Motors

E155

Sources

- Harris and Harris 2nd Edition: Chapter 8
- Erik Brunvand:
  - http://www.eng.utah.edu/~cs5968/2009/slides/motors.pdf

Types of Motors

- DC motors
  - High current
  - Powerful driver (H-bridge)
- Servo motors
  - Not as powerful
  - Specific position (not continuous rotation)
- Stepper motors
  - Rotates by a fixed angle (step)
  - Expensive and need a powerful driver

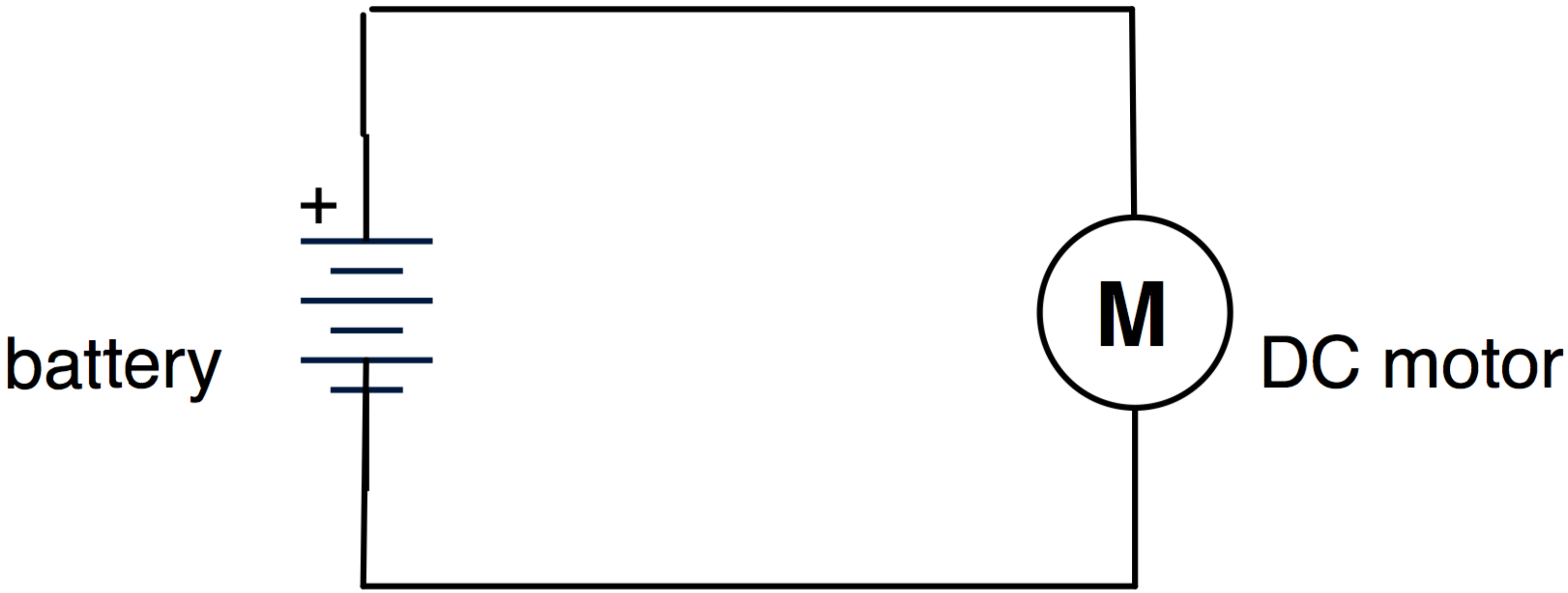
DC Motor Parameters

- Direct-drive vs. gearhead
- Voltage
- Current (efficiency)
- Speed
- Torque
- Other (sizes)

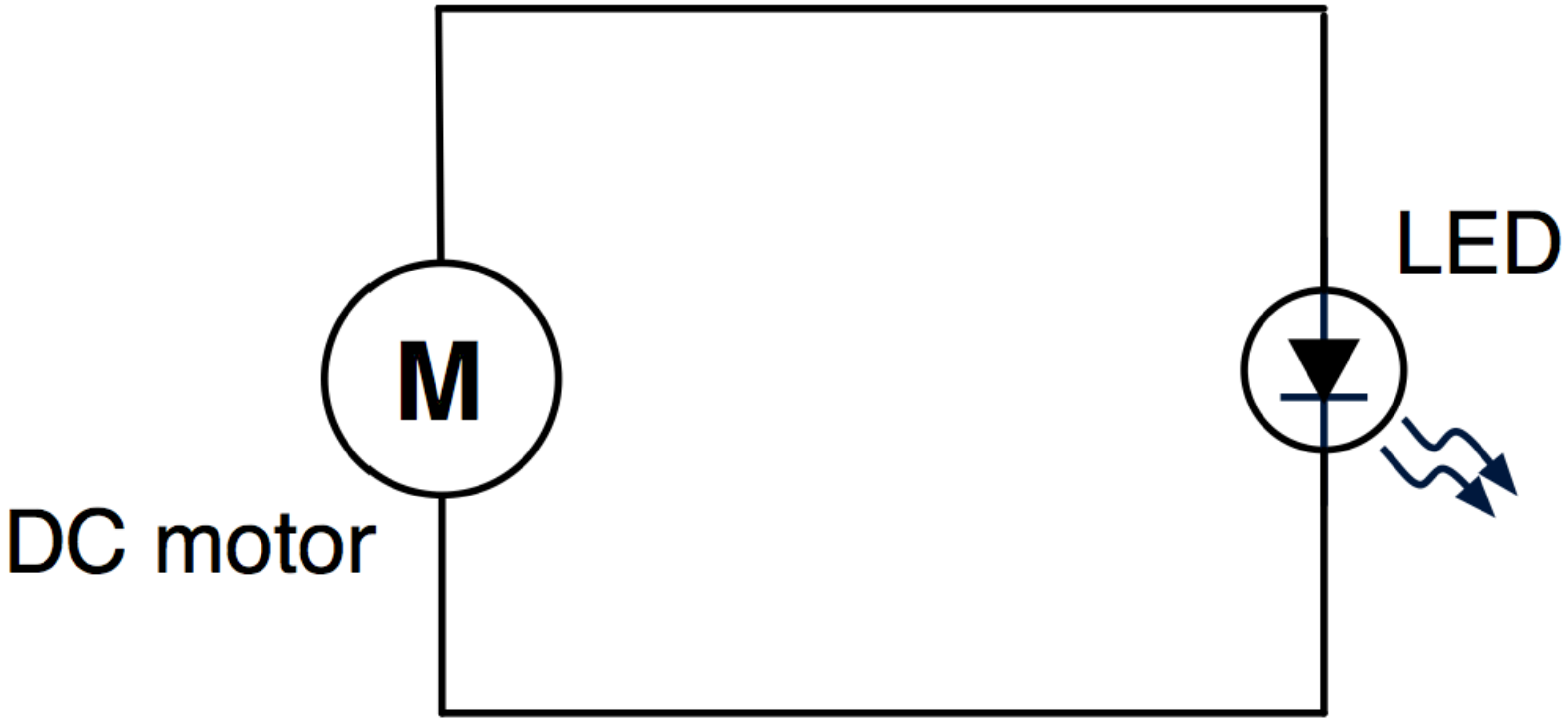
DC Motor Characteristics

- More current on startup
- Stall – lots of current
- Polarity determines direction
- Usually spin very fast: >1000 RPM
- To get slower spinning, need gearing

Battery Powered Motor

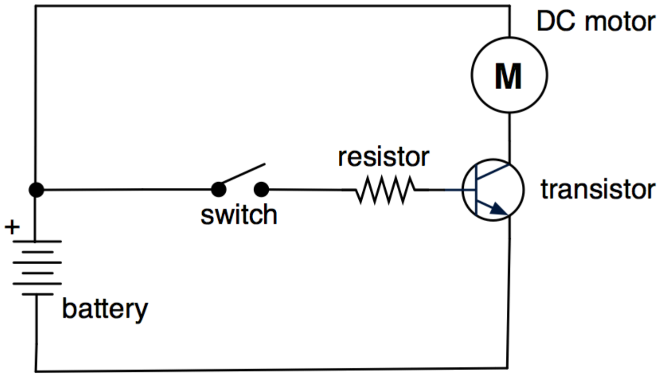


DC Motor as a Generator

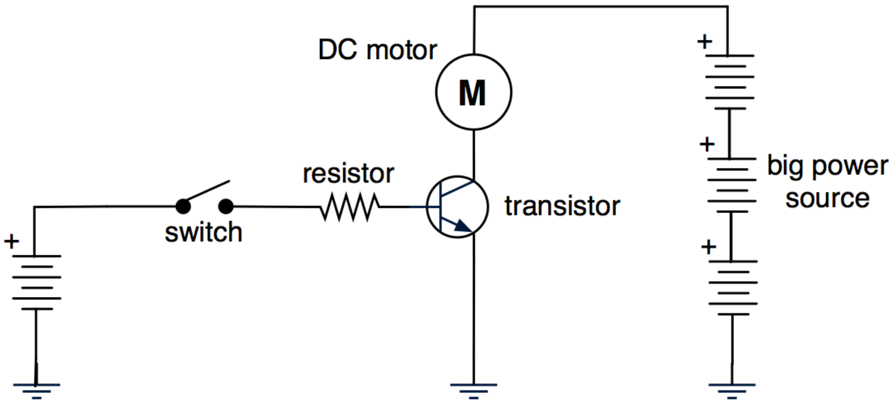


Switch Controlled Motor

little motor

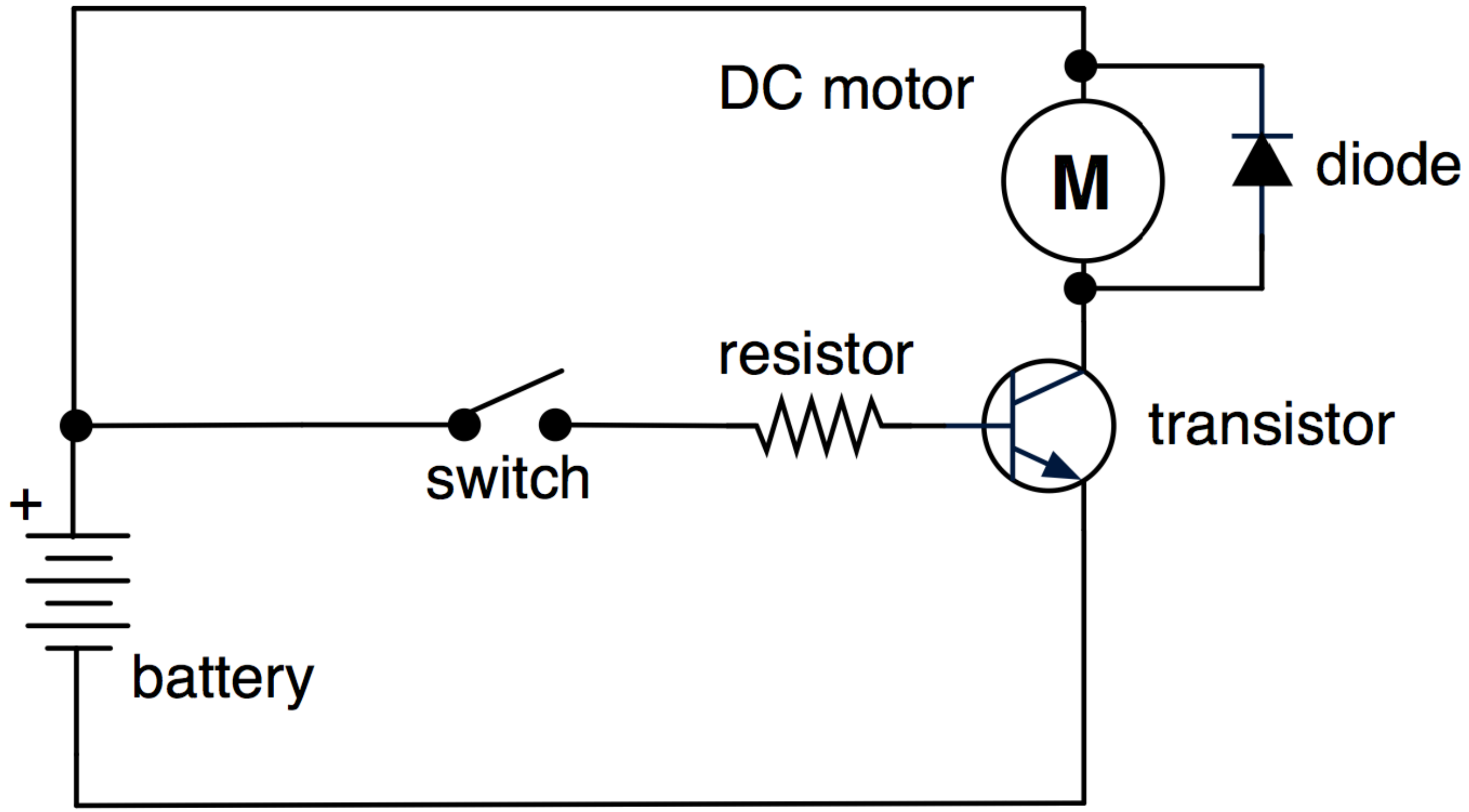


big motor



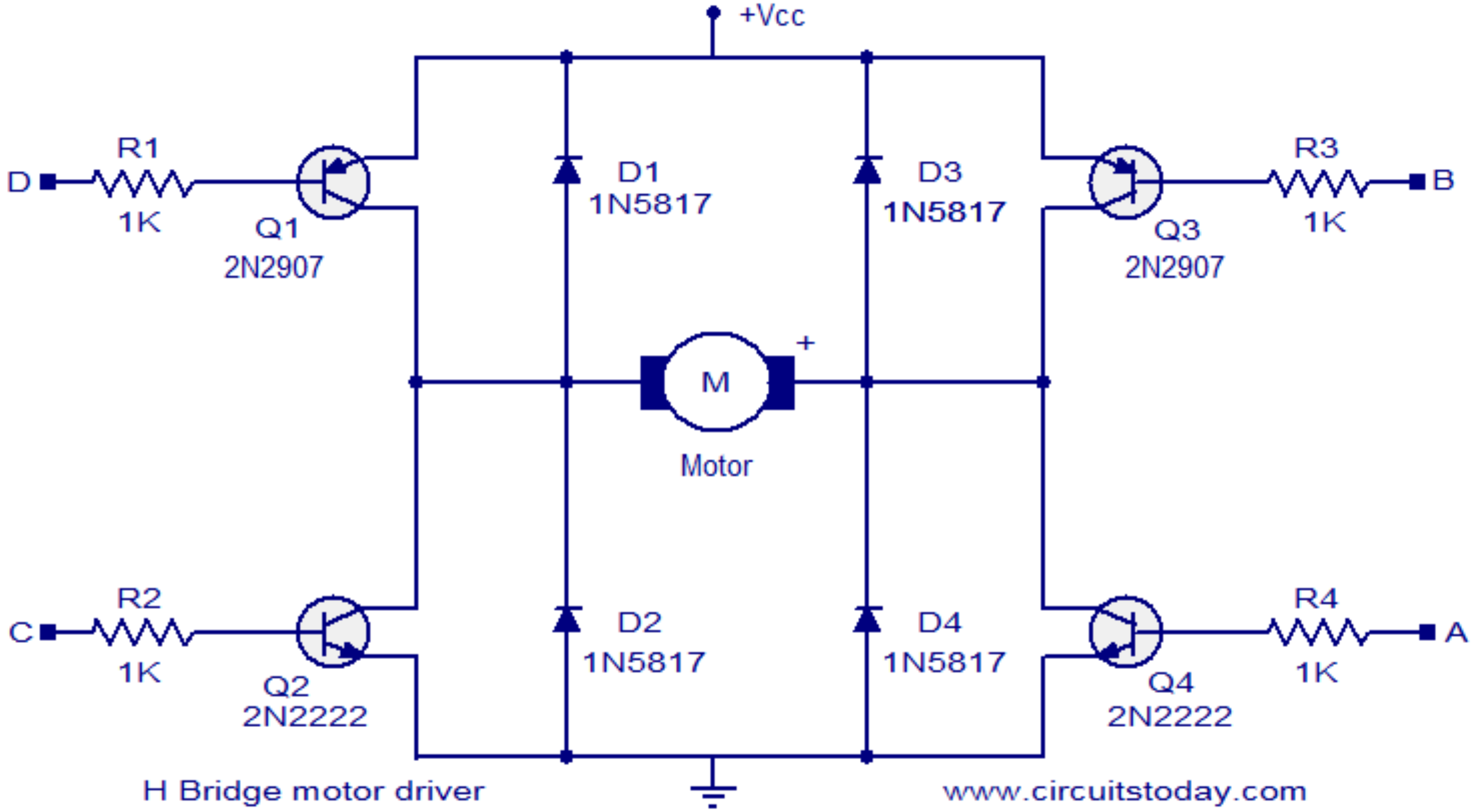
switching a different power source

Add a Diode



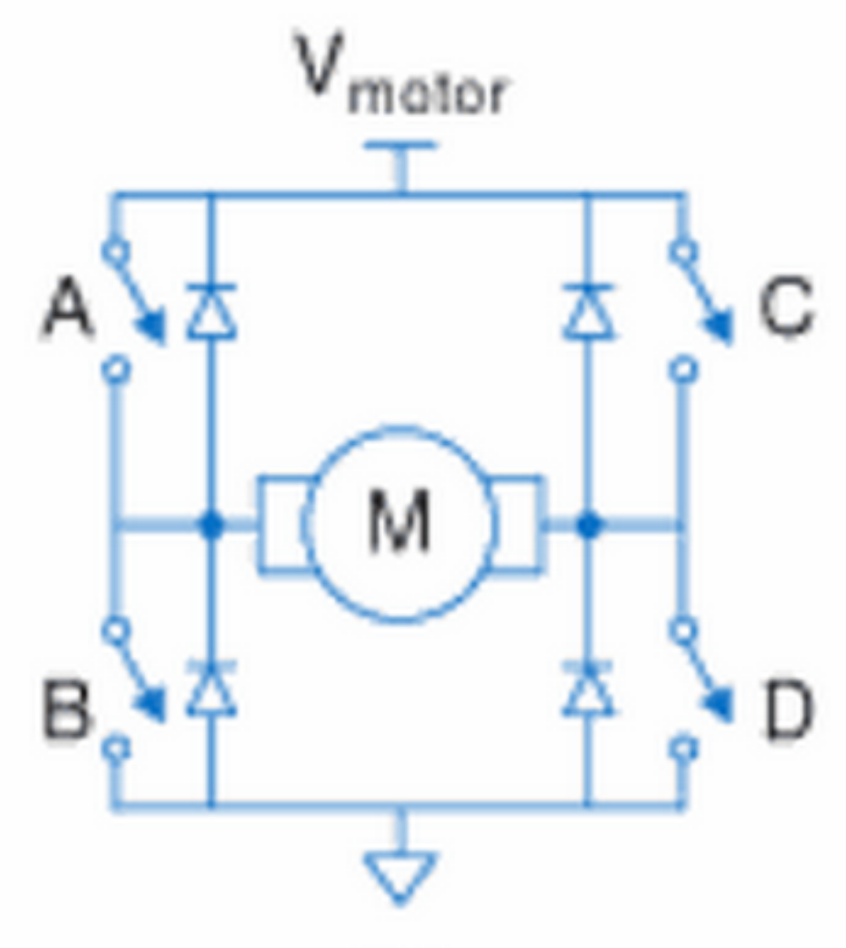
H-Bridge

- Four electrically controlled switches

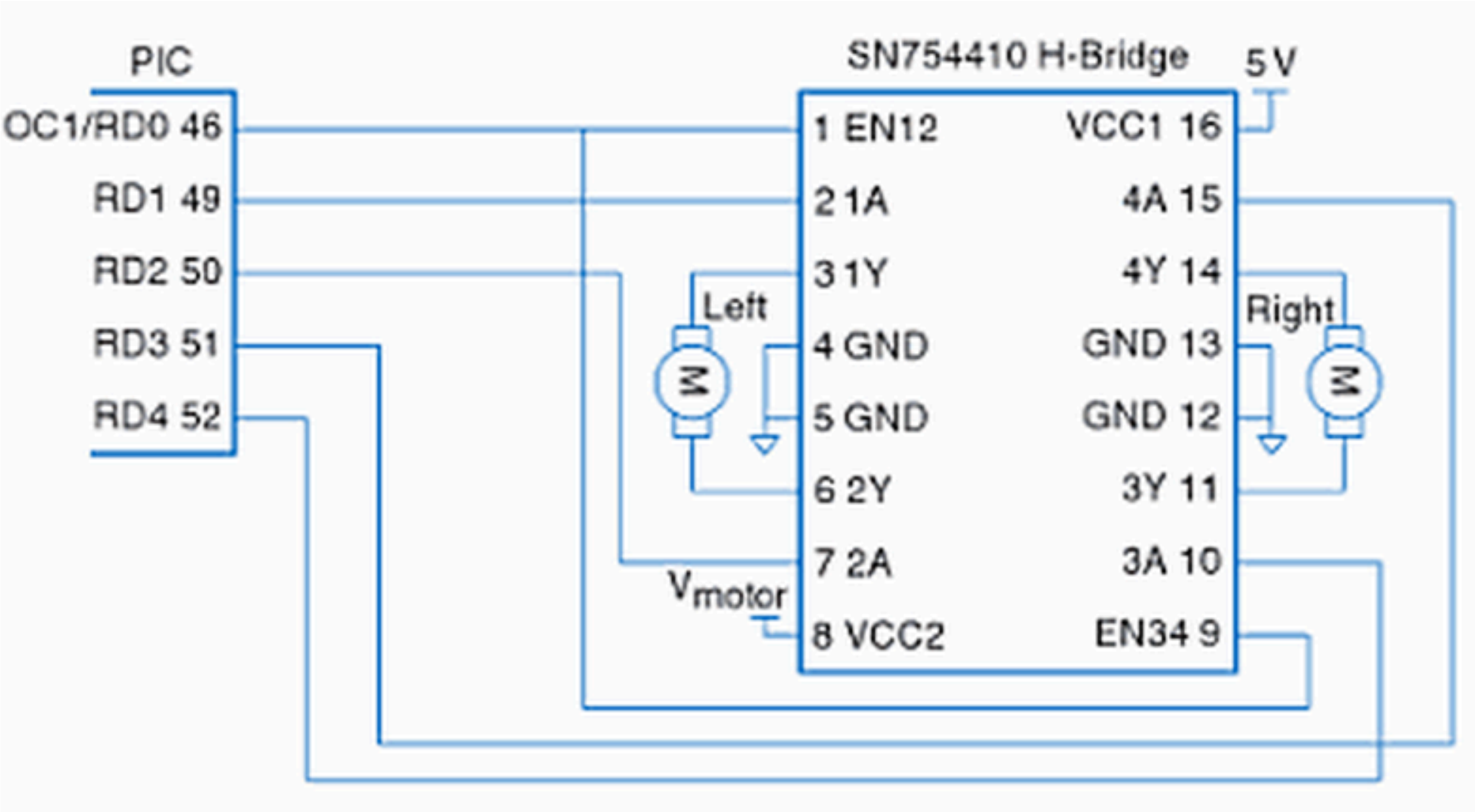


http://www.circuitstoday.com/wp-content/uploads/2011/01/h-bridge-motor-driver.png

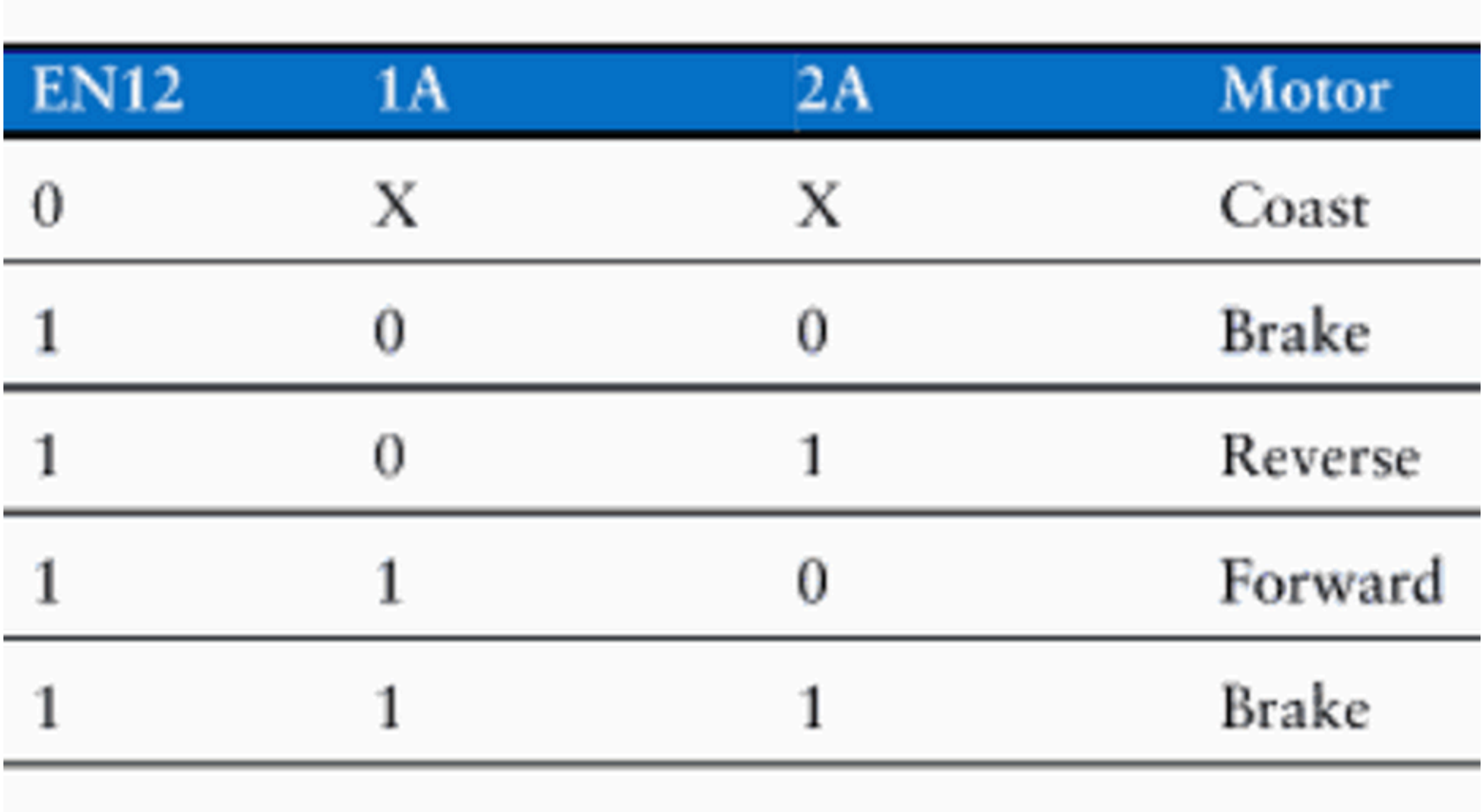
H-Bridge from book



PWM Example from Book



H-Bridge Control



EN12	1A	2A	Motor
0	X	X	Coast
1	0	0	Brake
1	0	1	Reverse
1	1	0	Forward
1	1	1	Brake

Example Code

```
void setspeed(int dutycycle) {
   OC1RS = dutycycle;     // set duty cycle between 0 and 100
}
void setmotorleft(int dir) {  // dir of 1 = forward, 0 = back
   PORTDbits.RD1 = dir; PORTDbits.RD2 = !dir;
}
void initmotors(void) {
   TRISD = 0xFFE0;
   halt();
   T2CONbits.TCKPS = 0b111; // prescale by 256 to 78.125 KHz
   PR2 = 99;                 // set period to 99+1 ticks ~781Hz
   OC1RS = 0;                // start with low H-bridge enable
   OC1CONbits.OCM = 0b110;   // set output compare to PWM
   T2CONbits.ON = 1;
   OC1CONbits.ON = 1;        // turn on PWM
}
void halt(void) {
   PORTDCLR = 0x001E;        // turn both motors off
}
```

Servo Motor

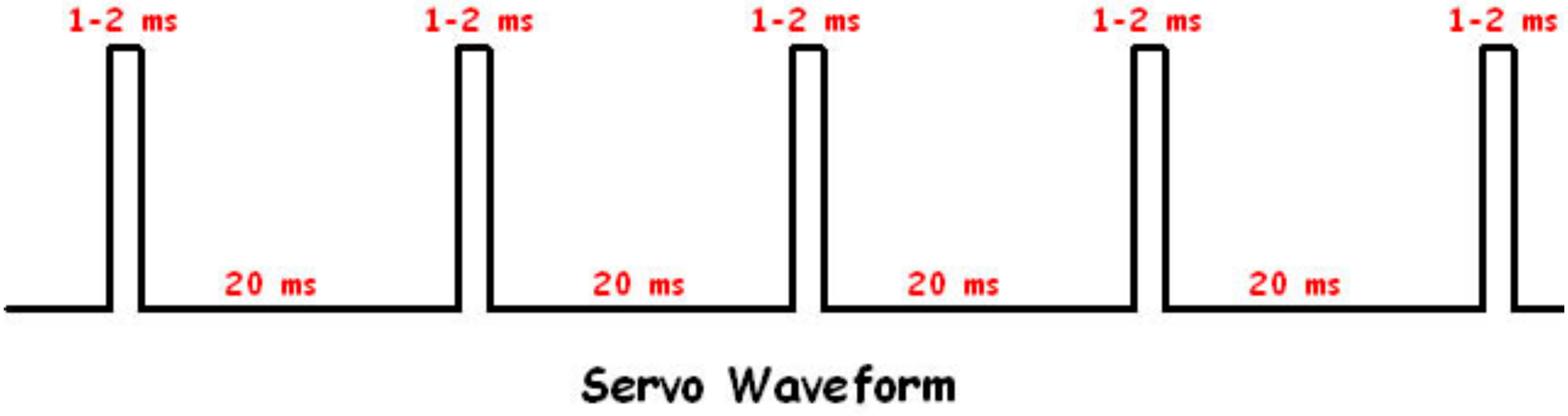
- DC Motor +
  - Gear train
  - Shaft encoder
  - Control logic
- Limited Rotation (180 Degrees)
- Pulse-Width Controlled

Servo Motor

- No need for an H-bridge
- Uses:
  - Model airplanes
  - Small robots
  - Kinectic sculptures
- Center red wire is usually power
- Black or brown is usually ground
- Datasheets hard to come by

Servo Control

- PWM (20ms Period is 50Hz)
  - 0.5 ms to 0 degrees
  - 1.5 ms to 90 degrees
  - 2.5 ms to 180 degrees



http://www.glacialwanderer.com/hobbyrobotics/?p=7

Servo PIC

```
#include <P32xxxx.h>

void initservo() {
        T2CONbits.TCKPS = 0b111;     // prescale by 256 to 78.125kHz
        PR2 = 1561;                  // set period to 1562 ticks = 50 Hz (20ms)
        OC1RS = 117;                 // set pulse width to 1.5 ms to center servo
        OC1CONbits.OCM = 0b110;      // set output compare 1 module to PWM mode
        T2CONbits.ON = 1;            // turn on timer 2
        OC1CONbits.ON = 1;           // turn on PWM
}

void setservo(int angle) {
        if (angle < 0) angle = 0;
        else if (angle > 180) angle = 180;

        OC1RS = 39+angle * 156.1/180; // vary from 0.5-2.5 ms
}
```

Stepper Motor

- Discrete steps based on pulses
- Usually a few degrees per step
- Precise position
- Continuous rotation

Stepper Motor C

```
#define STEPSIZE 7.5 // size of step in degrees
int curstepstate; // keep track of current stepper motor state

void initistepper(void) {
     TRISD = 0xFFE0;          // RD[4:0] as outputs
     curstepstate = 0;
     T1CONbits.ON = 0;        // turn timer off
     T1CONbits.TCKPS = 3;     // prescale by 256
}

void spinstepper( int dir, int steps, float rpm ) { // dir=0 for forward, 1 for reverse
     int sequence[4] = {0b00011, 0b01001, 0b00101, 0b10001}; // drive sequence
     int step;

     PR1 = (int)(20.0e6/(256*(360.0/STEPSIZE)*(rpm/60.0))); //time/step w/ 20MHz p clock
     TMR1 = 0;
     T1CONbits.ON = 1;
     for (step = 0; step < steps; step++) {
          PORTD = sequence[curstepstate];
          // determine next state
          if (dir == 0) curstepstate = (curstepstate+1) % 4;
          else curstepstate = (curstepstate + 3) % 4;
          while (!IFSObits.T1IF);                            // wait for timer to overflow
          IFSObits.T1IF = 0;                                 // clear overflow flag
     }
     T1CONbits.ON = 0;
}
```