Three-Phase Sine-Wave Generation using the PWM Unit of the ADMCF32X

ANF32X-03

Table of Contents

Summary

For any kind of three phase motor controllers, it is necessary to produce six perfectly timed PWM signals. It is usual for three-phase induction, permanent magnet synchronous and brush-less dc motor drives that the motor is supplied from a three-phase hard-switched voltage source inverter. As such, three complementary pairs of PWM signals are required in order to control, respectively, the high and low switches of the three-phase inverter. Ideally, these precise PWM timing signals must be generated with minimal processor overhead. The first part of this application note describes how the dedicated three-phase PWM generation unit of the ADMCF32X is controlled and what possibilities the PWM controller offers.

For correct variable-speed operation of three-phase ac machines, such as the ac induction motor or synchronous motors, it is necessary to supply the machine with a balanced set of three-phase voltages of variable frequency. The most convenient and traditional method of generating such variable-frequency ac voltages is through the use of pulsewidth modulation techniques. The ADMCF32X can be programmed to produce the required PWM signals based on the well-known three-phase sine-triangle PWM techniques. The PWM generation unit is programmed to produce a balanced three-phase set of output signals that have a constant volts/hertz ratio. For three-phase ac motors, adjusting the applied electrical frequency can alter the speed of rotation. In order to maintain an approximately constant magnetic flux in the machine, it is necessary to also adjust the applied fundamental voltage in relation to the frequency. As the magnetic flux is approximately proportional to the ratio of voltage to frequency, the aim is to maintain a constant volts/hertz ratio. The theoretical aspects of this will be described in the second part of the application note.

For three-phase induction motors, the rotor will rotate at a frequency that is somewhat less than the provided electrical input frequency. This asynchronous behavior is fundamental to the induction machine and the difference between the applied electrical frequency and the frequency of rotation is called the slip and is responsible for the torque production in induction motors. For synchronous motors, the rotor will rotate at a frequency exactly equal to the applied electrical frequency.

At last, a software example will be described that may be downloaded from the accompanying zipped files. This example produces output PWM patterns with a certain frequency and output voltage magnitude. The frequency and voltage amplitude is imposed through a data memory location, which can be adjusted from the data memory window of the Motion Control Debugger.

1 The PWM Generation Unit of the ADMCF32X

The PWM generator of the ADMCF32X device is a flexible three-phase PWM generator that can be programmed to generate different types of PWM-patterns. The aim of this application note is to describe the functionality of the PWM generation unit, describe the operation and programming of the registers that determine the operation of the PWM generation unit and provide a simple programming example designed to introduce the operation of the unit.

1.1 Functional Block-Diagram of the PWM Generation Unit

A functional block diagram of the three-phase PWM generation unit of the ADMCF32X is shown Figure 1. This diagram depicts the major functional blocks within the PWM generation unit that are used to produce the three-pairs of PWM signals that appear at the output pins, AH, AL, BH, BL, CH and CL.

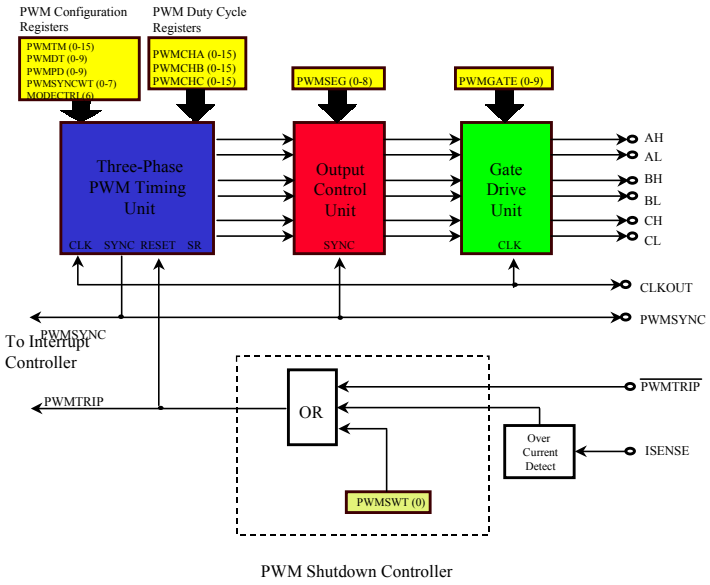

Figure 1: Functional Block Diagram of the PWM-controller of the ADMCF32X

Since the PWM-unit of the ADMCF32X is built upon an independent three-phase timing-unit, that is controlled by three dedicated duty cycle registers, one for each of the three pairs of PWM outputs, no actual limits for the PWM are set. Essentially, the DSP code can be constructed to produce the required PWM strategy that is required by the application.

Each of the six PWM outputs signals can be enabled or disabled by separate output enable bits of the PWMSEG register[1]. In addition, three control bits of the PWMSEG register permit crossover of the two signals of a PWM pair for easy control of electronically commutated motors (ECM) or brushless dc motors (BDCM). In crossover mode, the PWM signal destined for the high side switch is diverted to the complementary low side output and the signal destined for the low side switch is diverted to the corresponding high side output signals.

As can be seen in Sections 1.6.1 and 1.6.2 the PWM controller can operate in two different modes, *single and double update modes*. In single update mode, symmetrical PWM patterns are generated while in double update mode asymmetrical patterns can be produced. The asymmetrical pattern produces lower harmonic distortion in three-phase PWM inverter-fed systems and can therefore successfully be used in noise-critical control-systems. Additionally, the use of double update mode permits the applied voltage to the motor to be updated at a rate that is twice that of the inverter switching frequency. In high-performance applications, such as robotics and servo-drives, this feature permits higher closed loop control bandwidths to be achieved.

1.2 PWM Timer Operation

The PWM timer that is clocked at the DSP instruction rate, with period t_{CK} controls the internal operation of the PWM generation unit. The operation of the PWM timer over one full PWM period is illustrated in Figure 2 where the contents of the 16-bit PWMTM register is used to control the switching frequency of the PWM signals. It can be seen that during the first half cycle (bit 3 of the SYSSTAT register is cleared),

[1] For further details see the datasheet for the ADMCF32X

the PWM timer decrements from PWMTM to 0. At this point, the count direction changes and the timer continue to increment to the PWMTM value. Also shown in Figure 2 are the PWMSYNC pulses, which are produced at the dedicated PWMSYNC output pin, for operation in both single and double update modes. Clearly, an additional PWMSYNC pulse is generated at the mid-point of the PWM cycle in double update mode. Of course, the value of the PWMTM register could be altered at the mid-point in double update mode. In such a case, the duration of the second half period (SYSSTAT bit 3 is set) could be different to that of the first half cycle.

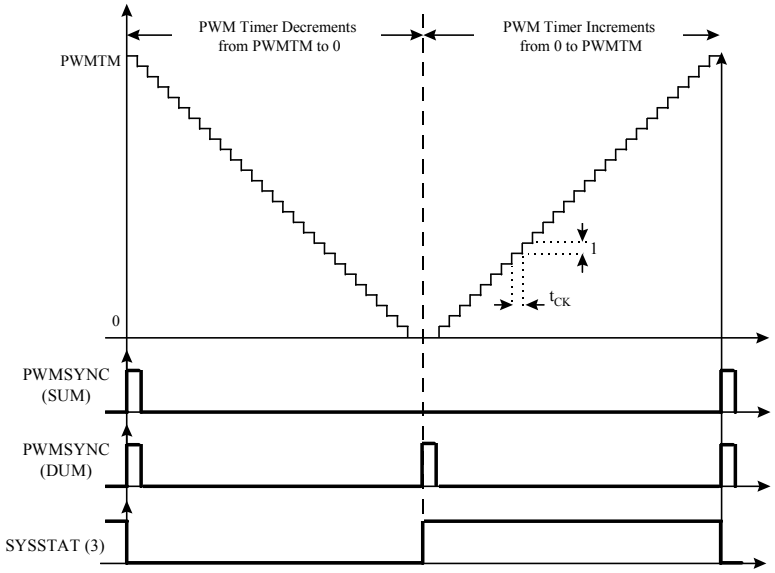

Figure 2: Operation of Internal PWM Timer of ADMCF32X

1.3 Startup of PWM Controller

It is important to fully understand the operation of the PWM controller during startup. Following a power up or reset operation (either hardware of software), the PWM generation unit is placed in a **lock-off** state and all six PWM output signals (AH to CL) are placed in the OFF state (as defined by the PWMPOL pin). The first write operation to any of the PWMTM, PWMCHA, PWMCHB or PWMCHC registers will cause the PWM generation unit to transition to a **disable** state and the PWM timer begins to increment from zero. During this time all six PWM outputs are still in the OFF state and the PWMSYNC pulse is not produced.

The PWM generation unit does not enter the **normal** state until all four of the PWMTM, PWMCHA, PWMCHB and PWMCHC registers have been written to and the first load signal is generated. The load signal is generated each time the PWM timer reaches the value in the PWMTM register in single update mode[2]. An additional load signal is generated in double update mode[3] when the PWM timer reaches 0. The load signal is used to latch new values into each of the PWMTM, PWMCHA, PWMCHB, PWMCHC, PWMDT, PWMPD, PWMSEG and PWMSYNCWT registers.

[2] See Section 1.6.1 Single Update Mode

[3] See Section 1.6.2 Double Update Mode

The operation during startup will be somewhat different depending on whether or not the PWMTM register is the first PWM register that is written. If the PWMTM register is written first, the PWM timer will begin incrementing from zero to the PWMTM value. This actually corresponds to the second half cycle (SYSSTAT bit 3 is set) of the PWM as can be seen from Figure 2. During this time, the PWM unit is in the **disable state** and all six PWM outputs are OFF. Provided the PWMCHA, PWMCHB and PWMCHC registers have all been written to at the time the PWM-timer reaches the PWMTM value, the first internal load signal will then be generated when the PWM timer reaches the PWMTM value. At this point the PWM generation unit enters the **normal** state and the new values are latched into the PWMCHA, PWMCHB and PWMCHC registers. However, no PWMSYNC pulse is generated at this first load signal. The values written to the PWMCHA, PWMCHB and PWMCHC registers (in conjunction with the other PWM registers such as the PWMDT, PWMPD, PWMSEG, PWMGATE registers) define the PWM outputs during the next PWM cycle (in single update mode) or the next half cycle (in double update mode). The appearance of the first PWMSYNC pulse (and associated interrupt) will occur at the end of this first PWM cycle in single update mode when the PWM timer again reaches the PWMTM value. The first PWMSYNC pulse and interrupt will occur at the mid-point of the first PWM cycle in double update mode when the PWM timer reaches zero. Therefore, in single update mode the first PWMSYNC pulse will occur one and a half PWM periods or:

$$T_{1,SUM} = 1.5 \times 2 \times \mathrm{PWMTM} \times t_{CK} \quad (1)$$

after the initial write to the PWMTM register. In double update mode the first PWMSYNC pulse appears one full PWM period or:

$$T_{1,DUM} = 1.0 \times 2 \times \mathrm{PWMTM} \times t_{CK} \quad (2)$$

after the initial write to the PWMTM register.

The startup operation of the PWM unit is illustrated in Figure 3 for single update mode. The first PWM register that is written to is the PWMTM register that begins the incrementing of the timer from 0. It can be seen from Figure 3 that both the AH and AL outputs (as well as the BH, BL, CH and CL outputs) are in the OFF state during this initial half period. Assuming an initial value of 0 is written to the PWMCHA register (and some initial value is written to the other two duty cycle registers) prior to the PWM timer reaching the PWMTM value, the AH output stays in the OFF state and the AL output goes to the full ON state for the entire next PWM period. Only at the end of this cycle is the first PWMSYNC pulse generated. Therefore, if the value in the PWMTM and duty cycle registers have not changed the AL and AH signals will remain in the same state for the next PWM period.

If the PWMTM register is not the first PWM register written to but instead PWMCHA, PWMCHB or PWMCHC is written to first, then the operation is different to that described above. The act of writing to one of the duty cycle registers causes the PWM timer to begin incrementing prior to writing to the PWMTM register. The PWMTM register will thus contain its default-reset value of 0. Therefore, the first load signal would not be generated until the PWM timer increments to 0 (corresponding to a counter overflow from 0xFFFF). Therefore, the PWMTM register will not be loaded with its initial value until $65{,}535 * t_{CK}$ (3.28 ms at 20 MHz CLKOUT) after the initial write to the first duty cycle register. Therefore, it is recommended that the PWM system be initialized by first writing to the PWMTM register and immediately following with initial writes to the three duty cycle registers and any other appropriate configuration registers.

After any shutdown situation of the PWM-controller, it is only possible to re-enable the PWM by writing to all of the four registers (PWMTM, PWMCHA, PWMCHB and PWMCHC). This can of course only be done when the external/internal fault has disappeared.

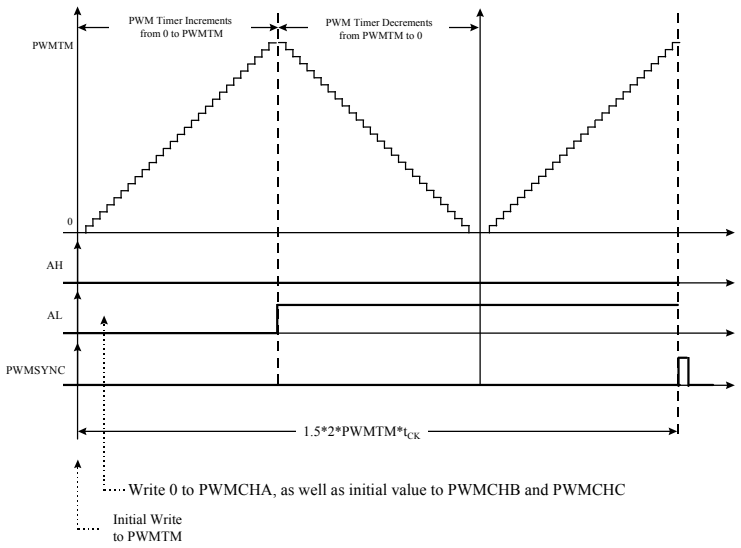

Figure 3: Startup of PWM Generation Unit in Single Update Mode (Active HI PWM).

1.4 PWM switching frequency - PWMTM Register

The switching frequency of the PWM is directly commanded from the PWMTM register. Depending on the type of controller / drivesystem the DSP has to control, it is important to chose the correct switching frequency, f_{PWM}. In most cases the switching frequency is chosen to be above the audible area > 15 kHz, so that the drive system makes as little audible noise as possible. The 16-bit PWM generation unit of the ADMCF32X can generate PWM switching frequencies from a minimum value of 153 Hz (when PWMTM = 0xFFFF). The maximum switching frequency is limited only by the acceptable resolution of the PWM process since the fundamental time increment of the PWM generation process, equal to the DSP instruction rate, t_{CK}, is fixed.

To generate PWM signals, an initialization of the PWMTM register needs to be accomplished during startup. The value is directly determined from the desired switching frequency, f_{PWM}, and the DSP clock rate, f_{CLKOUT}, and can be expressed as:

$$PWMTM = \frac{f_{CLKOUT}}{2 * f_{PWM}} \quad (3)$$

so that the switching period, T_s, is given by:

$$T_S = 2 * PWMTM * t_{CK} \quad (4)$$

EXAMPLE:

With the DSP clock rate of 20 MHz, and a chosen switching frequency of 10 kHz the PWMTM register has to hold the integer number of:

$$\text{PWMTM} = \frac{20*10^6}{2*10^4} = 1000 = 0\text{x}3\text{E}8 \quad (5)$$

1.5 Dead-time - PWMDT register

The dead-time register, PWMDT, is a parameter that controls the delay between the switching of two related switches, eg. AL and AH. If there were no delay between these signals, a short-circuit could appear and the power-switches could be damaged. The PWMDT is a 10-bit register that determines the introduced dead time as multiples of twice the DSP clock rate as:

$$T_D = 2 * PWMDT * t_{CK} \quad (6)$$

If PWMDT = 10 (0x00A) the dead time is 1 µs when t_{CK} is 50 ns. The maximum dead time can be calculated when the 10-bit PWMDT register contains its maximum value:

$$T_D = 2 * (2^{10} - 1) * t_{CK} = 2 * 1023 * \frac{1}{20 * 10^6} = 102 \mu\ s \quad (7)$$

1.6 PWM Duty-cycles:

The six PWM-outputs, AL to CH are controlled by the 16-bit PWM duty cycle registers PWMCHA, PWMCHB and PWMCHC, (see Figure 1). The 16-bit value of the registers control the duty cycle of the signals on the six PWM-outputs. The integer value of the register PWMCHA controls the duty-cycle of AH and AL, and so on for the other two registers. The duty-cycle registers are programmed in integer counts of the fundamental time unit, t_{CK}, and define the desired on-time of the high side PWM signal, produced over half the PWM-period.

1.6.1 Single Update Mode

In single update mode the three sets of PWM are perfectly symmetrical about the midpoint of the switching period. In this mode, as can be seen in Figure 4, the dead time, PWMDT, is incorporated to achieve the desired duty-cycle without any damaging of the power-switches.

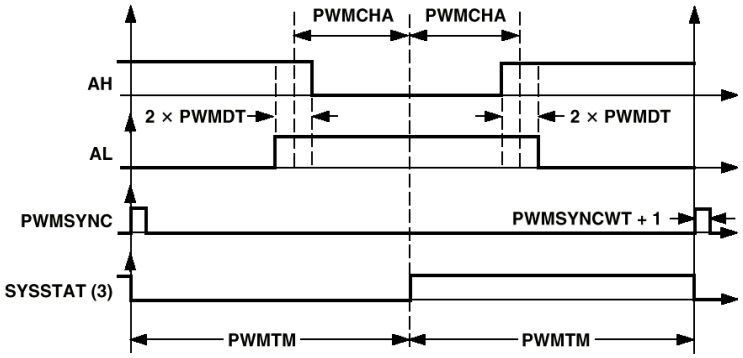

Figure 4: Single update-mode of the PWM unit of the ADMCF32X (Active Low)

The expression of the on-times for the PWM-signals can be written as:

$$T_{AH} = 2 * (PWMCHA - PWMDT) * t_{CK} \quad (8)$$

$$T_{AL} = 2 * (PWMTM - PWMCHA - PWMDT) * t_{CK} \quad (9)$$

Where the duty-cycle are:

$$d_{AH} = \frac{T_{AH}}{T_S} = \frac{(PWMCHA - PWMDT)}{PWMTM} \tag{10}$$

$$d_{AL} = \frac{T_{AL}}{T_S} = \frac{(PWMTM - PWMCHA - PWMDT)}{PWMTM} \tag{11}$$

Obviously negative value of T_{AH} and T_{AL} are not permitted and the minimum permissible value is zero, corresponding to 0% duty-cycle. In a similar fashion, the maximum value is Ts, corresponding to a 100 % duty-cycle.

This application note describes the implementation of library routines (See Section 3) that configure the PWM block in this mode. As will be seen in Section 4, these routines allow for simple generation of PWM patterns without having to manually calculating register values.

1.6.2 Double Update Mode

As already mentioned the ADMCF32X – series offers an unsymmetrical PWM mode besides from the standard **Single update mode**. This mode, **Double update mode** are not centered on the midpoint of the PWM period but are updated twice within one period. As it is illustrated in Figure 5, the switching frequency, dead time and duty-cycle can be changed in second (or first) part of the PWM period. This plot illustrated a completely general case where the switching frequency is changed also at the midpoint of the period. Of course, such a change is not necessary and, if desired, only the duty cycle values need be changed at the mid-point so that the switching frequency remains constant. This offers various possibilities in creating different switching patterns designed for any specific control demands.

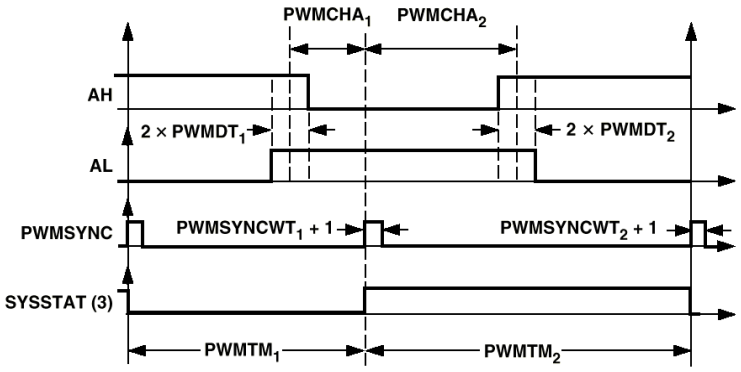

Figure 5: Double update-mode of the PWM unit of the ADMCF32X

In the case of double update-mode the on-times can be defined as:

$$T_{AH} = (PWMCHA_1 + PWMCHA_2 - PWMDT_1 - PWMDT_2) * t_{CK} \tag{12}$$

$$T_{AL} = (PWMTM_1 + PWMTM_2 - PWMCHA_1 - PWMCHA_2 - PWMDT_1 - PWMDT_2) * t_{CK} \tag{13}$$

Where the duty-cycle again can be expressed as:

$$d_{AH} = \frac{T_{AH}}{T_S} = \frac{(PWMCHA_1 + PWMCHA_2 - PWMDT_1 - PWMDT_2)}{PWMTM_1 + PWMTM_2} \tag{14}$$

$$d_{AL} = \frac{T_{AL}}{T_S} = \frac{(PWMTM_1 + PWMTM_2 - PWMCHA_1 - PWMCHA_2 - PWMDT_1 - PWMDT_2)}{PWMTM_1 + PWMTM_2} \quad (15)$$

Regarding the switching period in double update mode it is directly derived as in single update mode and given by:

$$T_S = (PWMTM_1 + PWMTM_2) * t_{CK} \quad (16)$$

For all the six signals; AH to CH, identical equations counts, such as PWMCHB and PWMCHC can be calculated in the same way as for PWMCHA.

1.7 PWMTRIP - Shutdown

In case of any external fault it is very important that all the six PWM signals are disabled immediately, to ensure the "safe mode" for the controller. A falling edge on the $\overline{\text{PWMTRIP}}$ pin provides an instantaneous, asynchronous shutdown of the PWM-controller, as illustrated in Figure 1. When the $\overline{\text{PWMTRIP}}$ has occurred the PWMSYNC pulses are disabled and the associated interrupt is stopped. By software it is also possible to read the state of the $\overline{\text{PWMTRIP}}$ pin, by monitoring bit 0 of the SYSSTAT register.

Another way of detecting a fault condition is though the I_{sense} pin on the analog block of ADMCF32X. Though the analog block the feedback from the DC-link can be monitored which represent the total current to the motor. When the voltage on Isense goes below Isense trip threshold $\overline{\text{PWMTRIP}}$ will be internally pulled low. The negative edge of the internal $\overline{\text{PWMTRIP}}$ will generate a shutdown in the same manner as a negative edge on pin $\overline{\text{PWMTRIP}}$. This fault condition corresponds to a 5.5 A trip current in a 0.10 Ω sense resistor in the DC-link bus.

In addition to the hardware shutdown, writing to the 1 bit read/write PWMSWT register can initiate a software shutdown of the PWM unit. This act disables the PWM-controller in the same way as the external fault condition. However, following a PWM shutdown, it is possible to determine if hardware or software generated the shutdown, by reading the PWMSWT register.

2 Generating Sinusoidal PWM Patterns.

Due to the design historic aspect in AC-motor designs most of the motors used with power electronics are design to operate on sinusoidal supplies. Therefore the inverter output voltage should be as nearly sinusoidal as possible. Adjustable frequency operation of these motors requires a symmetrical set of three-phase sine modulated PWM waveforms, adjustable in both amplitude and frequency. The reference voltages (sine waves) should be adjustable in the full speed range, normally in the area from a couple of hertz to several hundred hertz. Accordingly, if the motor has to be controllable from very low-speed the voltage amplitudes have to follow the same range. In the past, these variable-voltage, variable-frequency voltage references were generated using analog electronics. The PWM patterns were typically generated by comparing these analog reference voltages with a high-frequency triangular carrier wave at the desired switching frequency (typically between 5 and 20 kHz). Now, with the integrated PWM Generation Unit of the ADMCF32X, it is possible to compute the desired pulsewidths in the DSP and use the PWM Generation Unit to produce the output patterns.

2.1 Switching and Power devices

The desired output voltages are achieved by varying the frequency and amplitude of the reference voltage. The results of the comparisons between the sinusoidal references and the carrier waveform are the PWM signals used to control the power devices of the voltage source inverter that is used to supply the motor.

The variations in the amplitude and frequency of the reference voltages change the pulse width patterns of the voltages but keep the sinusoidal modulation, as shown in Figure 6.

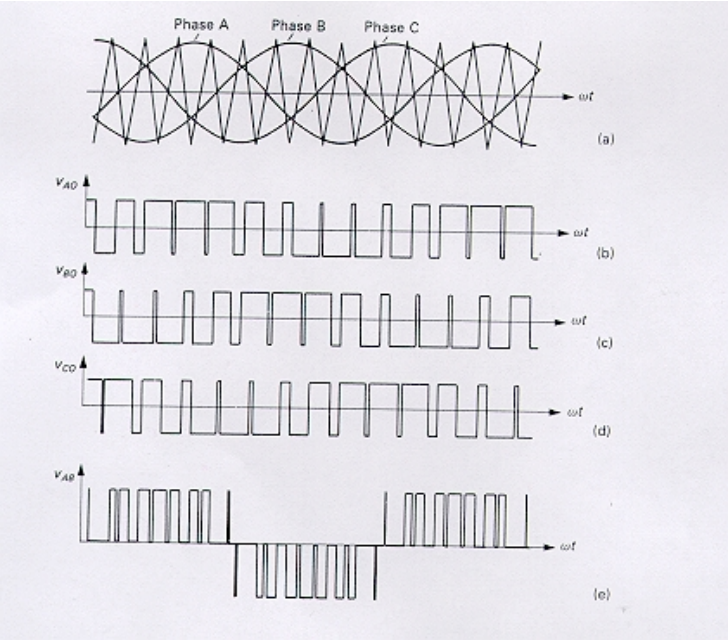

Figure 6: Three-phase sinusoidal PWM: **(a) Reference voltages and Triangular Wave Carrier, (b, c, d) Pole voltages and (e) Motor Line Voltage**

In order to supply the motor, a power converter is needed that translates the low-level PWM signals from the processor to the appropriate high voltage levels. The most common such power converter is the voltage source inverter, shown in Figure 7, constructed of six power switching devices such as MOSFETs or IGBTs. The choice of switching device is based on the desired operating power level, required switching frequency and acceptable inverter power losses.

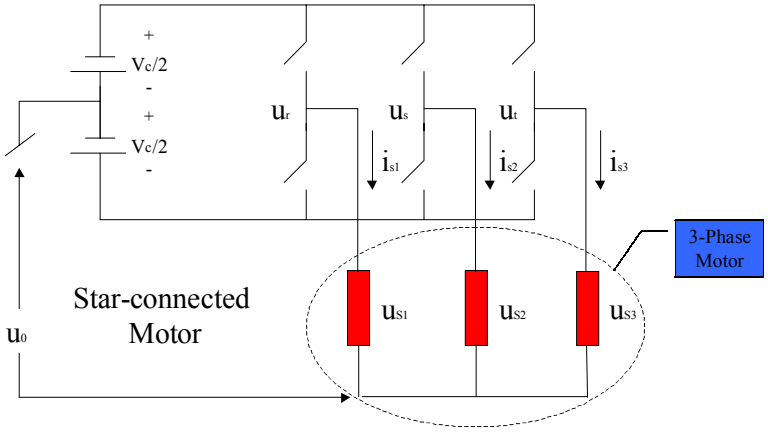

Figure 7: Standard Three-Phase Voltage Source Inverter with Motor Load

2.2 Computation of Desired Duty Cycle Values

Figure 8 shows an expanded view of the PWM generation process for a single PWM switching cycle. The diagram illustrates the relationship between the assumed reference voltage waveform, v_{refA}, and the resultant PWM signal, AH. In this application note, sinusoidal modulation is assumed so that the reference voltage is assumed to take on a sinusoidal time variation at the desired fundamental frequency and voltage level. However, the computation of the required on-time of the PWM signal, $T_{ON,A}$ is independent of the particular modulating function used.

Due to the digital implementation in the ADMCF32X, the reference voltage waveform is effectively sampled at a rate equal to the switching frequency. The sampled value of the input reference voltage (taken at the start of each modulation cycle) is used to compute the duty cycle for the entire PWM period. This strategy results in symmetrical PWM patterns using the so-called *Single Update Mode* of the PWM generation unit. Asymmetrical patterns could be produced using the *Double Update Mode*. In this mode, the PWMSYNC interrupt is generated at a rate twice that of the switching frequency. This permits the computation of the reference voltage level twice per switching period so that an additional sample of the v_{refA} waveform is effectively taken at the mid-point of the switching period[4]

In each modulation cycle (triggered by the PWMSYNC interrupt service routine) the DSP must compute the new on-time value to write to the three duty-cycle registers of the PWM generation unit. From, Figure 8, it can be seen that this reduces to the problem of computing the new value of $T_{on,A}$ in each cycle based on a new sample of the reference voltage, v_{refA}. In the figure, it is assumed that the reference voltage is represented as a signed, twos-complement number (1.15 format), so that a representation of 1.0 represents the maximum peak phase voltage applied to the motor.

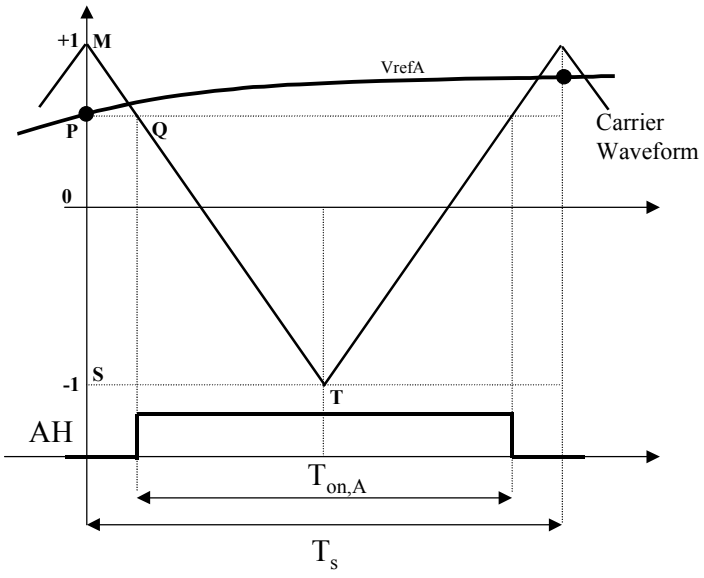

Figure 8: Expanded View of Generation of PWM signal for one Switching Period.

[4] see **ANF32X-02: Double Update Mode of PWM Generation Unit of ADMCF32X** for a description of the differences between the two operating modes

From Figure 8 it can be seen that the two triangles ΔMPQ and ΔMST are similar triangles. From this relationship, it is possible to state that the corresponding lengths of the two triangles satisfy the relation:

$$\frac{|PQ|}{|ST|} = \frac{|PM|}{|SM|} \tag{17}$$

so that:

$$\frac{\frac{1}{2}(T_s - T_{on,A})}{\frac{1}{2}T_s} = \frac{1 - v_{ref,A}}{2} \tag{18}$$

This expression can be solved for $T_{on,A}$ to yield:

$$T_{on,A} = \frac{T_s}{2}(1 + v_{refA}) \tag{19}$$

so that the value written to the PWMCHA register can be computed as:

$$PWMCHA = \frac{PWMTM}{2}(1 + v_{refA}) \tag{20}$$

From (20) it can be seen that if v_{refA} is set to 0.0, then the duty cycle register is written with a value equal to half the period, corresponding to 50% duty cycle. Therefore, the low and high side switches spend equal amounts of time connected to the positive and negative dc rails so that the average output voltage is 0, as required. Similarly, a value of v_{refA} of 1.0 gives a duty cycle value of PWMCHA=PWMTM, corresponding to the fully ON state (i.e. maximum positive voltage). Finally, a value of v_{refA} of -1.0 gives a duty cycle value of PWMCHA = 0, corresponding to the fully OFF state (i.e. maximum negative voltage). For any value of v_{refA} in the range –1.0 to 1.0, the duty cycle register is loaded with the appropriate value to provide the required average output voltage for the particular cycle.

The library routines will execute the above mentioned calculations through function-like macro calls.

3 The PWMF32X Library Routines

3.1 Using the PWM Unit Routines

The library provides some functions that configure the PWM generation unit in Single Update Mode and calculate and update register values for desired duty-cycles. The routines are developed as an easy-to-use library, which has to be linked to the user's application. The library consists of two files. The file "pwmF32X.dsp" contains the assembly code for the subroutines. This package has to be compiled and can then be linked to an application. The user simply has to include the header file "pwmF32X.h", which provides function-like calls to the routines. The following table summarises the set of macros that are defined in this library.

Operation	***Usage***
Configuration	PWM_Init(PWMsync_ISR, PWMtrip_ISR);
Update Duty Cycles	PWM_update_dutycycles(ChannelA, ChannelB, ChannelC);
Update Voltage	PWM_update_demanded_Voltage(ChannelA, ChannelB, ChannelC);

Table 1: Implemented routines for the PWM Block

The routines require some configuration constants, which are declared in a dedicated section of the main include-file "main.h" that comes with every application note. For more information about the general structure of the application notes and including libraries into user applications refer to the Library Documentation File. Section 4 shows an example of usage of this library. In the following sections each routine is explained in detail with the relevant segments of code which is found in either "pwmF32X.h" or "pwmF32X.dsp". For more information (e.g. about register use) see the comments in those files.

3.2 Configuring the PWM Block in Single Update Mode: PWM_Init

This macro initialises the PWM Block for operation in Single Update Mode. It expects some constants defined in "main.h". As may be seen in the following code segment, the frequency of the crystal, fundamental PWM switching frequency, deadtime, minimum pulse time and width of the PWM-Sync pulse are all defined as physical values in frequency or time units.

```
{*****************************************************************************
*                                                                            *
* Constants that need to be defined in main.h:                               *
*                                                                            *
* .CONST  Cry_clock        = xxxx;        Crystal clock frequency [kHz]      *
* .CONST  PWM_freq         = xxxx;        Desired PWM switching frequency [Hz]*
* .CONST  PWM_deadtime     = xxxx;        Desired deadtime [nsec]            *
* .CONST  PWM_minpulse     = xxxx;        Desired minimal pulse time [nsec]  *
* .CONST  PWM_syncpulse    = xxxx;        Desired sync pulse time [nsec]     *
*****************************************************************************}
```

Starting from these values, the actual set-up values for the configurations registers (i.e. PWMTM, PWMDT, etc.) are calculated. It is easy to see how the equations of Sections 1.4 and 1.5 are implemented here.

```
{*****************************************************************************
* Calculate Configuration Register Contents from Parameters                  *
*****************************************************************************}

.CONST  PWM_period_reg   = 1000*Cry_clock/PWM_freq;
                                        {PWMTM register value:
                                        Compute the divisor to be
                                        used for PWM switching period}

.CONST  PWM_deadtime_reg = Cry_clock*PWM_deadtime/1000000;
                                        {PWMDT register value:
                                        Compute the value to be used
                                        for Deadtime generation      }

.CONST  PWM_pulsedel_reg = 2*Cry_clock*PWM_minpulse/1000000;
                                        {PWMPD register value:
                                        Compute the value to be used
                                        for pulse deletion           }

.CONST  PWM_SYNCWT_reg   = 2*Cry_clock*PWM_syncpulse/1000000-1;
                                        {PWMSYNCWT register value:
                                        Compute the value to be used
                                        for sync pulse generation    }

.CONST  PWM_seg_reg    = 0x000;         { Enable all PWM outputs             }
.CONST  PWM_gate_reg   = 0x000;         { Disable high frequency chopping    }
```

The initialisation routine PWM_Init_ writes the above mentioned values into the corresponding registers (the macros Write_DM) are found in the general-purpose macro file "macro.h"). It then initialises the duty cycles of all channels to 50%, producing a zero average voltage on all three inverter legs. See the next section for this macro call.

```
PWM_Init_:
    NOP;
```

```
        Write_DM(PWMTM,  PWM_period_reg);  { Load period into the PWMTM - register      }
        Write_DM(PWMDT,  PWM_deadtime_reg);{ Load deadtime into the PWMDT - register    }
        Write_DM(PWMPD,  PWM_pulsedel_reg);{ Load pulsedel into the PWMPD - register    }
        Write_DM(PWMSYNCWT,  PWM_SYNCWT_reg);  { Load syncwt into the PWMSYNCWT - register  }
        Write_DM(PWMSEG, PWM_seg_reg);          { Load seg_value into the PWMSEG - register  }
        Write_DM(PWMGATE,PWM_gate_reg);         { Load gate_value into the PWMGATE - register}
        PWM_update_dutycycles(0x4000, 0x4000, 0x4000);
                                    { Set PWM duty cycle registers to 50%}
        rts;
```

The macro call of PWM_Init now puts everything together: it first calls the above mentioned initialisation routine, then sets the interrupt vectors for both Sync ISR and the Trip ISR (the entry labels of both service routines are passed as parameters, The macro is defined in macro.h and makes use of the Utility Put_Vector[5]). At last, the two interrupts are enabled by setting the corresponding mask bits in the MODECTRL register. At this point, the block is configured. The only thing that the user has to do is enabling the common peripheral interrupt line IRQ2 to the DSP core. Section 4 will show a complete example of how this is done.

```
        .MACRO PWM_Init(%0, %1);

                call PWM_Init_;
                NOP;
                Set_InterruptVector(PWMSYNC_INT_ADDR, %0); { Vector for PWM_SYNC int    }
                Set_InterruptVector(PWMTRIP_INT_ADDR, %1); { Vector for PWM_SYNC int    }

                ay0 = 0x000c;                           {Enable PWMSYNC & PWMTRIP    }
                ar = DM(MODECTRL);                      {interrupts from MODECTRL    }
                ar = ar or ay0;                         {of ADMCF32X. Preserve other  }
                DM(MODECTRL) = ar;                      {bits of MODECTRL by ORing   }
        .ENDMACRO;
```

3.3 Updating the Duty-Cycles: PWM_update_dutycycles

This macro provides an easy means of imposing duty-cycles in a more natural manner. The input arguments are the desired duty-cycles of the three inverter legs, expressed as 1.15 format values in the range of 0 to 1 (0x0000 to 0x7FFF). Those values are then converted into register values for PWMCHA, PWMCHB and PWMCHC. Since +1 corresponds to having a register value equal to PWMTM, this is achieved by simply multiplying the desired duty-cycle for each channel by the content of PWMTM.

```
        .MACRO PWM_update_dutycycles(%0, %1, %2);
                sr0= DM(PWMTM);
                my0 = %0; mr= sr0 * my0 (SS); dm(PWMCHA) = mr1;
                my0 = %1; mr= sr0 * my0 (SS); dm(PWMCHB) = mr1;
                my0 = %2; mr= sr0 * my0 (SS); dm(PWMCHC) = mr1;
        .ENDMACRO;
```

3.4 Updating the Duty-Cycles: PWM_update_demanded_voltage

This macro also is intended for imposing duty-cycles. However, unlike the previous macro, the input arguments are the desired average voltages of the three inverter legs, expressed as 1.15 format values in the range of -1 to 1 (0x8000 to 0x7FFF). Those values are then converted into register values for PWMCHA, PWMCHB and PWMCHC as shown in Section 2.2. This time, equation (20) is evaluated for each channel as

$$\mathrm{PWMCHx} = \frac{\mathrm{PWMTM}}{2} + \frac{\mathrm{PWMTM}}{2} \cdot \mathrm{v}_{\mathrm{refx}} \qquad (21)$$

[5] For more information on the PUT_VECTOR routine see "ADMCF32X - Developer's Reference Manual"

```
.MACRO PWM_update_demanded_Voltage(%0, %1, %2);
       sr0= DM(PWMTM);
       sr = LSHIFT sr0 by -1 (LO);          { PWMTM/2 in sr0                          }
       mr =0; mr1= sr0; my0 = %0; mr= mr + sr0 * my0 (SS); dm(PWMCHA) = mr1;
       mr =0; mr1= sr0; my0 = %1; mr= mr + sr0 * my0 (SS); dm(PWMCHB) = mr1;
       mr =0; mr1= sr0; my0 = %2; mr= mr + sr0 * my0 (SS); dm(PWMCHC) = mr1;
.ENDMACRO;
```

4 Software Example: Generation of Three-Phase Sine-Waves

As already seen sine wave PWM is commonly used for the general parts of motor-systems. Three phase systems, requires 6 independent PWM signals which are precisely timed and controlled relative to some reference waveform. The example beneath illustrates how three sine-wave reference curves are generated and how the PWM-block is controlled. The program accepts a commanded frequency demand that is read from the data memory location AD_IN. The default value is 1.0 (0x7FFF). However, this value can be adjusted by writing a new value to this data memory location (from the Data Memory window of the Motion Control Debugger). The example software adjusts the voltage amplitudes accordingly, in order to obtain a constant Volt/Hertz ratio.

4.1 Programming of Maximum Output Frequency

The program uses the trigonometric function Sine[6] to produce the reference sine waves. The phase argument is scaled such that the [-π, +π] interval is mapped into the 1.15 format. Essentially, each time the PWMSync ISR is serviced (every 100μsec at 10kHz switching frequency), the phase angle is incremented by an amount that depends on the desired fundamental frequency (Variable **AD_IN**) and the maximum output frequency (constant **delta**).

The maximum fundamental output frequency is achieved when the input demand (**AD_IN**) is set to 0x7FFF, corresponding to +1.0 in the 1.15 number format. The constant **delta** is used to select the actual maximum frequency that is implemented. This value is used in the computation of the angles used to define the three reference voltages. The relationship between the maximum fundamental frequency, $f_{1,max}$ and the value of **delta** is given by:

$$f_{1,max} = f_s \frac{\text{delta}}{2^{16}} \tag{22}$$

where f_s is the PWM switching frequency. Therefore, for the present example, with f_s = 10 kHz and delta = 0x400 = 1024, the maximum frequency is 156 Hz.

4.2 The main program: main.dsp

The file "main.dsp" contains the initialisation and PWM Sync and Trip interrupt service routines. To activate, build the executable file using the attached **build.bat** either within your DOS prompt or clicking on it from Windows Explorer. This will create the object files and the **main.exe** example file. This file may be run on the Motion Control Debugger. The main program is for debugging placed in Program RAM. When the program is ready for standalone operation (from Flash) the start location is moved from ABS=0x30 to ABS=0X2200. (See Reference Manual). Every module besides from the Main_program module is by default placed in either one of the three USERFLASH memory banks.

[6] Refer to ANF32X-10 Implementation of Basic Trigonometric Functions

In the following, a brief description of the code is given.

Start of code – declaring start location in program memory or FLASH memory. Comments are placed depending on whether the program should run in PMRAM or Flash memory.

```
{*****************************************************************************
* Application: Starting from FLASH        (out-comment the one not used)
*****************************************************************************}

!.MODULE/RAM/SEG=USERFLASH1/ABS=0x2200    Main_Program;

{*****************************************************************************
* Application: Starting from RAM   (out-comment the one not used)
*****************************************************************************}
.MODULE/RAM/SEG=USER_PM1/ABS=0x30         Main_Program;
```

*Next, the general systems constants and PWM configuration constants (main.h – see the next section) are included. Also included are the trigonometric library for sine calculation and the **PWM library**.*

```
{*****************************************************************************
* Include General System Parameters and Libraries                            *
*****************************************************************************}

#include <main.h>;

#include <pwmF32X.h>;
#include <trigono.h>;
```

Two constants are defined. Delta determines the maximum output frequency as seen in section 4.1. The hexadecimal equivalent in 1.15 format of 120° is called TwoPiOverThree.

```
{*****************************************************************************
* Constants Defined in the Module                                            *
*****************************************************************************}

.CONST Delta               = 0x0400;                  { Angle increment 64 pr. rev }
.CONST TwoPiOverThree      = 0xffff / 3;              { Hex equivalent of  2pi/3   }
```

Some Variables are defined hereafter. AD_IN contains the desired frequency, Theta is the current phase angle and Vrefx is the computed average voltage for phase x. . For debugging in the FLASH the STARTUP is declared as .ENTRY (See Reference Manual).

```
{*****************************************************************************
* Local Variables Defined in this Module                                     *
*****************************************************************************}

.VAR/DM/RAM/SEG=USER_DM AD_IN;                  { Volts/Hertz Command (0-1)          }
.VAR/DM/RAM/SEG=USER_DM Theta;                  { Current angle                      }
.VAR/DM/RAM/SEG=USER_DM VrefA;                  { Voltage demands                    }
.VAR/DM/RAM/SEG=USER_DM VrefB;
.VAR/DM/RAM/SEG=USER_DM VrefC;

.ENTRY STARTUP;                                 { For Debugging                    }
```

*The first thing that is done in the initialisation block (Startup) is checking a selected PIO line for level. If the PIO-pin is high jump to an ERASE BOOT FROM FLASH BIT routine in ROM and return. If not, just go ahead with normal operation. This small macro is done to enable re-coding of the FLASH memory. For further information (See Reference Manual). In this example the PIO-pin 6 is chosen as erase pin.The **initialisation of the PWM block** is executed. Note how the interrupt vectors for the PWMSync and PWMTrip service routines are passed as arguments. Then the interrupt IRQ2 is enabled by setting the corresponding bit in the IMASK register. Finally the variables are initialised with start-up values for the code. After that, the program enters a loop, which just waits for interrupts*

```
{*****************************************************************************}
{ Start of program code                                                       }
{*****************************************************************************}
Startup:
     FLASH_erase_PIO(6);           { Select PIO6 as clearing PIO              }
                                   { Remember that sport1 is muxed with the PIO-lines  }
```

```
                                { If the bit is high Boot from Flash bit            }

     PWM_Init(PWMSYNC_ISR, PWMTRIP_ISR);

     IFC = 0x80;                { Clear any pending IRQ2 inter.                     }
     ay0 = 0x200;               { unmask irq2 interrupts.                           }
     ar = IMASK;
     ar = ar or ay0;
     IMASK = ar;                { IRQ2 ints fully enabled here                      }

     ar = 0x7FFF;   dm(AD_IN) = ar;
     ar = 0x0000;   dm(Theta) = ar; dm(VrefA) = ar; dm(VrefB) = ar; dm(VrefC) = ar;

Main:                           { Wait for interrupt to occur                       }
     jump Main;
     rts;
```

*The interrupt service routine is executed at the PWM switching rate (10kHz in this example).Here the actual duty cylces are determined for each PWM period. First, the desired frequency is read from the command variable AD_IN. Then, the phase angle Theta is increased by the amount Delta*AD_IN and stored for the next interrupt. This phase angle is now used to compute Sin(Theta) and VrefA= AD_IN*Sin(Theta). Similarly, phase B is computed as AD_IN*Sin(Theta-TwoPiOverThree) and phase B as AD_IN*Sin(Theta+TwoPiOverThree) so that as three-phase system with phase displacements of 120° is obtained. This values may then be directly converted into the duty-cycle register values by calling the library routine* ***PWM_update_demanded_Voltage.***

```
{*****************************************************************************}
{ PWM Interrupt Service Routine                                               }
{*****************************************************************************}

PWMSYNC_ISR:

     Set_DAG_registers_for_trigonometric;
            my0 = DM(AD_IN);

            mr = 0;                     {Clear mr                               }
            mr1 = dm(Theta);            {Preload Theta                          }
            mx0 = Delta;
            mr = mr + mx0*my0 (SS);     {Compute new angle & store              }
            dm(Theta) = mr1;

            Sin(mr1);                   { Result in ar register                 }
            mr = ar*my0 (SS);           { Multiply by AD_IN for VrefA           }
            dm(VrefA) = mr1;

            ax1 = dm(Theta);            { Compute angle of phase B              }
            ay1 = TwoPioverThree;
            ar = ax1 - ay1;
            Sin(ar);                    { Result in ar register                 }
            mr = ar*my0 (SS);           { Multiply by AD_IN for VrefA           }
            dm(VrefB) = mr1;

             ax1 = dm(Theta);           { Compute angle of phase C          }
             ay1 = TwoPioverThree;
             ar = ax1 + ay1;
            Sin(ar);                    { Result in ar register             }
             mr = ar*my0 (SS);          { Multiply by AD_IN for VrefA       }
             dm(VrefC) = mr1;


     ax0 = DM(VrefA); ax1 = DM(VrefB); ay0 = DM(VrefC);
     PWM_update_demanded_Voltage(ax0,ax1,ay0);

RTI;
```

The PWM Trip Interrupt Service Routine is shown here. In this example it consists simply of a nop instruction, so that no action is taken in case of a PWM fault.

```
{*****************************************************************************}
{ PWM Trip Interrupt Service Routine                                          }
```

```
{*****************************************************************************}

PWMTRIP_ISR:
     nop;
     rti;

.ENDMOD;
```

4.3 The main include file: main.h

This file contains the definitions of ADMCF32X constants, general-purpose macros and the configuration parameters of the system and library routines. It should be included in every application. For more information refer to the Library Documentation File.

This file is mostly self-explaining. The relevant sections to this example are shown here. The frequency of the used crystal (10 MHz in case of the ADMCF32X Evaluation Kit) is expressed in kHz. Then ADMCF32X specific constants, PUT_VECTOR and FLASH_CONTROL (Erasing BootFromFlash bit) and general-purpose macros are included. Refer to Developers Reference Manual on the ADMCF32X.

```
{*****************************************************************************
* General System Parameters and Constants                                    *
*****************************************************************************}

.CONST  Cry_clock      = 10000;    {Crystal clock frequency [kHz]              }

#include <ADMCF32X.h>;
#include <putvectr.h>;             { Put_vector function                       }
#include <Flash_CT.h>;             { Special Control unit for the Flash        }
#include <macro.h>;
```

As described in the Library Documentation File, every library routine has a section in main.h for its configuration parameters. The following defines the parameters for the PWM block used in this example.

```
{*****************************************************************************}
{ Library: PWM block                                                          }
{ file   : PWMF32X.dsp                                                        }
{ Application Note: Usage of the ADMCF32X Pulse Width Modulation Block        }
.CONST  PWM_freq         = 10000;         {Desired PWM switching frequency [Hz]   }
.CONST  PWM_deadtime     = 1000;          {Desired deadtime [nsec]              }
.CONST  PWM_minpulse     = 1000;          {Desired minimal pulse time [nsec]      }
.CONST  PWM_syncpulse    = 1540;          {Desired sync pulse time [nsec]         }

{*****************************************************************************}
```

5 Experimental Results of Sinusoidal PWM.

Figure 9, Figure 10 and Figure 11 show some measured waveforms when this sine-wave PWM example software is run on the ADMCF32X Evaluation Board with a 10 MHz CLKIN. Figure 9 shows the three high-side PWM signals (AH, BH and CH) after they have been filtered on the oscilloscope. This filtering removes the high-frequency components due to the PWM process so that the fundamental output behavior is seen. Clearly, the three PWM signals produce a balanced three-phase system at the desired maximum output frequency of 156 Hz. Figure 10 and Figure 11 show instantaneous views of the PWM outputs where the modulation process is clearly seen by the changing of the duty cycles values from one PWM cycle to the next.

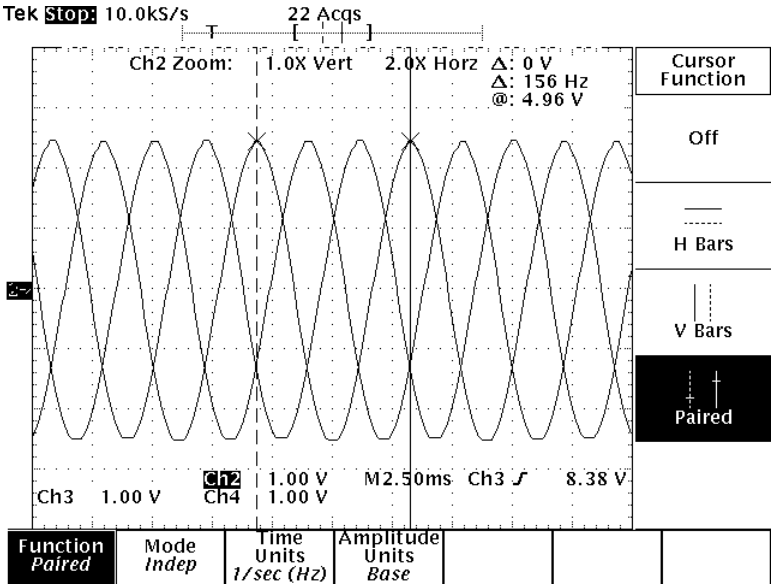

Figure 9: Measured Filtered PWM outputs with Sine PWM Scheme (AD_IN = 0x7FFF).

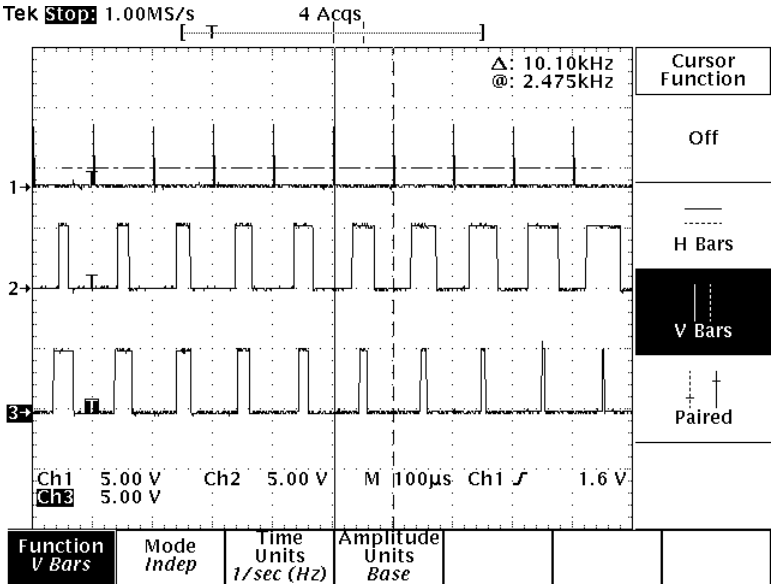

Figure 10: PWM Outputs (AD_IN = 0x7FFF), TOP: PWMSYNC, MIDDLE: AH, BOTTOM: BH.

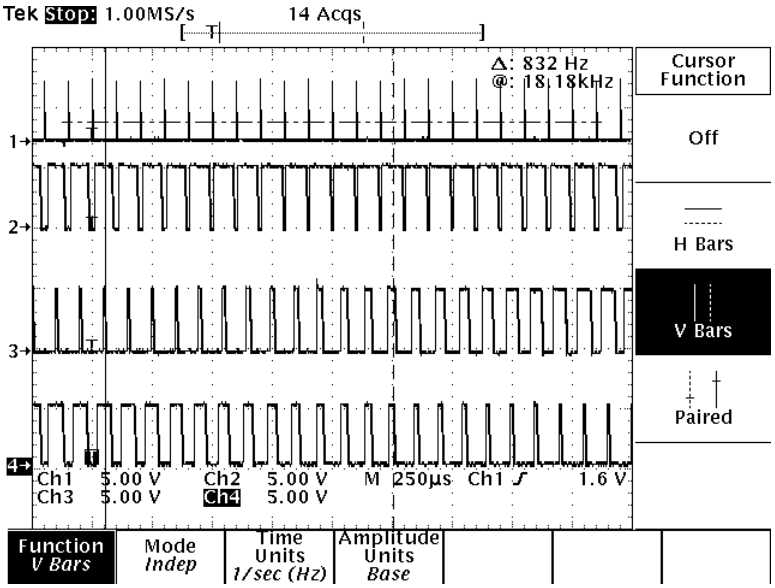

Figure 11: Measured PWM outputs (AD_IN = 0x6666), CH1: PWMSYNC, CH2: AH, CH3: BH, CH4: CH.