Digital control of inverters

Introduction

Three phase inverter control signals are commonly produced by comparing sinusoids (modulating waves) with a triangular or sawtooth (carrier waves) waveform. In the past, these waveforms were produced using analogue oscillators and analogue comparators. A digital approach is used these days whereby numeric values for the modulating waves are generated either online of offline. These values are compared using software and/or hardware resulting in sinusoidally varying output pulses on output port pins of a microcontroller (see Figure 1). These pulses are then optically isolated, amplified and passed on to the power devices in the inverter.

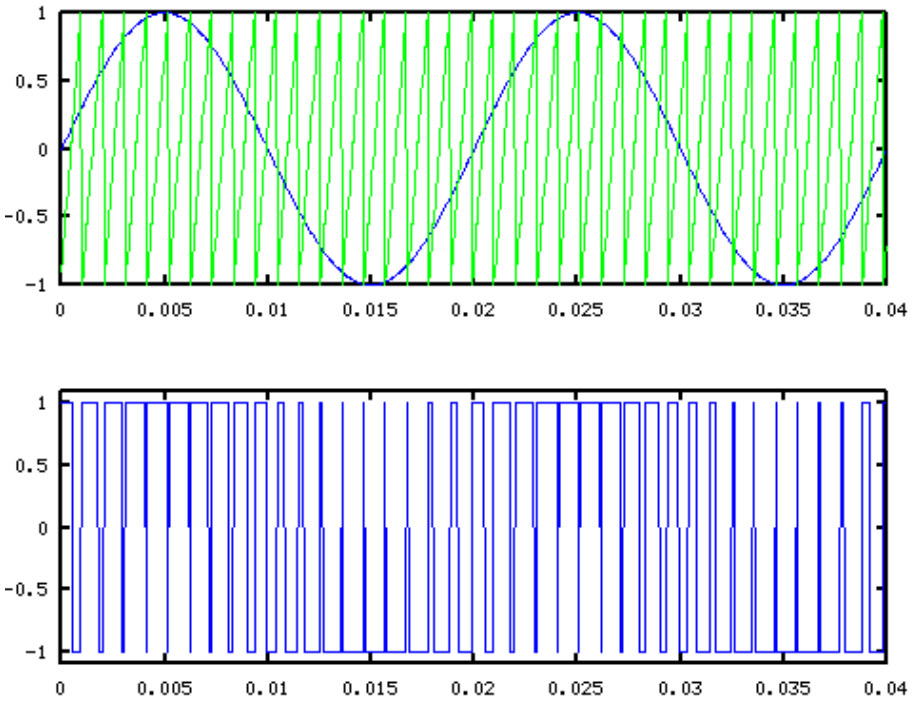

Figure 1. Sinusoidal PWM. Frequency modulation ratio=20, amplitude modulation ratio=1. The carrier (green) is compared to the modulating sine wave (blue). When the modulating wave is greater, the output is high (lower trace). The matlab/octave m-file used to generate this is available in the webcourses site for this module.

Digital timer systems

It is possible to generate SPWM signals using only software running on a microcontroller. While this technique works, it is quite wasteful of the microcontroller's cpu time and may be subject to jitter as interrupts etc. may take place while the system is running. The more usual approach taken now is to use a hardware timer to generate the PWM signals. The system operates as shown in Figure 2. A high speed clock signal (Fc) drives a free running “UP” counter. The counter value is compared to values in two counter compare registers (CCR1, CCR2). When a match occurs with CCR1, an output bit is set (high) and the free running counter is cleared/reset. An interrupt is also generated. When a match occurs with CCR2, the output bit is cleared. The net effect is that an output signal is generated with frequency and duty cycle given by the following:

$$Output\ Frequency = \frac{Fc}{CCR1}$$

$$Output\ Duty = \frac{CCR2}{CCR1}$$

In order to change the duty cycle, the CPU must simply write a new value to CCR2.

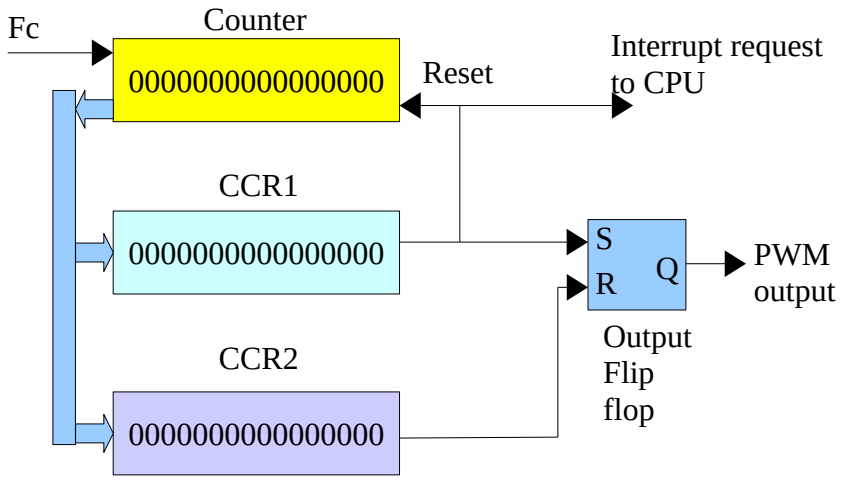

Figure 2. A hardware timer used for PWM signal generation

Fixed frequency and voltage inversion

Using the timing system shown in Figure 2 it is possible to implement a very simple inverter controller. The set of CCR2 values for a single cycle of the fundamental output frequency can be pre-computed and stored in a table in ROM in the microcontroller system. At each interrupt, a value is copied from the table and placed in CCR2. When all the values are exhausted, the control program simply moves back to the start of the table and repeats the process. A very low cost microcontroller could be used to implement such a system however it will be unable to control the output frequency / voltage if only a single set of values is used. A more complex solution might involve multiple tables of CCR2 values – one for each of the supported output frequencies. A three phase output can be generated if three timers are used – a single table of CCR2 values should be sufficient as the other output phases are simply time shifted versions of the first.

Variable frequency and voltage inversion

In the more general case, an inverter will allow independent adjustment of output voltage and frequency (perhaps as part of the control loop for some larger process). In order to do this, the control program will need to calculate new values for the three modulating waves at each interrupt generated by CCR1 (i.e. at the carrier frequency rate). This can be computationally challenging. Consider an inverter operating at a switching frequency of 20kHz. The control program will be required to calculate three sine values at each interrupt i.e. every 50µs as well as a number of multiplies and additions. A microcontroller operating at several 10's of MHz will typically be required in such situations along with an efficient 'sine' algorithm.

Outline implementation (pseudo code)

```
Initialize()
{
        Fm  = Carrier Frequency / Modulation Frequency;
        CCRA1 = Clock Frequency  / Carrier Frequency;
        CCRB1 = Clock Frequency  / Carrier Frequency;
        CCRC1 = Clock Frequency  / Carrier Frequency;
        AngleA = 0;
        AngleB = 120;
        AngleC = 240;
        AngleStepSize = 360 / Fm;
        CarrierAmplitude=( Clock Frequency / Carrier Frequency ) / 2;
        ModulatingAmplitude =CarrierAmplitude * Am;
}
InterruptHandler()
{
        AngleA = AngleA + AngleStepSize;
        if (AngleA >=360)
                AngleA = 0;
        AngleB = AngleB + AngleStepSize;
        if (AngleB >=360)
                AngleB = 0;

        AngleC = AngleC + AngleStepSize;
        if (AngleC >=360)
                AngleC = 0;
        CCRA2 = ModulatingAmplitude*(sin(AngleA)+1)/2;
        CCRB2 = ModulatingAmplitude*(sin(AngleB)+1)/2;
        CCRC2 = ModulatingAmplitude*(sin(AngleC)+1)/2;
}
```

Specialist microcontrollers

Some microcontrollers have additional features built in to their timing systems that assist inverter control algorithms. These features include

Dead time: It may be possible to insert a dead time between one output going low and another going high to allow the transistor in one arm of an inverter bridge fully turn off before its partner is turned on – this prevents “shoot-through” which occurs when both transistors are partially on at the same time.

Minimum pulse width: A finite amount of time is required to turn a transistor on and off again. If you attempt to drive it any faster you will simply incur a needless power loss in the device. Some hardware counters allow the programmer specify the minimum permissible output pulse width (by putting a value in a register). If an attempt is made to generate a pulse width shorter than this then the pulse is simply “dropped” and the output will stay low.