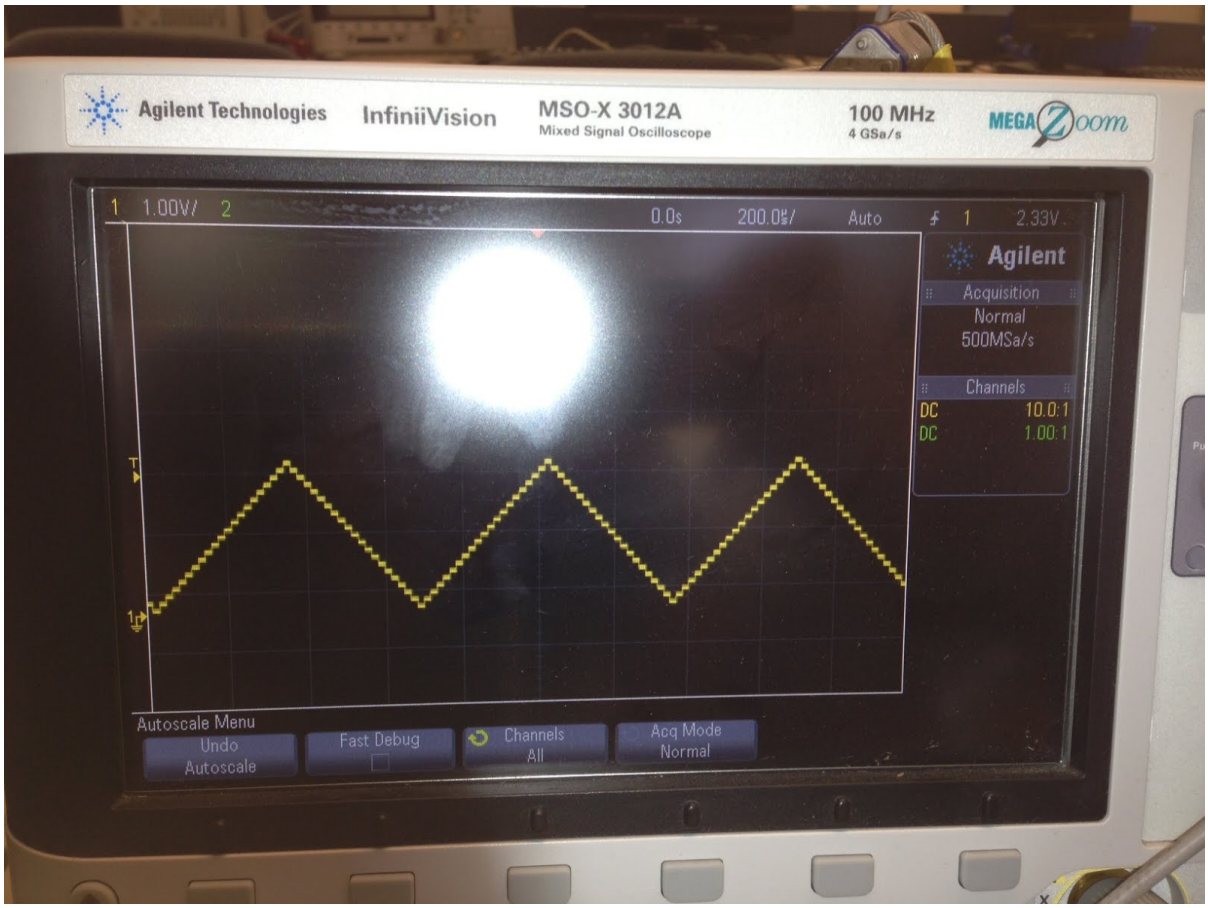
Jason Dreisbach
Jason Descenzo

DAC Tutorial



The most efficient way to produce a sin wave on a microcontroller would be to create a lookup table with the values of the sine function already computed in it. Then it would be possible to simply advance through the array and send the canned value to the DAC.

```
// Tutorial (DAC)
// Jason Dreisbach
// Jason Descenzo

#define MOSI 3                                  // PB pin 3
#define SCK  5                                  // PB pin 5
#define SS   2                                  // PB pin 2

#include <avr/io.h>
#include <util/delay.h>

void Initialize_SPI_Master(void);
void Transmit_SPI_Master(int Data);

int main(void)
{

    DDRB = 1<<MOSI | 1<<SCK | 1<<SS;  // make MOSI, SCK and SS outputs

    Initialize_SPI_Master();

    int count = 0;
    int step = 100;
```

```
    for(;;) {
       if (count > 2000) {
              step = -100;
       }
       else if (count < 200) {
              step = 100;
       }

       Transmit_SPI_Master(count & 0xFFF);   //0xAAA / 0xFFF = 2739 / 4096 = ~ 2.2V
              _delay_us(10);
              count += step;

    }  // end while
    return 0;
}  // end main


void Initialize_SPI_Master(void)
{
    SPCR = (0<<SPIE) |               //No interrupts
    (1<<SPE) |                           //SPI enabled
    (0<<DORD) |                      //shifted out LSB
    (1<<MSTR) |                      //master
    (1<<CPOL) |                      //rising leading edge
    (1<<CPHA) |                      //sample leading edge
    (0<<SPR1) | (0<<SPR0);  //clock speed
    SPSR = (0<<SPIF) |               //SPI interrupt flag
    (0<<WCOL) |                      //Write collision flag
    (0<<SPI2X) ;                     //Doubles SPI clock
    PORTB = 1 << SS;          //make sure SS is high


}

void Transmit_SPI_Master(int Data)
{
       PORTB = 0 << SS;                      // assert the slave select
       SPDR = ((Data >> 8) & 0xF) | 0x70;   // Start transmission, send high byte first
       while (!(SPSR & (1<<SPIF)));// Wait for transmission complete
       SPDR = 0xFF & Data;           // send low byte next
       while (!(SPSR & (1<<SPIF)));// Wait for transmission complete
       PORTB = 1 << SS;                      // de-assert slave select
}
```