Digital DC Motor Speed Regulator

This project is for a microcontroller-based DC motor speed regulator, being used to control a small Chinese-made rotary tool. The tool is available at Harbor Freight as item #97626. The tool fits very nicely in the hand and is extremely useful for precision cutting, drilling and polishing. However, for delicate work, the tool is in bad need of a speed regulator. The tool would scream at high rpm when unloaded, but it would slow to a crawl when much of a load is put on it. Back in 1994, I created a PWM motor speed regulator that solves this problem, using the 555 timer IC. I still use the circuit today and I published it online in 2005 (see link below). I recently created a digital version of it, using a motor control technique that I invented years ago. Why the patent application failed will be explained later. This digital speed regulator concept can be used with other DC motors as well, and I’ve seen it used in an electric floor sweeper. The gadget described here requires a 12 to 15 volt, 2 amp power supply that need not be well regulated. I use a discarded 12 volt printer power supply with mine.

This digital version is at least as energy efficient as the analog version. It is also much cheaper and simpler to build than the analog version. Since the microcontroller does all the work, the circuit has few hardware components. The code is so simple that it uses less than half the program memory of the cheapest available microcontroller, the PIC10F200. The code should be extremely easy to rewrite to run in other microcontrollers. The circuit has a bit more torque than the analog version, because no series resistor is needed to sense and compensate for the motor’s load. The analog version also has a current limiting circuit, whereas the digital version limits the overload time.

The circuit employs a combination of “bang-bang” control and PWM (pulse width modulation) to efficiently control the motor. “Bang-bang” control is like your home thermostat. It’s either on or off, but never somewhere in between. The control system “bangs” back and forth between the limits. “Bang-bang” motor control has the same efficiency as a linear control system, except that the surplus energy is wasted in the motor windings instead of a control transistor. The motor speed oscillates back and forth across the threshold, but the average speed is quite stable. The speed jitter (rapid fluctuation) becomes quite bad if the input voltage is really high or the set motor speed is really low.

PWM is much more efficient and controls the power to the motor by switching it on and off at a high rate (3 KHz in this case). The motor’s inductance stores the energy in the pulses and releases it as a steady current through the motor. The power level is changed by varying the on-time in relation to the off-time of the power applied to the motor. This on/off ratio is called “duty cycle”. This motor speed regulator is a hybrid of both control techniques.

The hybrid speed regulation scheme employed here is a compromise, exploiting the best of both worlds. It has most of the simplicity of a “bang-bang” system along with most of the energy efficiency of a PWM system. Because the speed sense input of the microcontroller used here is just a digital I/O pin, full-up PWM is not possible. A full-up, but basic PWM system similar to my analog unit would require a microcontroller with an analog input. It would also require a power-wasting series resistor to sense motor load and stall condition. The hybrid system uses only software to detect motor stall and needs no series resistor to sense the motor speed. A full-up PWM system, without a tachometer, senses only the motor load and compensates for it (called IR compensation). IR in this context means I (current) times R (resistance) or in other words, the load-related voltage drop in the motor wiring. The hybrid system senses the actual speed of the motor.

This system has 18 discrete power levels (not including the zero level) which are selected based on whether or not the voltage across the momentarily switched-off motor is above or below a threshold. This voltage is the “Back EMF” (BEMF) and is proportional to the motor speed. The spinning motor acts like a generator, producing a voltage that opposes the applied voltage. EMF means “electromotive force”, the same thing as voltage, which is the force that motivates the movement of electrons through a circuit. If the BEMF of the motor is above the threshold when sampled, the next lower power level is selected and if the BEMF is below the threshold, the next higher power level is selected. If the system tries to go above the maximum power level, a red “Overload” LED is lit. If this occurs continuously for 3 seconds, the motor will shut down and the LED will stay on. The main power must then be turned off and back on again to reset the system.
The more discrete power levels used, the smoother the motor operation, but the longer it takes the motor to respond to a sudden load change. Also, the higher the speed sample rate, the smoother the motor operation, but the lower the maximum power available to the motor, due to the time that the motor has to be turned off to sample the speed. The software can be tuned to a particular motor by changing four “parameters” specified at the beginning of the code without having to mess with the code itself. The overload timer delay (currently 3 seconds) is also adjusted there.

This system runs a bit jittery because of the ripple (rapid fluctuations) in the motor’s BEMF. This is caused by the fact that the strength of the magnetic field seen by the armature of the cheap motor is not consistent throughout the motor’s rotation. The digital algorithm (process) used here was developed specifically to deal with this ripple. The motor in the invention it was created for had to come into speed regulation almost instantly and without a tachometer. There was no time to average out the ripple in the BEMF, but the average speed still had to be precisely controlled. The motor itself did the speed averaging. The analog version of my motor speed regulator runs a lot smoother, because the motor sees smooth DC, continuously adjusted by the motor’s BEMF. The PWM is not directly applied to the motor. The digital system takes periodic instantaneous samples of the BEMF and makes corrections after-the-fact. Also, in the analog system, the control-loop gain is adjusted to precisely compensate for load effects, which is not possible with this digital design. Still, the simplicity of the digital system may make it attractive for certain things, especially where cost is an issue. The control system will have a lot less jitter in applications where the motor and its load have a lot of mass, such as the floor sweeper with its large motor and rotary brush.

I prefer using a knob to set the motor speed, but pushbuttons could be used for a more compact design if a microcontroller is used that has an analog input, allowing the switching threshold to be changed in software. The second of the two invention prototypes mentioned below used an analog input and software to precisely set the switching threshold, even though it ran at only one speed. Without an analog input, several resistors and multiple I/O pins can be used to set multiple switching thresholds. For example, three resistors and two I/O pins could be used to set three fixed speed thresholds. The control system will have a lot less jitter in applications where the motor and its load have a lot of mass, such as the floor sweeper with its large motor and rotary brush.

I developed the algorithm used in this project back in late 1999 for use in two prototypes of a commercial product. The prototypes worked well and a patent application was completed in June of 2003. My employer acquired a competitor and shut down the facility where I worked in February 2003 while the lawyers were still working on the patent application. If interested, see USPTO published application #20030107341. The patent application was published because it is required when applying for patents overseas, which my former employer did as a matter of course. A patent was not granted because I was unable to come up with signatures of two qualified “witnesses” to verify the invention. The patent office requires two witnesses who understand the invention, but are not co-inventors. I was the only electrical engineer where I worked and the signature form was sent to me after I was laid off. At that time, I had no access to anyone with the technical background needed to understand the invention and the motivation to help my former employer. The invention, therefore, came into the public domain on June 13, 2004, one year after being published. I developed the system entirely on my own and I don’t think my employer was very interested in it, anyway. No one really understood the “outside-the-box” concept, which was a lot more complex than just the motor speed regulation scheme, which is the only part of the invention that is used here.

Back in 2006, I bought and reverse-engineered a “microprocessor-controlled” cordless electric floor sweeper which contains a microcontroller (PIC12F508) and employs a motor speed regulation system that is similar to mine. Perhaps the designer used my idea, after having found it in a patent search. It would have been entirely legal to do so.

Link to the 555-based (analog) version: http://www.discovercircuits.com/Andy/MotorSpeedRegulator.pdf
R5 sets the motor speed. D5 is the “Overload” LED. The ON-OFF switch is wired this way to speed up resetting the PIC in case the overload protection is tripped and to extend the life of the switch contacts. When the PIC is off, nothing else can draw current. D3 is temperature compensation for Q1. The PIC input will already have some temperature sensitivity, which is unknown as this is not considered normal use of the I/O pin and is not documented. C4 and C5 filter out motor brush noise. C3 should be mounted as close as possible to U1 to protect it from motor voltage spikes and to ensure that enough peak current is available to quickly overcome Q2’s gate capacitance. R8 limits this peak current to below the maximum specified in the PIC datasheet. It also limits current from motor voltage spikes flowing into the PIC output through Q2’s capacitance. Since the PIC I/O pins become high impedances when the PIC is not powered up, R7 ensures that Q2 remains off. D2 prevents the PIC input from seeing any voltage higher than about 5.1 volts while Q2 is on. Q3 is an emitter-follower, designed to minimize the load on the PIC and to keep a constant LED current, regardless of power supply voltage. This circuit is specifically designed not to require a regulated power supply. This is why Q1 exists. Its purpose is to reference the motor’s BEMF to ground instead of the +12V. The floor sweeper has no such transistor, because it uses NI-CAD batteries which maintain a constant voltage. Note that it is better to use an N-channel MOSFET as the motor driver because it has half the on-resistance for a particular die-size (affects cost and gate capacitance) than an otherwise similar P-channel MOSFET.

The PIC was programmed with a PICSTART PLUS, using MPLAB IDE Version 8.70. If you are new to programming the PIC10F200 or the PIC10F202, be aware that pin 1 of the PIC10F20x is plugged into pin 9 of the PICSTART PLUS ZIF socket.

Besides making your workmanship more precise, this circuit should significantly extend the life of your mini-drill. Using the analog version of this circuit, my first mini-drill is still working fine after about 18 years of occasional, but sometimes heavy, use. I bought a new tool to preserve the integrity of this article by verifying that the tool specified is interchangeable with the one for which this gadget was originally designed. Analog or digital, when I need to do precision cutting or drilling, I want low RPM but I also need decent torque. This gadget does that.

Please feel free to email me with questions about either the digital or the analog version of this gadget.
Andrew R. Morris  morr3763@bellsouth.net
Figure 2. Component layout.
Most reference designators were omitted to avoid obscuring the view of the connecting wires. The circuit board will fit into either the first or the fifth card slot in the enclosure, depending on how the circuit board is oriented. The hole for the pot is centered and drilled exactly 23mm from the inside edge of the flange inside the front cover. The LED is press-fitted into a size #10 hole. No glue is needed if you use the specified LED.

Figure 3. Exterior photo

The mini-drill has a 2.1mm in-line jack on it. Therefore a 2.1mm power cable is specified to go from the control unit to the tool. The plug from my AC adapter is 2.5mm, but 2.1mm is more common. I have therefore listed part numbers for both a 2.1mm and a 2.5mm jack.
The jack where the AC adapter plugs into is partially visible in the lower right side of the photo. Notice that the cord to the motor plug is wrapped around the switch for strain relief. I cut the length of that cord to about 2 feet (outside the enclosure), because the tool itself already has a fairly long cord on it. Excessive cord length could reduce the maximum available power.

I barely managed to get all the parts onto the tiny circuit board, but I wanted to mount the pot onto the board if I could. The anti-rotation pin on the pot is snipped off, since the circuit board can’t turn, because it’s held in place by the card slots in the enclosure. Things may not fit this way if you don’t use the specified pot. BTW, the test point (lower left of the PIC) serves no function except to provide a sync point for a scope during testing or troubleshooting.

Below is the ASM code for the gadget. Just copy and paste it into an ASM file and assemble it into a HEX file. I have already done this and verified that there are no errors in it.
Motor speed regulator

Written by:
Andrew R. Morris
morr3763@bellsouth.net

This code was written for a PIC10F200 or PIC10F202

This code uses a combination of "bang-bang" and PWM to stabilize
the speed of a small DC motor. The current settings are optimized
for a small Chinese-made rotary tool, used for cutting and drilling.

LIST P=10F200
#include P10F200.INC

errorlevel -302 ;suppress message 302 from list file

__CONFIG _CP_OFF & _MCLRE_OFF & _WDT_OFF & _IntRC_OSC

;Register equates

LCVR1 EQU 10 ;Loop count variable register 1
LCVR2 EQU 11 ;Loop count variable register 2
LCVR3 EQU 12 ;Loop count variable register 3
LCVR4 EQU 13 ;Loop count variable register 4
LCVR5 EQU 14 ;Loop count variable register 5
TEMP EQU 15 ;Temporary storage register
ON_TIME EQU 16 ;On time register
OFF_TIME EQU 17 ;Off time register

;Initialization equates

INIT_GP EQU B'00000000' ;Initialize GPIO
CFG_GP EQU B'00001000' ;Configure GPIO
OPTINI EQU B'11000000' ;Initialize option register

;Parameter equates --Used to tune motor response--

PWM_CNT EQU D'18' ;Number of PWM levels, excluding 0.
PWM_FRQ EQU D'5' ;Sets PWM frequency (around 3KHz)
SAM_FRQ EQU D'27' ;Sets sample frequency (around 100Hz)
SNS_TIM EQU D'255' ;Wait time to sense motor speed
OVL_TIM EQU D'3' ;Overload time in seconds before tripping
OVL_CAL EQU D'100'; Overload timer 1 second calibration

; I/O pin equates

; GPIO
MOT_OUT EQU 0 ; Output to motor drive
TST_OUT EQU 1 ; Test point output
OVL_LED EQU 2 ; Overload LED
SPD_FB EQU 3 ; Speed feedback from motor

; ORG 0 ; Reset vector rolls over to here
GOTO Boot ; Go to beginning of program

; Subroutines

PWM MOVLW SAM_FRQ ; Number of PWM iterations (Affects sample frequency)
MOVWF LCVR1
MOVLW PWM_CNT ; Total PWM levels, excluding 0 level.
MOVWF TEMP
SUBWF ON_TIME,W
BTFSC STATUS,Z ; If ON_TIME is equal to PWM_CNT (maximum), turn
GOTO MAX ; output on for total PWM time.
MOVF ON_TIME,W
BTFSC STATUS,Z ; If ON_TIME is 0, turn output off for total PWM time.
GOTO MIN
SUBWF TEMP,W
MOVWF OFF_TIME

LP1 BSF GPIO,MOT_OUT ; Apply power to the motor and wait for the
MOVF ON_TIME,W ; ON_TIME period.
MOVWF LCVR2 ; This delay routine is duplicated three times
MOVLW PWM_FRQ ; and not called as a separate subroutine
MOVWF LCVR3 ; for timing reasons and because there is
DECFSZ LCVR3,F ; plenty of available memory.
GOTO $-1
DECFSZ LCVR2,F
GOTO $-5

BCF GPIO,MOT_OUT ; Turn off motor and wait for the
MOVF OFF_TIME,W ; OFF_TIME period.
MOVWF LCVR2 ; Same delay routine as above.
MOVLW PWM_FRQ
MOVWF LCVR3
DECFSZ LCVR3,F
GOTO $-1
DECFSZ LCVR2,F
GOTO $-5

DECFSZ LCVR1,F ;Count off the correct number of PWM on/off cycles
GOTO LP1 ;before the next speed sample.
RETLW 0 ;Leave PWM routine when done.

MAX BSF GPIO,MOT_OUT ;Turn on motor for the full PWM on/off duration.
GOTO $+2
MIN BCF GPIO,MOT_OUT ;Turn off motor for the full PWM on/off duration.
LP2 MOVLW PWM_CNT ;Counts off time motor is to be on or
MOVWF LCVR2 ;off continuously, instead of PWM. Same delay
MOVLW PWM_FRQ ;routine as used twice before.
MOVWF LCVR3
DECFSZ LCVR3,F
GOTO $-1
DECFSZ LCVR2,F
GOTO $-5

DECFSZ LCVR1,F ;Counts off the correct number of PWM on-cycles
GOTO LP2 ;or off-cycles before the next speed sample.
BCF GPIO,MOT_OUT ;Turn off motor if it was on.
RETLW 0

OVL_INI ;Overload counter initialize.
    BCF GPIO,OVL_LED ;Turn off overload LED.
    MOVLW OVL_TIM ;Counts off 3 seconds.
    MOVWF LCVR4
    MOVLW OVL_CAL ;Counts off 1 second, assuming
    MOVWF LCVR5 ;a 100Hz BEMF sample rate.
    RETLW 0

OVL_CNT ;Overload counter.
    BSF GPIO,OVL_LED ;Turn on overload LED
    DECFSZ LCVR5,F
    GOTO START ;Counts down every time an attempt is made
    MOVLW OVL_CAL ;to to move to a higher power level while
    MOVWF LCVR5 ;already at the maximum level.
    DECFSZ LCVR4,F
    GOTO START ;Continue normal operation.
    BCF GPIO,MOT_OUT ;Overload timed out. Stop motor
    GOTO $ ;Freeze. Need to reboot.

DL1 MOVWF LCVR1 ;Delay routine. D'255' in the W register
DECFSZ LCVR1,F ;when this routine is called will produce
GOTO $-1 ;a delay of about 768uS.
RETLW 0
Boot

MOVWF OSCCAL ;Load the factory internal oscillator calibration value

; Initialize I/O ports

MOVLW CFG_GP ;Select direction of GPIO bits
TRIS GPIO ;Write selection to GPIO data
            ;direction register

MOVLW INIT_GP ;Initialize GPIO
MOVWF GPIO

CLRF STATUS
CLRF TMR0 ;Reset TMR0 and prescaler

CLRWDT ;Reset watchdog timer

MOVLW OPTINI ;Initialize option register
OPTION
MOVLW PWM_CNT ;Initialize at full power
MOVWF ON_TIME

START ;The actual program starts here.

; MOVFW D'9' ;+++ PWM forced at 50% +++

; MOVWF ON_TIME ;+++ for hardware testing. +++

CALL PWM
BSF GPIO,TST_OUT ;Set TEST POINT high

CALL DL1 ;Wait for motor inductive kick to die out.
BCF GPIO,TST_OUT ;Set TEST POINT low
BTSC GPIO,SPD_FB ;Test if motor feedback is high or low.
GOTO PDN

PUP

MOVLW PWM_CNT
SUBWF ON_TIME,W
BTSC STATUS,Z ;If ON_TIME is equal to PWM_CNT (maximum), do not
GOTO OVL_CNT ;increment. Go and decrement overload counter.
INCF ON_TIME,F ;Bump power level up.
GOTO START

PDN

CALL OVL_INI ;Initialize (reset) overload counter.

MOVF ON_TIME,F
BTSC STATUS,Z ;If ON_TIME is 0, do not decrement.
GOTO START
DECF ON_TIME,F ;Bump power level down.
GOTO START

END