Atmel AVR Contest

Entry: AT2792

A Microstepping Bipolar Stepper Motor Driver for CNC Applications

Abstract

Project Summary

This project is the implementation of a low-cost, high-performance bipolar stepper motor driver. This driver was designed for the hobbyist user (myself!) who is retrofitting a piece of equipment for CNC operation using one of the inexpensive and readily available interpreter programs. Generally speaking, these programs are capable of reading the GCode output from a CAM program and outputting step and direction signals to the parallel port. It is the function of the driver built in this project to receive these step and direction signals and control a stepper motor based upon them.

The 'brain' of the driver is an **ATMega 48** running on its internal 8 Mhz oscillator. When tenth stepping, this is sufficient to run the stepper motor at speeds up to 3000 rpm, or a step pulse every 80 instruction cycles.

The 'brawn' of the driver is two National Semiconductor LMD18245 full-bridges – one for each stepper motor phase. Each full-bridge includes a built-in 4-bit D/A converter as well as current sensing capability. The sensed current is limited to a maximum value as set by the user with an external trimmer potentiometer. The current is further limited based on the input signal to the D/A converter and its reference voltage. This capability allows us to (a) implement microstepping and (b) drive the motor with up to 55 V_{DC} for maximum speed performance.

To allow movement in subdivisions of a full-step, microstepping tables are stored in the controller's program memory. Microstepping allows for maximum smoothness and increased resolution. For maximum flexibility, there are a total of 8 different tables of various microstepping resolutions that the user may use DIP switches to select. An enable signal is continuously monitored by the microcontroller. When enabled and upon the receipt of a step signal, the direction signal is sampled. Based on the state of the direction signal, either the previous or next microstepping table entry is read from memory. This data is then transferred to inputs of the full-bridges' D/A converters. One D/A converter (and winding current) is driven with a sine-wave approximation and the other D/A converter (and winding current) with a cosine-wave approximation.

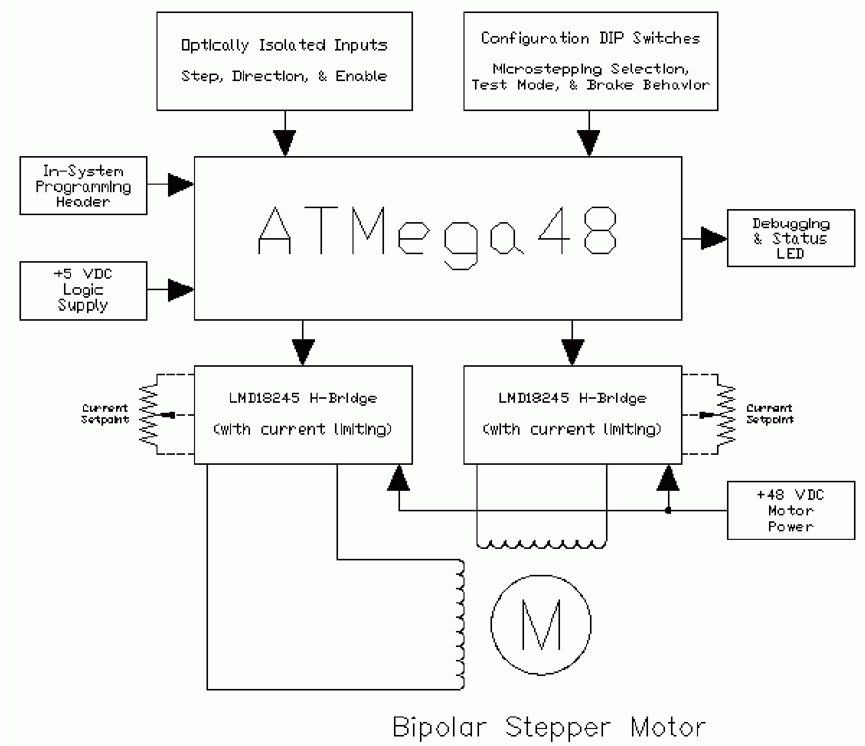
Time between step pulses is monitored. If three seconds of no activity pass, the microcontroller reduces the reference voltage to the full-bridge D/A converters. This serves to reduce the winding currents and allows for reduced motor heating. Upon the recipient of another step signal, the reference voltage is restored for normal winding currents.

A test mode is also implemented. By setting a DIP switch, the motor will rotate at a pre-determined rate of 1 revolution per second. In this mode, the microcontroller is internally generating the step signal. This allows the user to verify power wiring and driver/motor functionality without an externally generated step signal.

Another DIP switch allows the user to determine the motor behavior when the driver is not enabled. One option is to energize the lower half of both full-bridges. This effectively brakes the motor and can hold a load in place against an external force like gravity. The other option is to fully disable both full-bridges and allow the motors to freewheel. If a double-shafted motor is used, a handwheel can then be mounted on the rear motor shaft and allow the operator to manually position the motor.

A status LED is also provided. It normally flashes at a predetermined rate. If a fault is detected – though one of the numerous software traps – it will illuminate solid.

Block Diagram



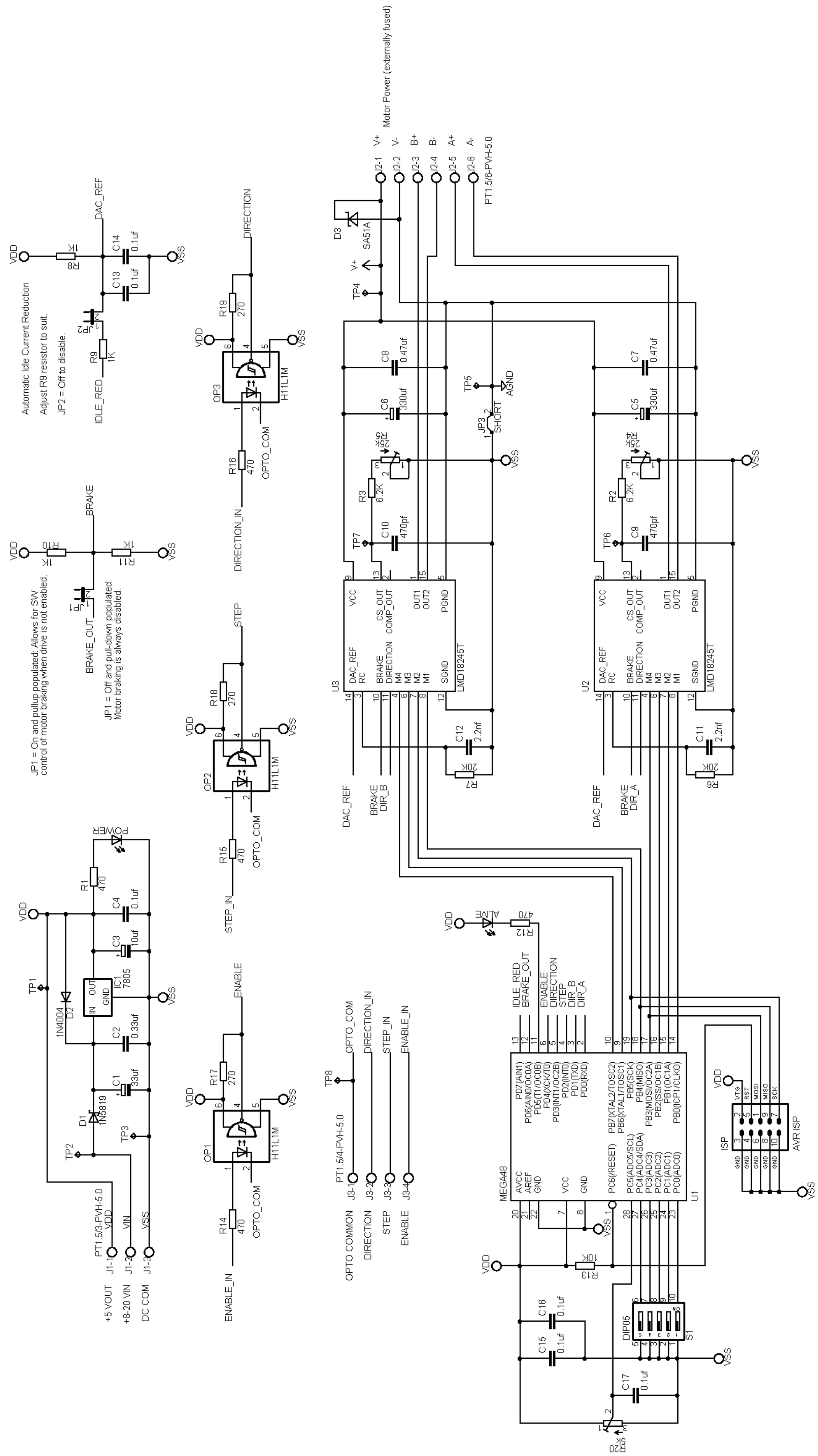
Photographs

An Assembled Board

Final Home for 3 Boards

The mill & motors the boards will drive.

Schematic


Motor Power (externally fused)
Automatic Idle Current Reduction
Adjust R9 resistor to suit.
JP2 = Off to disable.
JP1 = On and pullup populated: Allows for SW control of motor braking when drive is not enabled.
JP1 = Off and pull-down populated: Motor braking is always disabled.
LMD18245T
LMD18245T
MEGA48
H11L1M
H11L1M
H11L1M
AVR ISP

Code Sample

```
; **********************************************************************
ext_int0:    ; IRQ0 handler
             ; This interrupt handles the step input from the opto-isolator.
             ; As a new step has occurred we first disable the idle current reduction.
             ; Then, based on the polarity of the direction input, we read the
             ; previous/next microstepping codes from program memory and
             ; output them to the H-bridges.

    in  isr_SREG, SREG          ; save status register

    sbi portd, idle_red_out     ; disable idle current reduction
    clr tim2_idlerolls          ; reset idle current reduction rollover counter
    ldi isr_temp1,  0b00000001  ; re-enable idle current reduction timer
                    ; xxxxx---  ; reserved
                    ; -----0--  ; 0 = timer/counter2 Output Compare B Match interrupt disabled
                    ; ------0-  ; 0 = timer/counter2 Output Compare A Match interrupt disabled
                    ; -------0  ; 0 = timer/counter2 Overflow interrupt disabled
    sts TIMSK2, isr_temp1

    mov ZL, seq_start_L         ; initialize ZL and ZH
    mov ZH, seq_start_H

    sbic pind, dir_in           ; check if direction pin is high...
    rjmp forward                ; ...yes, forward direction

reverse:
    clr isr_temp1
    dec current_position                ; decrement to next position...
    dec current_position                ; ...requires two decrements to make the word entries...
                                        ; ...into byte equivalent address
    cp  max_entries, current_position   ; check if we are past the beginning of the sequence...
        brsh calc_Z                     ; ...no, then read data
    mov current_position, max_entries   ; ...yes, then point to last set of data entries
    rjmp calc_Z

forward:
    inc current_position                ; increment to next position...
    inc current_position                ; ...requires two increments to make the word entries...
                                        ; ...into byte equivalent address
    cp  max_entries, current_position   ; check if we are we past the end of the sequence...
        brsh calc_Z                     ; ...no, then read data
    clr current_position                ; ...yes, then point back to beginning

calc_Z:                                 ; use the current_position to calculate the Z pointer to
new sequence
    clr isr_temp1
    add zl, current_position            ; 16 bit addition to point to the new sequence
    adc zh, isr_temp1

read:                                   ; read the step sequence data
    lpm isr_temp1, Z+                   ; read new data for dir pins, will be 0b000000xx where xx
                                        ; is the new data
    in  isr_temp2, portd                ; get a copy of portd to work with
    andi isr_temp2, 0b11111100          ; clear direction pins
    or  isr_temp1, isr_temp2            ; OR in the new data
    out portd, isr_temp1                ; update portb

    lpm isr_temp1, Z                    ; read new data for M pins
    out portb, isr_temp1                ; update portb

    out SREG, isr_SREG                  ; restore status register
    reti
; **********************************************************************
```