PWM Sine Wave Generation

Also known as Direct to Digital Synthesis (DDS)

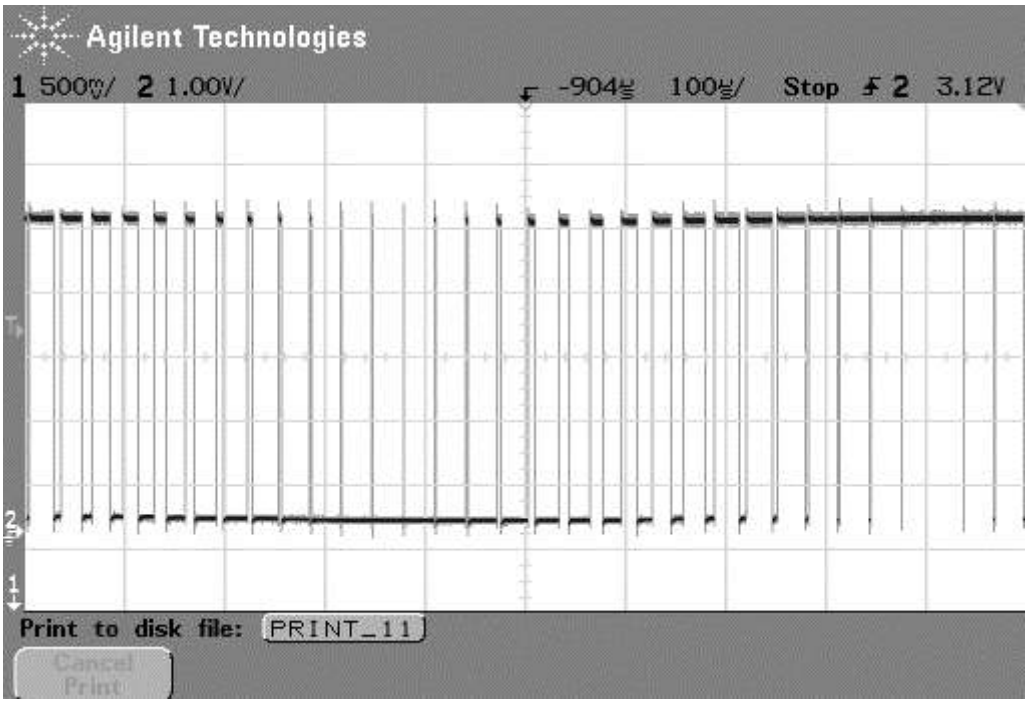
References:
http://www.atmel.com/dyn/resources/prod_documents/doc2542.pdf - AVR flowchart with best overview write-up. The PWM mode is selected so the OC1A pin toggles on compare match and the Top value of the timer is set to 0xFF. The Top value affects the resolution and the base frequency of the PWM – the higher the Top value is the higher resolution and the lower base frequency.
http://www.analog.com/library/analogdialogue/archives/38-08/dds.html - Analog Devices introduction to Direct Digital Synthesis Theory and DDS ICs
http://www.evilmadscientist.com/article.php/avrdac - ATmega168 detailed how-to tutorial with code for generating periodic waveforms. Tutorial stops after generating rectangular and triangle waveforms. Sine waves are left to the advanced student. Waveform Generation Mode (WGM) set to 0b1110 - *Fast PWM*
http://www.arduino.cc/playground/Code/PCMAudio - ATmega168 code programmed in Arduino IDE. Code is designed to play sound files up to 10000 bytes long at a sample rate of 8 ksps (i.e, 1.25 second sound effect). WGM = 0b1110 - *Fast PWM*
Generate sine wave modulated PWM with AVR microcontroller - ATmega8 code programmed in AVRStudio IDE. ATmega8 code. PWM = 0b0001 *Phase Correct*.
AVR DDS signal generator V1.0 - This is the documentation schematic, pcb, code for for an ATmega8 based waveform generator
http://documentation.renesas.com/eng/products/region/rtas/mpumcu/apn/sinewave.pdf - Nice tutorial based on Renesas H8/38024F
http://ww1.microchip.com/downloads/en/AppNotes/00655a.pdf - PIC16C620 tutorial with a two pole filter to pass the 60 Hz signal while eliminating the step (1920 Hz) and the PWM (19 khz) frequencies.

This appears to be a variant on DDS known as magic sine wave
http://www.sxlist.com/techref/io/pwm/harmonic.htm
http://www.tinaja.com/magsn01.asp

Table of Contents

Introduction

PWM combined with an analog filter can be used to generate analog output signals, i.e. a digital to analog converter (DAC). A digital pulse train with a constant period (fixed ***base frequency***) is used as a basis. To generate different analog levels, the duty cycle and thereby the pulse width of the digital signal is changed. If a high analog level is needed, the pulse width is increased and vice versa.



Base Frequency

In the AVR, the timer/counters are used to generate PWM signals. To change the PWM base frequency, the ***timer clock frequency*** and ***top counter*** value is changed. Faster clock (f_{clk_i/o_max} = 16 MHz) and/or lower top value will increase the PWM base frequency, or timer overflow frequency. With full resolution (top value 255) the maximum PWM base frequency is 250 kHz. Increasing the base frequency beyond this frequency will be at the expense of reduced resolution, since fewer step are then available from 0% to 100% duty cycle.

Duty Cycle

Altering the value of the Output Compare Registers (OCR) changes the duty cycle. Increasing the OCR value increases the duty cycle. The PWM output is high until the OCR value is reached, and low until the timer reaches the top value and wraps back to 0. This is called Fast-PWM mode.

Design Trade-offs

The filter crossover frequency must be chosen high enough to not alter the analog signal of interest. At the same time it must be as low as possible to minimize the ripple from the PWM base frequency.

The step-size difference between the analog levels is dependent on the resolution of the PWM. The higher base frequency the more easily is it to attenuate the base frequency and thereby minimize the signal ripple. The selection of PWM resolution versus base frequency is thus an application dependent trade-off.

For an extreme example, the highest possible timer clock frequency for the Arduino Duemilanove ATtmega328P is 16 MHz (no prescaling). At 4 MHz (16MHz/4) PWM base frequency (top value 3) the OCR value can be set to 0, 1 (25% duty cycle), 2 (50% duty cycle, A in Figure 4) or 3 (100% duty cycle). This shows that lowering the top value to increase the PWM base frequency reduces the resolution.

Program Example

Initialization

To be able to generate an output from the PWM, the Output Compare Pin of Timer0 (OC0A) is set up as output. This is Digital Pin 6 (Port D pin 6) on the

Next the Timer0 is set up: The clock source for the timer is prepared – the PLL is started and locked to the system clock (required). The PLL takes approximately 100 ms to lock onto the system clock and it is therefore necessary to wait for the PLL lock flag before proceeding. Once the PLL is locked it is selected as clock source for the timer.

The PWM mode is then selected so that the OC0A pin toggles on compare match and the Top value of the timer is set to 0xFF. The Top value affects the resolution and the base frequency of the PWM – the higher the Top value is the higher resolution and the lower base frequency.

The Timer is now ready to be started: The prescaler is set, which also starts the timer. Finally, the Overflow interrupt is enabled.

ISR

When the Timer0 value reaches the OCR0B value (0xFF), the Timer Overflow interrupt service routine (ISR) is executed. This happens at a constant interval, since OCR0B is constant. This interval is the base frequency of the fast PWM output signal.

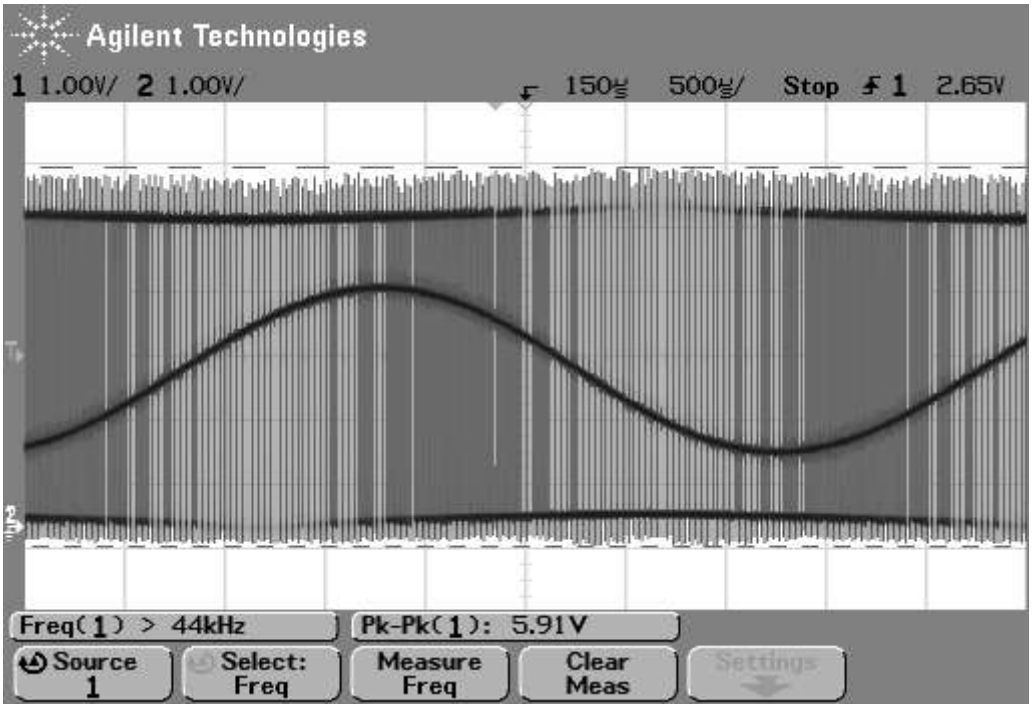
In the Timer0 Overflow ISR, a look up in a sine table is made. On each look-up the index to the look-up table is incremented so that new values can be loaded. The value from the sine table is written to OCR0A. In this way the pulse width is modulated to the sine wave. Note that the OCR0A register is buffered and that the latching of the buffer into the actual OCR0A register takes place on the timer overflow.

The interrupt routine takes 13 clock cycles to execute. The call and return to and from the interrupt comes in addition – in total 21 system clock cycles. Since Timer0 is an 8-bit timer

the interrupt occurs every 256/(PWM_clock/system_clock) cycle. The example is based on that the device is clocked from the internal RC oscillator, which is 16 MHz (prescaler = 1). 256/16 MHz = 16 usec

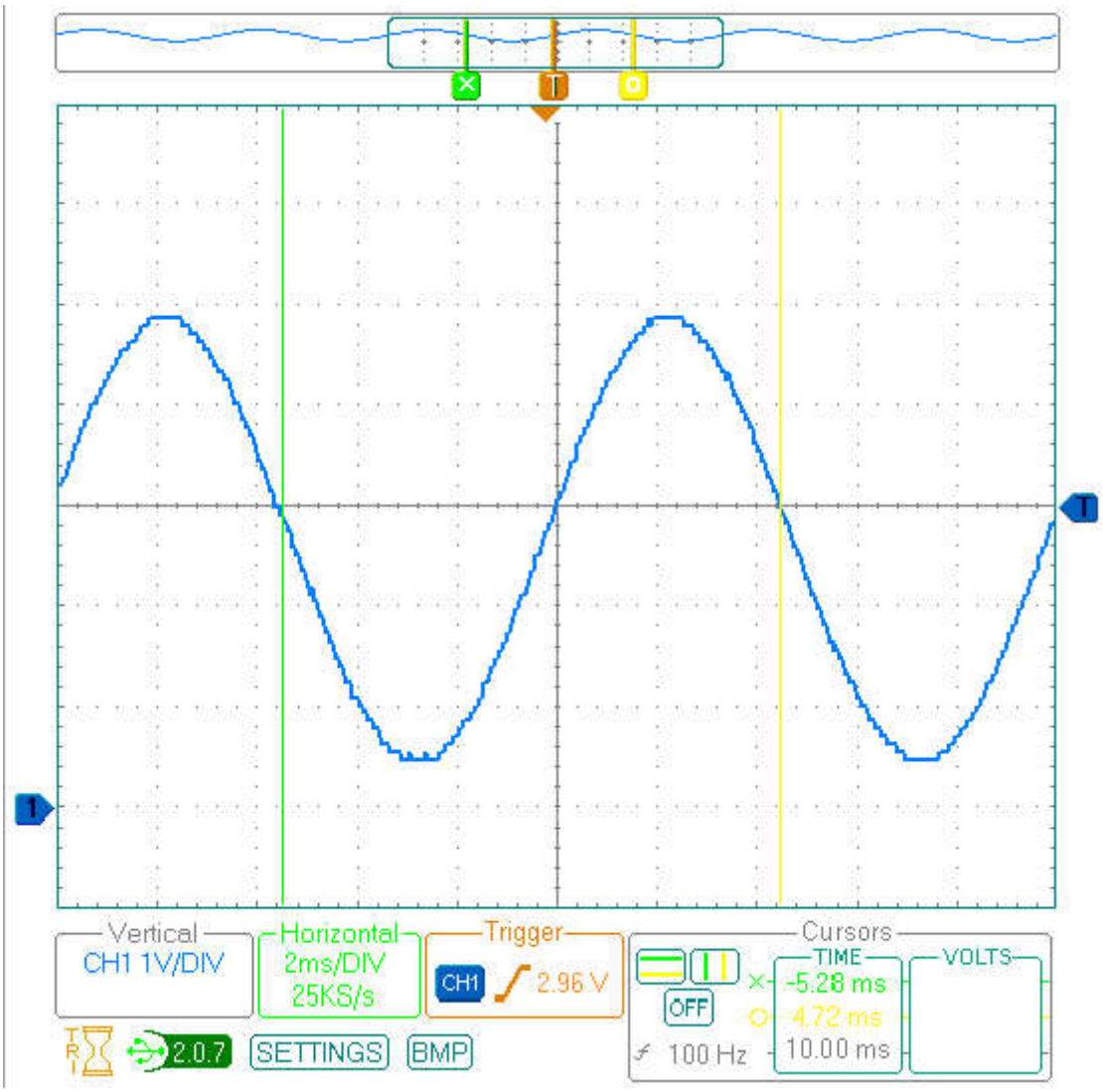
Example

The following scope pictures are examples of sine wave signals generated by the ATmega328P PWM. The scope snap-shots show the output on the OC0A pin, which is the digital pulse modulated signal, and the filtered/shaped PWM signal. A simple RC filter is used to shape the PWM signal to a sine wave – an analog signal where the amplitude is given by the duty cycle of the PWM output. The RC filter used has an R = 10 kΩ and a C = 100 nF, resulting in a filter crossover frequency of 1 kHz, which will let the low frequency sine wave pass while filtering out the high frequency PWM base.



ATmega328P Arduino Sine Wave Code - Version 1.0

The following scope picture is an example of a 100Hz sine wave signal generated by the ATmega328P PWM. The scope snap-shot show the output on the OC0A pin (Arduino Digital Pin 6), which is the filtered/shaped PWM signal. A simple RC filter is used to shape the PWM signal to a sine wave – an analog signal where the amplitude is given by the duty cycle of the PWM output. The RC filter used has an R = 10 kΩ and a C = 100 nF, resulting in a filter crossover frequency of 1 kHz, which will let the low frequency sine wave pass while filtering out the high frequency PWM base (62.5 kHz).



Arduino Main Program

```
/*
 * sinewave_pcm
 *
 * Generates 8-bit PCM sinewave on pin 6 using pulse-width modulation (PWM).
 * For Arduino with Atmega368P at 16 MHz.
 *
 * Uses timers 1 and 0. Timer 1 reads the sinewave table, SAMPLE_RATE times a
second.
 * The sinewave table has 256 entries. Consequently, the sinewave has a
frequency of
 * f = SAMPLE_RATE / 256
 * Each entry in the sinewave table defines the duty-cycle of Timer 0. Timer
0
```

```
 * holds pin 6 high from 0 to 255 ticks out of a 256-tick cycle, depending on
 * the current duty cycle. Timer 0 repeats 62500 times per second (16000000 /
256),
 * much faster than the generated sinewave generated frequency.
 *
 * References:
 * http://www.atmel.com/dyn/resources/prod_documents/doc2542.pdf
 * http://www.analog.com/library/analogdialogue/archives/38-08/dds.html
 * http://www.evilmadscientist.com/article.php/avrdac
 * http://www.arduino.cc/playground/Code/R2APCMAudio
 * http://www.scienceprog.com/generate-sine-wave-modulated-pwm-with-avr-
microcontroller/
 * http://www.scienceprog.com/avr-dds-signal-generator-v10/
 * http://documentation.renesas.com/eng/products/region/rtas/mpumcu/apn/
sinewave.pdf
 * http://ww1.microchip.com/downloads/en/AppNotes/00655a.pdf
 *
 * By Gary Hill
 * Adapted from a script by Michael Smith <michael@hurts.ca>
 */

#include <stdint.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/pgmspace.h>

#define SAMPLE_RATE 8000    // 8 ksps

/*
 * The sinewave data needs to be unsigned, 8-bit
 *
 * sinewavedata.h should look like this:
 * const int sinewave_length=256;
 *
 * const unsigned char sinewave_data[] PROGMEM = {0x80,0x83, ...
 *
 */

#include "sinewavedata.h"

int outputPin = 6; // (PCINT22/OC0A/AIN0)PD6, Arduino Digital Pin 6
volatile uint16_t sample;

// This is called at SAMPLE_RATE kHz to load the next sample.
ISR(TIMER1_COMPA_vect) {
    if (sample >= sinewave_length) {
      sample = -1;
    }
```

```
    else {
      OCR0A = pgm_read_byte(&sinewave_data[sample]);
    }
    ++sample;
}

void startPlayback()
{
    pinMode(outputPin, OUTPUT);

    // Set Timer 0 Fast PWM Mode (Section 14.7.3)
    // WGM = 0b011 = 3  (Table 14-8)
    // TOP = 0xFF, update OCR0A register at BOTTOM
    TCCR0A |= _BV(WGM01) | _BV(WGM00);
    TCCR0B &= ~_BV(WGM02);

    // Do non-inverting PWM on pin OC0A, arduino digital pin 6
    // COM0A = 0b10, clear OC0A pin on compare match,
    //               set 0C0A pin at BOTTOM (Table 14-3)
    TCCR0A = (TCCR0A | _BV(COM0A1)) & ~_BV(COM0A0);
    // COM0B = 0b00, OC0B disconnected (Table 14-6)
    TCCR0A &= ~(_BV(COM0B1) | _BV(COM0B0));

    // No prescaler, CS = 0b001 (Table 14-9)
    TCCR0B = (TCCR0B & ~(_BV(CS02) | _BV(CS01))) | _BV(CS00);

    // Set initial pulse width to the first sample.
    OCR0A = pgm_read_byte(&sinewave_data[0]);

    // Set up Timer 1 to send a sample every interrupt.
    cli();   // disable interrupts

    // Set CTC mode (Section 15.9.2 Clear Timer on Compare Match)
    // WGM = 0b0100, TOP = OCR1A, Update 0CR1A Immediate (Table 15-4)
    // Have to set OCR1A *after*, otherwise it gets reset to 0!
    TCCR1B = (TCCR1B & ~_BV(WGM13)) | _BV(WGM12);
    TCCR1A = TCCR1A & ~(_BV(WGM11) | _BV(WGM10));

    // No prescaler, CS = 0b001 (Table 15-5)
    TCCR1B = (TCCR1B & ~(_BV(CS12) | _BV(CS11))) | _BV(CS10);

    // Set the compare register (OCR1A).
    // OCR1A is a 16-bit register, so we have to do this with
    // interrupts disabled to be safe.
    OCR1A = F_CPU / SAMPLE_RATE;    // 16e6 / 8000 = 2000

    // Enable interrupt when TCNT1 == OCR1A (p.136)
    TIMSK1 |= _BV(OCIE1A);
```

```
    sample = 0;
    sei();  // enable interrupts
}


void setup()
{
    startPlayback();
}

void loop()
{
    while (true);
}
```

sinewavedata.h

```
/* Sinewave table
 * Reference:
 * http://www.scienceprog.com/generate-sine-wave-modulated-pwm-with-avr-
microcontroller/
 */

const int sinewave_length=256;

const unsigned char sinewave_data[] PROGMEM = {
0x80,0x83,0x86,0x89,0x8c,0x8f,0x92,0x95,0x98,0x9c,0x9f,0xa2,0xa5,0xa8,0xab,0xae,
0xb0,0xb3,0xb6,0xb9,0xbc,0xbf,0xc1,0xc4,0xc7,0xc9,0xcc,0xce,0xd1,0xd3,0xd5,0xd8,
0xda,0xdc,0xde,0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xed,0xef,0xf0,0xf2,0xf3,0xf5,
0xf6,0xf7,0xf8,0xf9,0xfa,0xfb,0xfc,0xfc,0xfd,0xfe,0xfe,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xfe,0xfe,0xfd,0xfc,0xfc,0xfb,0xfa,0xf9,0xf8,0xf7,
0xf6,0xf5,0xf3,0xf2,0xf0,0xef,0xed,0xec,0xea,0xe8,0xe6,0xe4,0xe2,0xe0,0xde,0xdc,
0xda,0xd8,0xd5,0xd3,0xd1,0xce,0xcc,0xc9,0xc7,0xc4,0xc1,0xbf,0xbc,0xb9,0xb6,0xb3,
0xb0,0xae,0xab,0xa8,0xa5,0xa2,0x9f,0x9c,0x98,0x95,0x92,0x8f,0x8c,0x89,0x86,0x83,
0x80,0x7c,0x79,0x76,0x73,0x70,0x6d,0x6a,0x67,0x63,0x60,0x5d,0x5a,0x57,0x54,0x51,
0x4f,0x4c,0x49,0x46,0x43,0x40,0x3e,0x3b,0x38,0x36,0x33,0x31,0x2e,0x2c,0x2a,0x27,
0x25,0x23,0x21,0x1f,0x1d,0x1b,0x19,0x17,0x15,0x13,0x12,0x10,0x0f,0x0d,0x0c,0x0a,
0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x03,0x02,0x01,0x01,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x01,0x02,0x03,0x03,0x04,0x05,0x06,0x07,0x08,
0x09,0x0a,0x0c,0x0d,0x0f,0x10,0x12,0x13,0x15,0x17,0x19,0x1b,0x1d,0x1f,0x21,0x23,
0x25,0x27,0x2a,0x2c,0x2e,0x31,0x33,0x36,0x38,0x3b,0x3e,0x40,0x43,0x46,0x49,0x4c,
0x4f,0x51,0x54,0x57,0x5a,0x5d,0x60,0x63,0x67,0x6a,0x6d,0x70,0x73,0x76,0x79,0x7c};
```

ATmega8 AVRStudio Sample Code

http://www.scienceprog.com/generate-sine-wave-modulated-pwm-with-avr-microcontroller/

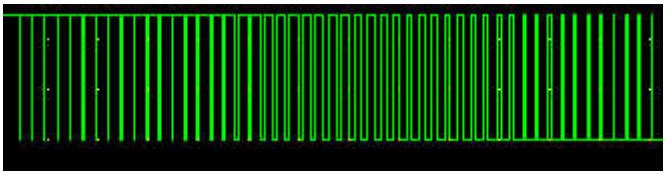
This example will show how to generate a sinewave using PWM modulation. The example uses a AVR microcontroller and only a few lines of code. For this example I used an Atmega8 MCU. All project are set-up on a VMLAB simulator.
To achieve this I saved the sinewave lookup table in Flash program memory (don't forget to include interrupt.h header file):

```
const uint8_t sinewave[] PROGMEM= //256 values {0×80,0×83,0×86,…};
```

PWM is generated by using Phase and frequency correct PWM using 16 bit timer in Atmega8. Modulation is done by updating OCR1A value with one from sinewave table each time when compare matches. Reading from flash program memory is done simply:

```
OCR1A=pgm_read_byte(&sinewave[i]);
```

The line above is placed inside the output compare interrupt for OCR1A service routine. Resulting signal in scope simulator:



```
#include <avr\io.h>
#include <avr\interrupt.h>
#include <avr/pgmspace.h>
const uint8_t  sinewave[] PROGMEM= //256 values
{
0x80,0x83,0x86,0x89,0x8c,0x8f,0x92,0x95,0x98,0x9c,0x9f,0xa2,0xa5,0xa8,0xab,0xae,
0xb0,0xb3,0xb6,0xb9,0xbc,0xbf,0xc1,0xc4,0xc7,0xc9,0xcc,0xce,0xd1,0xd3,0xd5,0xd8,
0xda,0xdc,0xde,0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xed,0xef,0xf0,0xf2,0xf3,0xf5,
0xf6,0xf7,0xf8,0xf9,0xfa,0xfb,0xfc,0xfc,0xfd,0xfe,0xfe,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xfe,0xfe,0xfd,0xfc,0xfc,0xfb,0xfa,0xf9,0xf8,0xf7,
0xf6,0xf5,0xf3,0xf2,0xf0,0xef,0xed,0xec,0xea,0xe8,0xe6,0xe4,0xe2,0xe0,0xde,0xdc,
0xda,0xd8,0xd5,0xd3,0xd1,0xce,0xcc,0xc9,0xc7,0xc4,0xc1,0xbf,0xbc,0xb9,0xb6,0xb3,
0xb0,0xae,0xab,0xa8,0xa5,0xa2,0x9f,0x9c,0x98,0x95,0x92,0x8f,0x8c,0x89,0x86,0x83,
0x80,0x7c,0x79,0x76,0x73,0x70,0x6d,0x6a,0x67,0x63,0x60,0x5d,0x5a,0x57,0x54,0x51,
0x4f,0x4c,0x49,0x46,0x43,0x40,0x3e,0x3b,0x38,0x36,0x33,0x31,0x2e,0x2c,0x2a,0x27,
0x25,0x23,0x21,0x1f,0x1d,0x1b,0x19,0x17,0x15,0x13,0x12,0x10,0x0f,0x0d,0x0c,0x0a,
0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x03,0x02,0x01,0x01,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x01,0x02,0x03,0x03,0x04,0x05,0x06,0x07,0x08,
0x09,0x0a,0x0c,0x0d,0x0f,0x10,0x12,0x13,0x15,0x17,0x19,0x1b,0x1d,0x1f,0x21,0x23,
0x25,0x27,0x2a,0x2c,0x2e,0x31,0x33,0x36,0x38,0x3b,0x3e,0x40,0x43,0x46,0x49,0x4c,
0x4f,0x51,0x54,0x57,0x5a,0x5d,0x60,0x63,0x67,0x6a,0x6d,0x70,0x73,0x76,0x79,0x7c
};
```

```
uint8_t i=0;
ISR(TIMER1_COMPA_vect){
    OCR1A=pgm_read_byte(&sinewave[i]);
   i++;
    }
int main(void) {
//Port D pins as input
DDRD=0x00;
//Enable internal pull ups
PORTD=0xFF;
//Set PORTB1 pin as output
DDRB=0xFF;
// initial OCR1A value
OCR1A=80;
//Output compare OC1A 8 bit non inverted PWM
TCCR1A=0x91;
//start timer without prescaler
TCCR1B=0x01;
//enable output compare interrupt for OCR1A (TIMSKI=0x10 for ATmega8)
TIMSK=0x02;    // OCIE1A = 1
//enable global interrups
sei();
    while (1) {
   //loop forever. Interrupts will do the job.
    }
}
```

Appendix A Methods for Digitally Generating Sine Waves

This page provides an overview of the different methods for digitally generating sine waves **http://homepage.ntlworld.com/moonshadow/New_Folder/electronics.htm**

Square-wave to high order low pass filter

http://www.maxim-ic.com/app-notes/index.mvp/id/2081
The app-note uses an AT90S2323 AVR Microcontroller and
MAX7400CSA+-ND 8th order low-pass filter $5.96, in stock at digi-key

Direct Digital Synthesis

A simple RC filter is used to shape the PWM signal to a sine wave – an analog signal where the amplitude is given by the duty cycle of the PWM output. The RC filter used has an R = 10 kohm and a C = 100 nF, resulting in a filter crossover frequency of 1 kHz,

which will let the low frequency sine wave pass while filtering out the high frequency PWM base.

http://www.atmel.com/dyn/resources/prod_documents/doc2542.pdf
Generate sine wave modulated PWM with AVR microcontroller
http://www.arduino.cc/playground/Code/PCMAudio
http://www.shaduzlabs.com/article-8.html
http://www.scienceprog.com/generate-sine-wave-modulated-pwm-with-avr-microcontroller/
http://documentation.renesas.com/eng/products/region/rtas/mpumcu/apn/sinewave.pdf
http://ww1.microchip.com/downloads/en/AppNotes/00655a.pdf

This appears to be a variant on DDS known as magic sine wave

http://www.sxlist.com/techref/io/pwm/harmonic.htm
http://www.tinaja.com/magsn01.asp

Direct Digital Synthesis ICs

http://www.analog.com/en/rfif-components/direct-digital-synthesis-dds/products/index.html
All DDS ICs have 10 bit resolution
AD9834BRUZ-ND $8.53 (minimum qty. 1, 1,800 in stock), 75 MHz, 28 bit tuning word, serial interface, 2.3v to 5.5v, 20-lead TSSOP
AD9835BRUZ-ND $9.93 (minimum qty. 1, 394 in stock), 50 MHz, 32 bit tuning word, serial interface, 5v, 16-Lead TSSOP
AD5932YRUZ-ND $8.80 (minimum qty. 1, 467 in stock), 50 MHz, 24 bit tuning word, serial interface, 2.3v to 5.5v, 16-pin TSSOP
AD9832BRUZ-ND $8.63 (minimum qty. 1, 1 in stock)
AD9833BRMZ-ND $8.20 (minimum qty. 1, not in stock)
AD5930YRUZ-ND $11.20 (minimum qty. 1, not in stock)

Other solutions do not seem to be widely supported (or at least supported a digikey)
Fairchild Semiconductor ML2035 8-pin DIP

Micro Linear Corporation ML2036, ML2037, not found at Digikey
Maxim MAX038CPP+, MAX038 , 20-pin DIP
Cypress Semiconductor **CY25701FLXC**

Appendix B Other DDS Reading Material

The following articles were reviewed by me and considered to offer, within the context of the main sources cited at the beginning of this report. Having said that, there is some pretty interesting outside-the-box ideas represented in some of the articles

...*of little value*
Simple but powerful Function generator - The generator was build and tested by nuxie1 using XR-2206 function generator
Consider PWM signal
http://www.shaduzlabs.com/article-8.html - Arduino tone generation using low-pass filter

Reference documentation for and early version of waveform generator AVR DDS signal generator V1.0) *article*
AVR controlled DDS generator software writing - *some source code for a waveform generator*
AVR controlled signal generator design considerations - *schematic of waveform generator*
AVR controlled signal generator-skeleton board - *soldered pcb of waveform generator*
AVR controlled signal generator-1 layer PCB prototype - *pcb of waveform generator*

Reference documentation from http://www.arduino.cc/playground/Code/PCMAudio article
http://www.uchobby.com/index.php/2007/11/11/arduino-sound-part-1/f - overview article on methods to generate tones
http://www.evilmadscientist.com/article.php/avrdac - ATmega168 detailed how-to tutorial with code
http://gonium.net/md/2006/12/27/i-will-think-before-i-code/ - code for Arduino with ATmega8, discusses upgrading to ATmega168
http://fly.cc.fer.hr/GDM/articles/sndmus/speaker2.html - article on playing Soundblaster .VOC files
http://www.gamedev.net/reference/articles/article442.asp - Pascal code for playing PC speaker

Stimuli Generator for AVR Studio helps debugging programs in AVR Studio - *general purpose waveform wysiwug editor/generator*