 7**, we realize that even with correction enabled, if we wait for the current polarity to change before toggling the correction value, the current has already flattened out at the zero crossings, and a minimal amount of distortion is still present.

However, what would happen if we didn't wait for the polarity to change? **Figure 9** shows an uncorrected current waveform with proposed thresholds indicating where the correction value should be toggled. If this can be accomplished, the output voltage waveform will transition before the undesirable effects of the Δv of the distortion waveform occur and immediately drive the current through the zero crossings. The flat spots at the current zero crossings will thus be eliminated at the expense of small vertical distortions in the waveform instead.

If a current sensor (such as a LEM sensor) is used, the current waveform could be directly measured and the correction values toggled at the appropriate points on the waveform. However, current sensors (especially with isolated outputs) are expensive. Is there a way to obtain this information without current sensors?

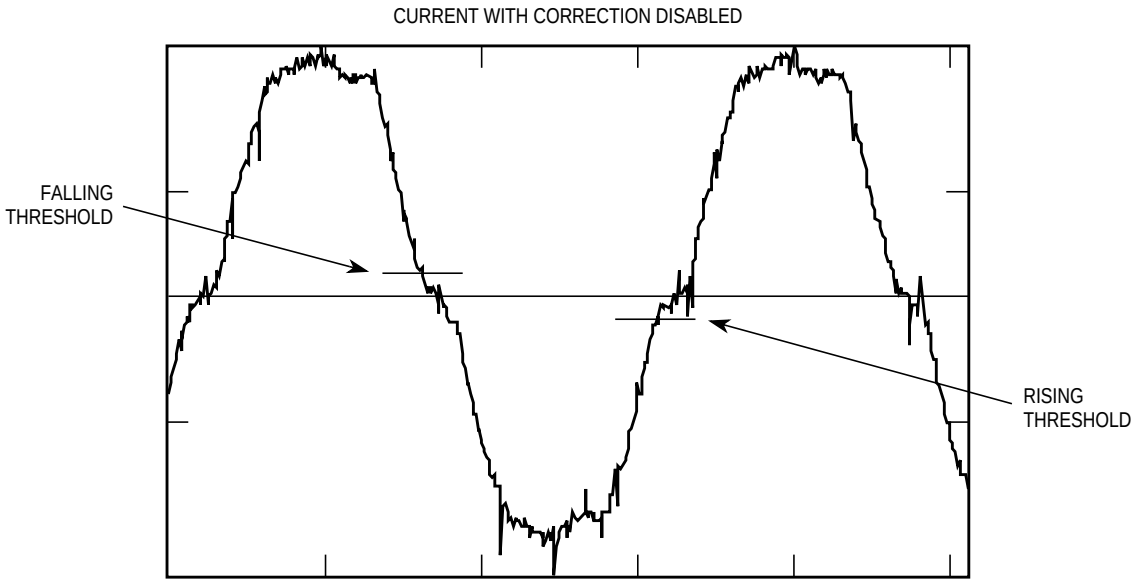
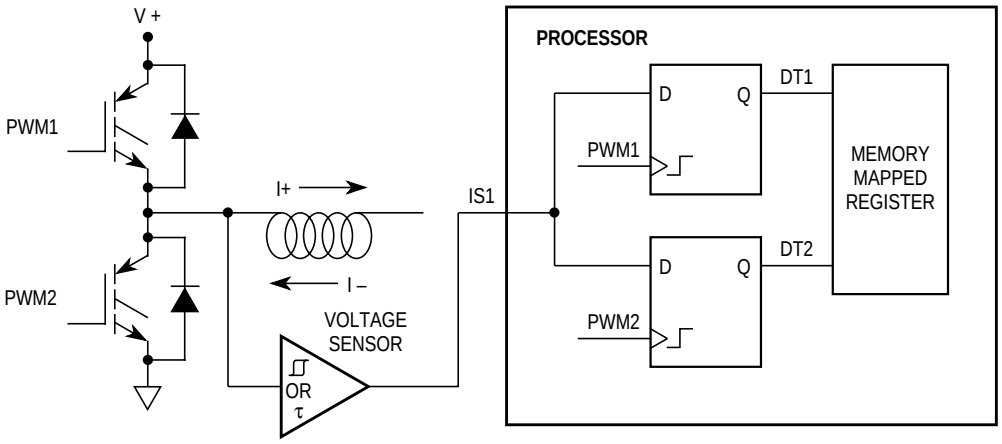


Figure 9. Proposed Current Thresholds for Correction Toggling

Let's return to **Figure 8** and examine the output voltage waveforms under different current conditions. If the lighter shaded dead-time region is compared with the darker shaded region, notice that when the current magnitude is large, the voltages during the two dead-time intervals are the same, regardless of polarity. However, when the current magnitude is small, the voltage waveforms are different in the dead-time intervals. This suggests a strategy for detecting when the current waveform is approaching a zero crossing and acting before it actually occurs. What is needed is a "waveform discriminator," which can tell the difference between the voltage waveforms of the two dead-time intervals under low current conditions.

Figure 10 illustrates a distortion correction system which incorporates such a sensor. By using either hysteresis or a simple low pass filter, the sensor detects whether the load inductor aggressively snaps the voltage waveform during the dead-time intervals. At the end of each dead-time region, the comparator output is sampled by the D-type flip-flops, and the results are stored for both dead-time intervals. As seen in the chart of **Figure 10**, if the voltage waveform is quickly snapped by the inductor, then both results will agree with each other. For example, if both outputs are low, the current is large and flowing out of the inverter. If both outputs are high, the current is large and flowing into the inverter. However, under low current conditions regardless of polarity, the sampled results will be different, indicating to the control algorithm that a current zero-crossing is looming in the near future. The distortion correction value can thus be toggled before the current begins to flatten out.

Figure 11 shows an example schematic of a voltage sensor using hysteresis. In this case, the resistors are sized to work with a bus voltage of about 150 volts. The thresholds on this circuit correspond to input voltages of approximately 15 volts and 125 volts. Also shown as part of the interface is an opto-isolator, which may not be required if the processor is referenced to the same ground as the bus voltage.



BY SAMPLING THE OUTPUT OF THE VOLTAGE SENSOR AS SHOWN INSIDE THE PROCESSOR, THIS INFORMATION CAN BE DEDUCED:

DT1	DT2	LOAD CURRENT CONDITION
0	0	HIGH AMPLITUDE I +
1	1	HIGH AMPLITUDE I–
0	1	LOW AMPLITUDE, EITHER POLARITY

Figure 10. Sense Scheme for Optimized Dead-Time Distortion Correction

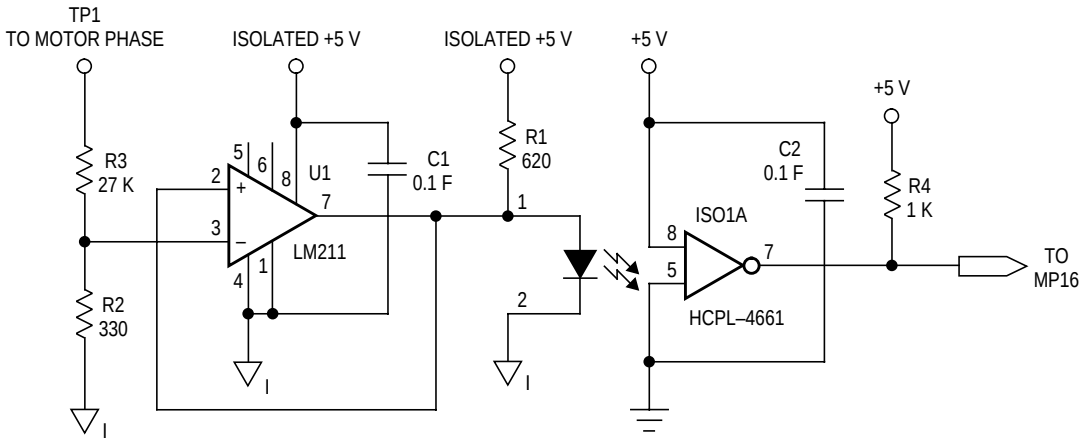


Figure 11. Voltage Sensor Based on Hysteresis

If the bus voltage is constant, then the circuit in **Figure 11** works just fine. However, if the load must be decelerated rapidly or if the motor is connected to a load which can also supply energy, then regeneration can occur which can "pump up" the bus voltage. The need thus exists for a sense circuit whose threshold is scaled from the bus voltage, as shown in **Figure 12**. In this case, the sensed motor voltage waveform is filtered in such a way that only a sharp transition of the waveform at the start of the dead-time interval will result in the comparator threshold being reached by the end of the dead-time, which is when the comparator output is sampled. With the circuit shown, optimum results were achieved when the filter time constant was about 60% of the dead-time value, plus or minus 20%. Again, if the 68HC708MP16 is referenced to the bus voltage ground, the opto-isolator may not be required.

The software can access the outputs of the double-action samplers (2 bits per phase for a total of 6 bits) via location $0024 in the memory map. The bit pattern progression per phase should be %00 at the positive peak of the current waveform, %01 around the current zero crossing, %11 at the negative peak of the waveform, %01 again near the rising zero crossing, and back to %00 at the positive peak. The software is responsible for sensing when the current is near the zero crossing, and changing the correction value appropriately.

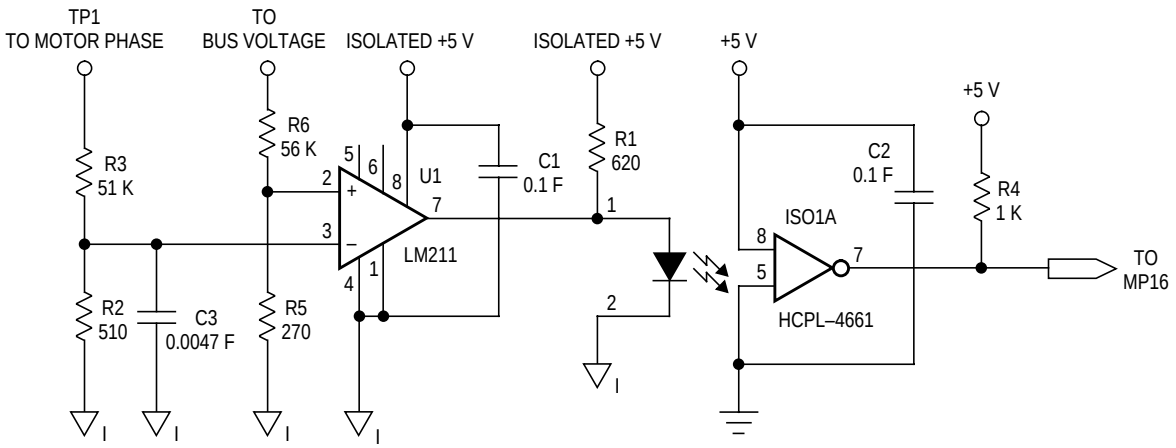
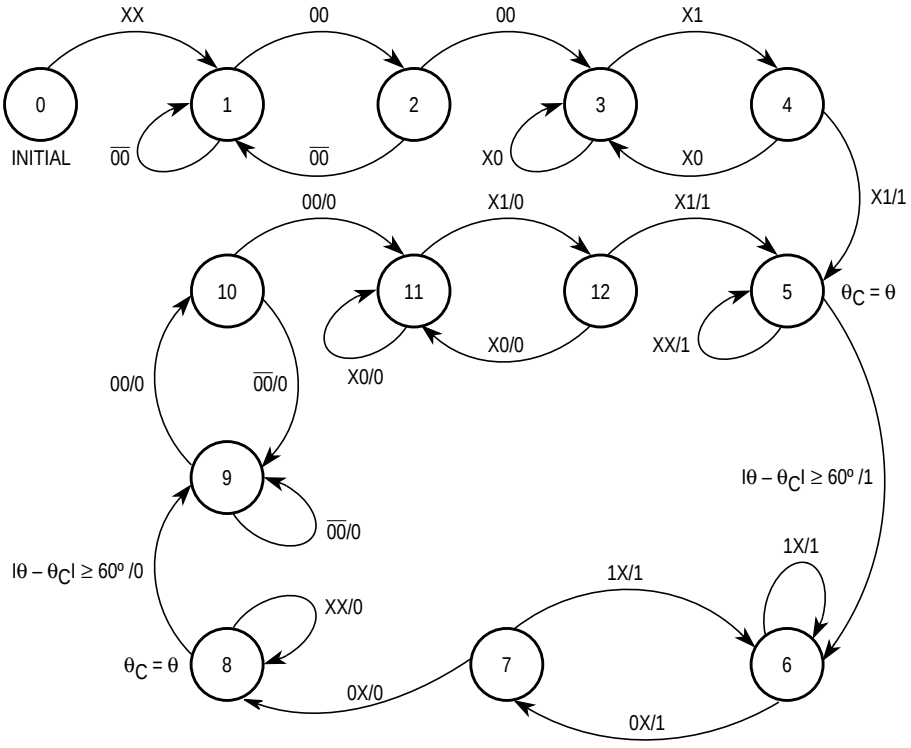


Figure 12. Voltage Sensing Scaled Off of the Bus Voltage

Figure 13 shows a state diagram which illustrates one proposed method of accomplishing this on a per-phase basis. DT1 and DT2 are taken from **Figure 10** and correspond to the two outputs of the double-action sampler. States 5 and 8 correspond to the condition of a near zero current being detected. During these states, the waveform pointer value (θ_C) is recorded, and further polling of the DTn bits is disabled until the waveform angle changes by more than 60 degrees, allowing for a strong %00 or %11 reading to be re-established. States 9 and 10 are included to synchronize the states with the current waveform every positive cycle. Also notice that certain states (2, 4, 7, 10, and 12) exist to confirm a particular DTn bit pattern reading before transitioning to the next state, thus mitigating any potential noise.



STATE TRANSITION KEY: DT1 DT2/IPOL
IPOL = 0: ODD NUMBERED PWM REGISTER CONTROLS OUTPUT
IPOL = 1: EVEN NUMBERED PWM REGISTER CONTROLS OUTPUT

Figure 13. State Diagram for Distortion Correction

Code based on **Figure 13** has been written using the Cosmic C compiler and is listed in the appendix for one phase. The algorithm performs corrections on all three phases simultaneously by driving the IPOLn bits in the PCTL2 register, which in turn dictate which PWM value register (odd or even) is used to derive the output PWM signals for a given phase. The execution of the algorithm is synchronized to run once each time the PWMs are updated. The state information for each phase is preserved as static variables so that execution may resume in the proper states the next time the routine is run.

Assuming an 8-MHz 68HC708MP16 clock frequency, the algorithm processes all three phases in about 20 µs. Even though the Cosmic C compiler generates remarkably tight code, computer jocks who laugh at the perils of assembly language programming could no doubt pick up some more speed.

The Results

If you have made it this far through this application note, congratulations! Having read through the section covering the source of the distortion and lasted through the discussion on how to correct it, you're now ready for dessert! After all, it's the results that count.

By using the techniques described in **The Solution** section, the conditions of **Figure 3** are recreated. Once again, the modulation index is reduced to match the amplitude of the waveforms in **Figure 3**, since removing the distortion creates higher magnitude current waveforms for a given modulation index and thus more torque.

The resulting waveforms shown in **Figure 14** were obtained consecutively, and their time axes are not aligned. Careful examination of the current waveform reveals the early transitions before the zero crossings. However, this distortion is minimal compared to allowing the current to flatten out at the zero-crossing. But best of all, little or no torque disturbance can be detected on the motor shaft, even when loaded.

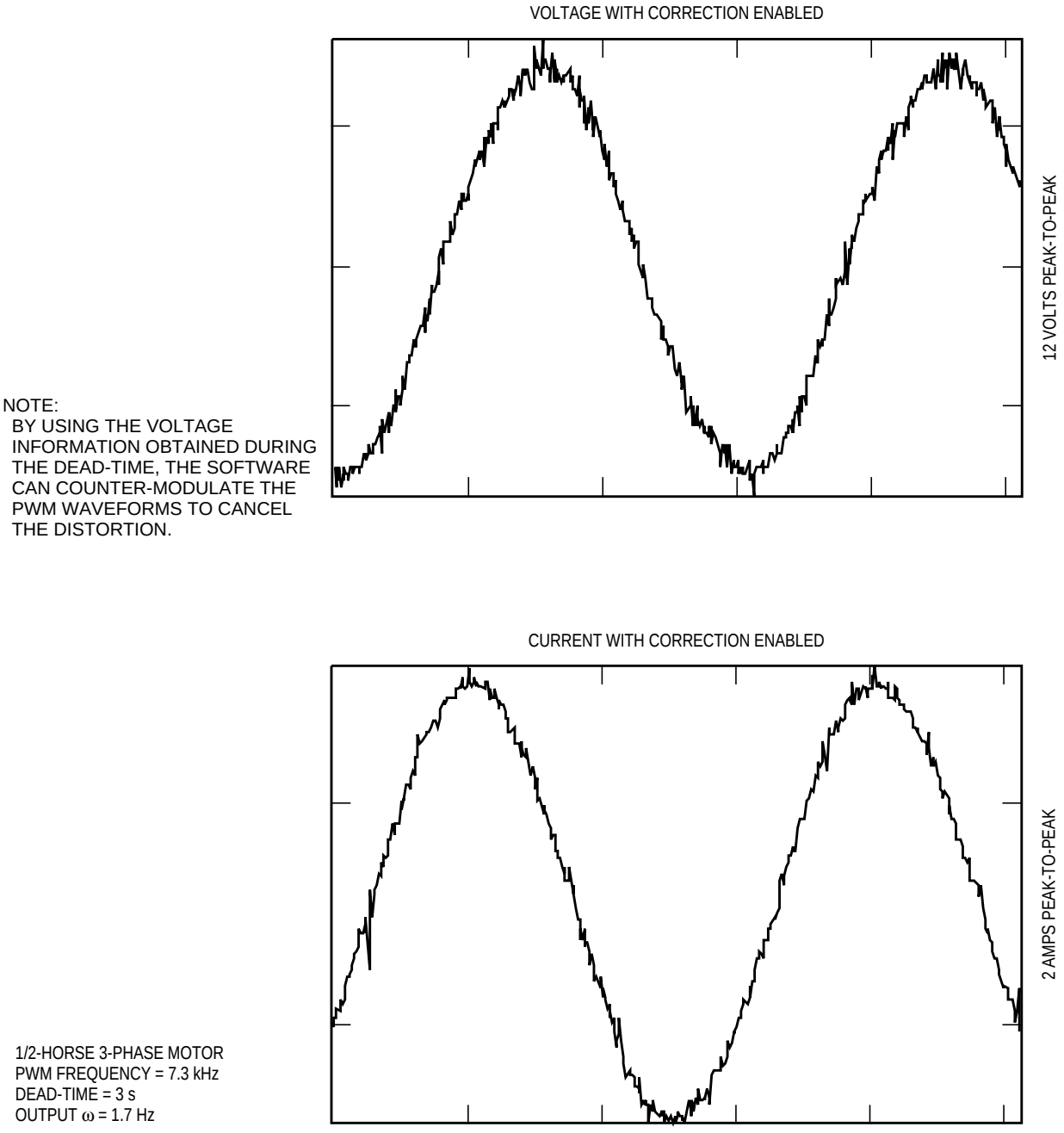


Figure 14. Results of Distortion Correction Using the MC68HC708MP16

Conclusion

A source of distortion has been identified which is common to power stages employing totem-pole transistor configurations requiring dead-time. On AC induction motors running open-loop, the problem typically manifests itself as poor low-speed performance, such as torque ripple and rough operation.

A solution was presented which involves sensing the motor phase voltages during the dead-time intervals and has been implemented on the MC68HC708MP16 microcontroller, which was designed specifically for motor control applications. Software running on the microcontroller can poll the data from the voltage sensing circuitry and modify the modulation waveform to cancel the effects of the distortion. This results in quieter, smoother running, happier motors.

Appendix

```
/*      68HC08MP16 Distortion Correction Test Program
 *      V0.1 Dave Wilson
 *      9 bit PWM
 *      */

#include <mp16io.h>

#include <stdlib.h>


/*      Authorize interrupts  */
#define ENABLE_INT()   _asm("cli\n")
#define DISABLE_INT()  _asm("sei\n")

typedef struct
{
        unsigned char b0:1;
        unsigned char b1:1;
        unsigned char b2:1;
        unsigned char b3:1;
        unsigned char b4:1;
        unsigned char b5:1;
        unsigned char b6:1;
        unsigned char b7:1;
}       FLAGBITS;

typedef union
{
        unsigned char byte;
        FLAGBITS bit;
}       FLAGS

@tiny unsigned int halfPMOD;
@tiny volatile unsigned int pointer;
@tiny FLAGS corr_flag;
@tiny FLAGS flag_a:
@tiny FLAGS flag_b;
@tiny FLAGS flag_c;

extern void pwmcalc();
extern void setvolts(char volts);

void setup(void);
void correct (void);
void correct_a(void);
void correct_b(void);
void correct_c(void);
@interrupt     void     pwm(void)

{
```

```
        COPCTL=0x00;                        /* Tickle the watchdog so that it doesn't
                                               bite!*/
        PCTL1.byte={CT:1/byte & 0xef;       /* clear PWMF bit.*/
        PORTA.bit.b2 = 1;                   /* set I/O pin high for scope reading.*/
        pwmcalc();
        PCTL1.byte=PCTL1.byte | 0x2;        /* set LDOK bit*/
                                            /* set I/O pin low. Measure time of
                                               routine execution with scope*/
        PORTA.bit.b2 = 0;
                                            /* Run distortion correction routine now
                                               that PWM ISR has run*/
        corr_flag.bit.b0 = 1;
}

/*      main program                        */

void main(void)
{
setup();
while (1)
   {
        if (PORTA.bit.b3 != 1)              /* jumper setting to enable correction.
                                            */
                correct();
        else                                /* reset correction variables. */
       {
                flag_a.byte = 0;
                flag_b.byte = 0;
                flag_c.byte = 0;
       }
                                            /* Make a squarewave on a port pin which
                                              corresponds to the voltage waveform of
                                               phase A. Use it as a trigger source
                                              for the oscilloscope to measure P.F.*/

                if (pointer <= 0x7FFF)
                   PORTA.bit.b1 = 1;
                else
                   PORTA.bit.b1 = 0;
   }
}
/*************************************************************************/
void setup (void)

{
        PCTL.byte = 0x30;                   /* use PLL clock*/
        frequency = 20;                     /*specify output frequency to be
                                              approx. 2.2 Hz */
                            /* PWMs are negative polarity, center-aligned,
                               complimentary mode*/
        CONFIG = 0x60;
        PMOD = 0x01FC;      /* PWM frequency set to 7.3 kHz*/
        halfPMOD = 0x00FE;  /* set half of PMOD value */
        PVAL1 = halfPMOD;   /* set pwm 1 to 50% */
        PVAL3 = halfPMOD;   /* set pwm 3 to 50% */
```

```
        PVAL5 = halfPMOD;     /* set pwm 5 to 50% */
        setvolts (25);        /* modulation index set to approximately 20% */
        PCTL1.byte |=0x00;    /* Distortion Correction based on IPOLx bits */
        PCTL1.byte |= 0x20;   /* enable pwm interrupts */
        DEADTM= 28;           /* set the deadtime to 3.8 µs */
        PCTL1.bit.b1 = 1;     /* set LDOK bit */
        DDRA.byte=0xF7;       /* bit 3 is an input, all others are outputs */
        ENABLE_INT();         /* turn on interrupts */
        PCTL2.byte=0x00;      /* interrupt every pwm cycle */
        PCTL1.byte |=1;       /* enable pwm module */
}

/*************************************************************************/
void correct(void)
{
if (corr_flag.bit.b0 == 1)                  /* if PWM ISR has run*/
   {
              corr_flag.bit.b0 = 0;         /* don't run this again until next PWM
                                               ISR */
              PORTA.bit.b0 = 1;             /* set I/O pin high for scope reading */
              correct_a();                  /* distortion correct phase A */
              correct_b();                  /* distortion correct phase B */
              correct_c();                  /* distortion correct phase C */
              PORTA.bit.b0 = 0;             /* I/O high time = correction execution
                                               time */
   }
}

/*************************************************************************/
void correct_a(void)

{
       static @tiny char count_a = 0;
       static @tiny unsigned int point_a = 0;

if (flag_a.bit.b1 == 0)                     /* are we in state 1, 2, 9, or 10? */

   {
       if (FTACK.bit.b1 == 0 && FTACK.bit.b0 == 0)   /* If DT1 and DT2 = 0 */
      {
              ++count_a;                    /* Transition to next state */
       if (count_a == 2)                    /* If we are transitioning to state 3 or
                                               11 */
         {
                     flag_a.bit.b1 = 1;     /* set to state 3 or 11 */
                     count_a = 0;
      }
   }
else                                        /* DT1 or DT2 = 1 */

           { count_a = 0; }                 /* set to state 1 or 9 */
}

                                            /* are we in state 3, 4, 11, or 12? */
else if (flag_a.bit.b2 == 0 && flag_a.bit.b1 == 1)
```

```
{
                if (FTACK.bit.b1 == 1)          /* If DT2 = 1 */

   {
                        ++count_a;              /* Transition to next state */
                if (count_a == 2                /* If we are transitioning to state 5 */

      {
                                                /* Inform the PWM ISR to start applying
                                                   corrected PWMs */
                        flag_a.bit.b0 = 1;
                                                /*Setting flag_a bit 0 means this
                                                  algorithm has synchronized with the
                                                  current waveform, and the PWM routine
                                                  can start generating corrected
                                                  waveforms by loading different values
                                                  in PWM registers 1 & 2. While this bit
                                                  is 0, the PWM routine loads the same
                                                 (uncorrected) values into PWM registers
                                                  1 & 2. */
                        flag_a.bit.b2 = 1;      /* set to state 5 */
                        count_a = 0;
                                                /* Instruct the PWM module to generate
                                                  phase A PWMs from PWM value register 2
                                                */
                        PCTL2.bit.b4 = 1
                                                /* Record pointer value at time of entry
                                                  into state 5 */
                        point_a = pointer;
      }
   }
else                                            /* DT2 = 0 */

{       count_a = 0; }                          /* set to state 3 or 11 */
}
                                                /* are we in state 5? */
else if (flag_a.bit.b3 == 0 && flag_a.bit.b2 == 1)

{
                                                /* if pointer value has changed 55
                                                   degrees from time of entry into state
                                                   5 */
if (abs(pointer - point_a) > 10000)

   {

       flag_a.bit.b3 = 1;                       /* set to state 6 */

   }
}
                                                /* are we in state 6 or 7? */
else if (flag_a.bit.b5 == 0 && flag_a.bit.b3 == 1)
```

```
{
       if (FTACK.bit.b0 == 0)                /* if DT1 = 0 */
   {
       ++count_a;                            /* transition to the next state */
       if (count_a == 2)                     /* if we are transitioning to state 8 */
      {
                       flag_a.bit.b5 - 1;    /* set state to 8 */
                       count_a = 0;
                                             /* Instruct the PWM module to generate
                                                phase A PWMs from PWM value register
                                                1 */
                       PCTL2.bit.b4 = 0;
                                             /* Record pointer value at time of entry
                                               into state 8 */
                       point_a = pointer;
      }
   }

else                                         /* DT1 = 1 */

{ count_a = 0; }                             /* set to state 6 */
}
                                             /* are we in state 8? */
else if (flag_a.bit.b6 == 0 && flag_a.bit.b5 == 1)

   {
                                             /* if pointer value has changed 55
                                                degrees from time of entry into state
                                                8 */

if (abs(pointer - point_a) > 10000
      {
               DISABLE_INT();
               flag_a.byte = 0;              /* set all flags to zero (state 1 or 9) */
                                             /* since the algorithm is already in sync
                                                with the phase A current waveform, it
                                                doesn't need to be initalized again.
                                                This is indicated by setting flag_a
                                                bit 0, which eliminates state 1 as a
                                                possibility, thus specifying state 9.


               flag_a.bit.b0 = 1;
               ENABLE_INT();
      }
   }
}
```

```
/*************************************************************************/
void correct_b(void)

{
                                        /* Since correction is performed on a
                                          per-phase basis, the correction code
                                          for phase B is identical to phase A
                                          except that different variables are
                                           used and different bits are referenced
                                          in the PWM module which pertain to
                                          phase B. */

}

/*************************************************************************/
void correct_c(void)

{                                       /* Since correction is performed on a
                                           per-phase basis, the correction code
                                           for phase C is identical to phase A
                                           except that different variables are
                                          used and different bits are referenced
                                         in the PWM module which pertain to
                                         phase C. */

}
```