RX62T

Brushed DC Motor Control with IR Compensation

R01AN0958EU100
Rev. 1.00
May 29, 2012

Introduction

Many applications require a simple bi-directional motor for moving material, people, etc. The simplest motor in many cases is the brushed DC motor. By combining the motor with a MOSFET H-bridge and a microcontroller with PWM timers, simple bi-directional control is realized. In addition, since the mathematical model of a DC motor is fairly simple, we can utilize a limited amount of CPU bandwidth to add a simple form of speed regulation, IR Compensation. This method adjusts the Motor Drive Voltage (PWM) by monitoring the current through the motor and adding back the voltage lost to the IR drop.

Target Device

RX62T

NOTE: Although this demonstration is show for the RX62T, this technique and much of the code can easily be ported to any Renesas MCU with a PWM timer and an ADC

Contents

1. Overview

In this application note we will show how a Brushed DC motors speed and direction can be controlled using the PWM from an MCU's timer. Along with this we will show a rudimentary form of speed regulation called IR Compensation. IR Compensation is basically positive feedback that causes the Motor Voltage to rise in response to increased load as indicated by an increase in motor current.

2. Brushed DC Motor Control

2.1 Motor Model

This paper is not intended to be a dissertation on DC Motor theory, but to reach our goal, we need to cover the basics of Brushed DC motors and their models. The user is given several references in the back so they can delve into the subject in more detail if desired.

Figure 1 shows a typical DC motor consisting of the Stator Magnets, the rotor (with its windings) and the brushes (the commutator).

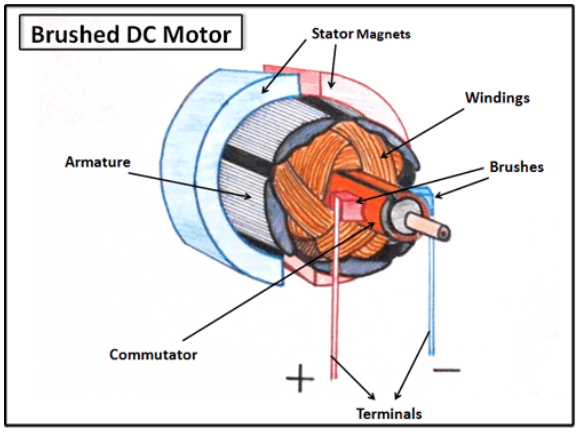

Figure 1: Typical Brushed DC Motor

As voltage s applied to the terminals, current (I) builds in the rotor windings producing flux. The brushes are set-up so that the flux produced is at the correct angle to the flux in the Stator Magnets that torque is produced causing the rotor to spin. At the proper time, the brushes switch the rotor current direction so that the correct angle is maintained relative to the stator flux thus producing a continuous torque so the rotor continues to spin.

This works very well, but as the motor turns, the flux from the stator cutting through the rotor coils produces an electromotive force (*emf*). This emf produces a torque that is counter to the torque being produced by the voltage applied to the motor. So the more we drive the motor, basically the more it resists us driving it. Let look at this in a little more detail.

Figure 2 shows the "basic model" of a Brushed DC motor.

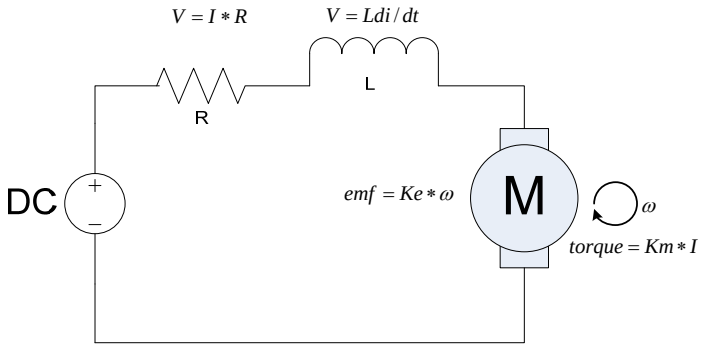

Figure 2: Basic Brushed DC Motor Model

So some basic terms:

K_m is the torque constant of the motor, usually in Newton-meters (Nm). Torque being produced is proportional to the current (I) being supplied to the motor as a function of this constant.

K_e is the emf constant of the motor. *emf* is the electromotive force (voltage) produced by the motor as it spins and it is proportional to the speed as function of K_e .

R is the resistance of the Motor, mostly rotor winding resistance, but inclusive of the brush resistance.

So torque T = Km*I and emf = Ke*ω. In terms of a simple analysis, the motor will speed up until the torque from our drive matches the torque produced by the emf, so for this analysis we can set Km=Ke. Based on Kirchoff's law, from the motor model we can now write the equation:

$$L\frac{dI}{dt} + RI = VDC - Ke \cdot \omega$$

We will look at the steady state motor (i.e. spinning at some speed ω), so we can ignore the time depended part, the Ldi/dt and reduce the equation to:

$$RI = VDC - Ke \cdot \omega$$

Re-arranging we get:

$$Ke \cdot \omega = VDC - RI$$

Equation 1 : Motor voltage as a function of speed

So if want a given speed, we just need to multiply Ke by the reference speed we desire and this will tell us the voltage we need to produce using our PWM H-Bridge. So one thing quickly jumps out at us, and that is that *I* is not constant. It will increase and decrease at the load increases and decreases, so we need to measure that and add or subtract as necessary for this IR drop to keep ω constant, or another way to say it is to "compensate" for the IR drop. Remember Ke=Km so to keep the torque constant we must maintain the current through the motor. As more voltage is dropped across the R, the more voltage drive we must provide.

If you look back at Equation 1, you might also say, "What happens when I push the motor and it becomes a generator?" When the current (I) decreases or even finally reverses, the IR term will become less and less and then finally reverse. Since we are driving the bridge in complementary mode, the IR factor will cause use to spend less time driving the motor and more time "braking" the motor so as to resist the "push", so again IR compensation is perfect for speed regulation in this case also. We show the drive and recirculation in Figure 6 and Figure 7

Finally we must consider a motor stall. If the motor stalls the current will quickly go beyond what the motor can withstand and it will be damaged (remember, when ω = 0, no *emf* is produced, so the only thing impeding the current flow is R which is typically very low). This over-current or stall condition must be handled in a properly designed system to prevent damage, so it is typical to use the current reading in software for compensation/control, but to connect it to some additional hardware to shut down the PWM in the event of an over-current condition. We will discuss this more in the Hardware and Software overview sections.

2.2 Hardware Overview

The motor is driven using 2 legs of the 3 phase drive in the RX62T demo kit. The hardware consists of gate drivers to convert the PWM from the MCU to adequate levels to drive N-Channel power MOSFETs, forming the basic H-Bridge for driving the motor. Since we are driving the bridge in complementary mode, we can provide 4 quadrant operation. By using a full H-Bridge, we can drive the motor in both directions by simply changing the duty cycle so that one side of the bridge is "higher" than the other. Figure 3 shows the typical hardware used to drive a DC motor.

The current can be monitored in 2 places, a low-side shunt, and a high-side shunt.

NOTE: The high-side monitor circuit is not designed into the standard RX62T demo kit and was added in order to facilitate this algorithm.

In addition to the current, we must monitor the Battery voltage so we can account for high and low battery conditions. In fact some motors can be damaged by higher than normal current when trying to run from low batteries. When running from batteries with a high charge, the motor may be allowed to exceed its rated speed. A simple voltage divider made of two resistors brings the 24V battery voltage down to a level usable by the ADC on the RX62T.

NOTE: When driven from an H-Bridge the maximum DC voltage applied to the motor is the battery voltage, so for example, if you had a 48V battery, and limited the duty cycle to 50%, the effective voltage is 24V, but the motor actually sees a switched 48V. Design needs to account for this in the insulation characteristics of the motor.

Finally, the reference speed (i.e. speed desired by user) is set using the potentiometer on the Demo board. The voltage on the wiper of the potentiometer is read by the ADC.

NOTE: The circuit show can monitor both the motor current and the low-side bus current. Typically high-side current monitoring will cost a little more, but it has the distinct advantage of always being able to read the current through the motor. Low-side shunt cannot read recirculation currents.

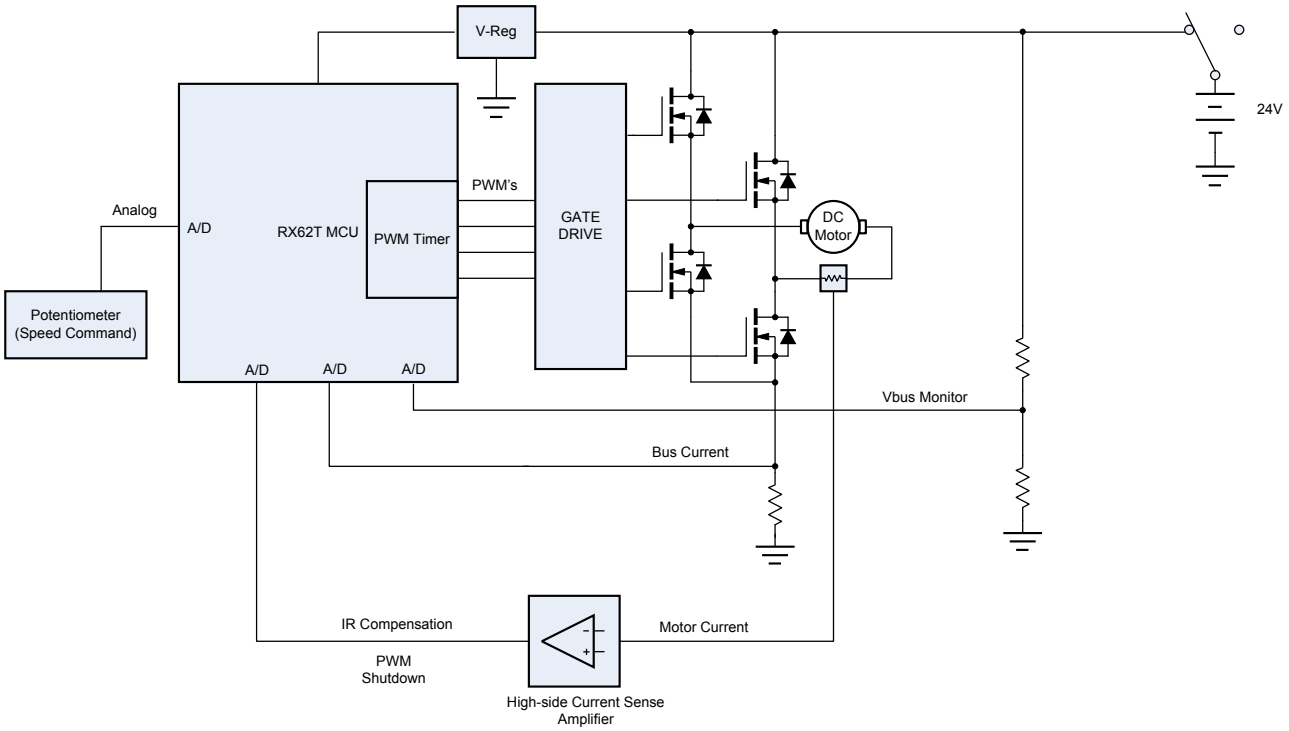

Figure 3: Typical DC Motor Drive Circuit

3. Software Description

So, now that we have seen the Motor Model and the hardware we can discuss how we use all this to control the motor. The code in the demo project is fairly well commented, so we will only try to hit the high points here in the application note. One of the nice things about the RX is that it has a floating point unit, so we can discuss all of this in straight, floating point mathematics, without have to go through all the explanation of why we might have chosen a particular fixed point resolution and the scaling that goes along with it.

NOTE: Some of this software is written in "multiple lines" for clarity (i.e. we do motor voltage and IR compensation as two separate lines to make it standout), you can easily optimize source file by combining lines.

3.1 customize.h

We will not dwell on this file too long, because as we stated earlier, we are using floating point arithmetic and we can describe our System in straight forward terms. It is important to note, that we will be using single precision operations and all parameters are using the "f" suffix (i.e. 1.23f or 1.0f, etc.)

The customize.h is basically divided in to three sections, the PWM definitions, the Inverter Hardware definitions and the Motor specific information.

3.1.1 PWM Definition

Only two items are referenced here, the carrier frequency and the dead-time (since we are using complementary drive). The carrier is listed as being in the range from 3,000 to 20,000, but can go higher due to the simple math done in this algorithm.

Note that the Carrier numbers are used in fix_par.h which relates them to timer and clock counts.

3.1.2 Inverter Hardware

This section defines the hardware and any scaling associated with it. The comments make the code fairly self-explanatory. We will use these values to calculate conversion constants (K) for the various ADC readings (See 0

ADC Scaling, Offset and Gain Correction for additional information).

```
/* MCU ADC definitions */
#define ADC_VREF        5.0f
#define ADC10_BIT_WGT  (ADC_VREF/1024.0f)   // in mV/bit
#define ADC12_BIT_WGT  (ADC_VREF/4096.0f)   // in mV/bit

/* Voltage divider for Monitoring VBus definitions */
#define VBUS_R1     47000.0f
#define VBUS_R2     10000.0f

/* Low-side shunt for Monitoring Motor current definitions */
/*  Low side shunts */
#define R_SHUNT_LO  0.100f              // 100 milli-ohms
#define I_AMP_GAIN  5.0f

/* High-side shunt for Monitoring Motor current definitions */
#define R_SHUNT_HI  0.050f              // 100 milli-ohms
#define I_AMP_GAIN_HIGHSIDE  50.0f      // LT1999 with 50 gain

/* dead-band in pot center to allow easier selection of 0 speed */
#define POT_DEADBAND    10.0f          // counts to ignore at middle of pot
```

3.1.3 Motor Definitions

The motor definitions are used to create the operational parameters for the motor. Some are used to calculate other operational constants dynamically. Most of these you can find in the data sheet for the motor. If the R of the windings is not given, you can get a good starting point by measuring it simply using a meter, then adjust up and down as you tune the motor dynamically (See section 4 Hints, Tips, and Tricks).

```
#define MOTOR_R          10.0f           // measured
#define MOTOR_VBEMF      24.0f
#define MOTOR_Ke        (24.0f/135.0f)   // volts-per-RPM (24/135)

/* System Parameters */
#define V_BUS_MAX       26.0f
#define V_BUS_MIN       22.0f
#define I_BUS_MAX       0.400f

// 24V Demo motor parameters
#define V_MOTOR_MAX     24.0f
#define V_MOTOR_MIN     22.0f
#define I_MOTOR_MAX     0.500f
#define MAX_RPM         135.0f     // Based on Motor
#define MIN_RPM         10.0f      // User selected based on requirements
#define K_RAMP          10.0f      // Ramp Constant in RPM/seconds
```

3.2 ADC Scaling, Offset and Gain Correction

The ADC scaling is handled by values in the customize.h file. Some of the constants are used directly in the calculations, while other are used to calculate constants (K) values at run time. In the function *InitMotorEnvironment()*, we calculate three constants (K) for use by the program when reading the ADC channels. Basically whenever an ADC value is read, you can multiply by the constants to get a direct conversion to a "real" value (Volts, Amps, etc.).

k_vbus – Used to scale ADC reading of VBus into VOLTS

k_Ishunt – Used to scale ADC reading of low-side shunt voltage in CURRENT (includes amplifier gain)

k_Ihigh_sense – Used to scale ADC reading of high-side shunt voltage in CURRENT (includes amplifier gain)

These values are then directly used in the code to convert as follows:

```
v_bus = ADC_correction * (adc12_results[3] * k_vbus);

i_M = ADC_correction * ((k_Ihigh_sense * (float) ((int16_t) adc12_results[6]
      - i_M_offset)) );
```

So if we are doing a straight forward calculation, you might ask " what is that ADC_correction?". ADCs are subject to various tolerance and errors. The simplest to correct is for ADC Gain.

In this hardware platform we have a very accurate 4.25V reference on channel 0 of the 10 bit ADC. Given that the ADC blocks all work from AVREF and AVCC, we can measure this, knowing it should be exactly 4.25V and based on the ADC value we read for it relative to v_ref we can come up with a Gain correct value that we can apply to all ADC readings.

```
ADC_correction = 4.25f / v_ref;         // make a gain correction value
```

Finally, since the ADC only reads positive values and the Motor currents will typically be both positive and negative, we must have some way of converting these signals so the ADC can handle them. This is typically done by offsetting the output of the amplifier to ½ VRef at 0 motor current. This value may not be accurate, so it is easier to take several readings with the motor current at zero (I=0), average them and then store this "zero current" offset in RAM for use during the current calculations. So when we "start up", we allow the Motor timer to tick the ADC 8 times, then we average the readings and store them away.

```
while (start_up_cnt1 !=0){};
i_v_offset /= 8;                // and average to get the offset
i_u_offset /= 8;                // and average to get the offset
i_M_offset /= 8;                // and average to get the offset
```

Once we have these offsets, they can be used, along with the Gain correction value to calculate the actual Motor Current.

```
i_M = ADC_correction * ((k_Ihigh_sense * (float) ((int16_t) adc12_results[6]
      - i M offset)) );
```

So just to review, we take the reading from the ADC, and subtract the offset give in us a bipolar current reading (can be either + and -). We then multiply this by the conversion constant for the Current High Side sensor and finally the Gain correction to give us a Motor Current (i_M) in amperes.

3.3 Hardware Over-current Detection

The over-current monitoring is handled by the internal comparators on the RX62T. The small voltage developed by the motor current flowing through the shunt is converted to a voltage of suitable level by the LT1999 high-side current sensor as shown in Figure 3. This sensor is set-up so that 0 current is at mid-point ($V_{REF}/2$). Since the motor drive support bi-directionality, we must use the comparator in "window mode" so we can detect over-current in both directions. Since this motor is rather small, the current is small, so we set the voltage "trip level" to be 1/8 above and below the center point. Figure 4 shows the operation of motor current versus time. So the motor starts up at time t(0). The motor current varies as a function of load and speed. If at any time, the motor current, as represented by the voltage on the ADC pin, exceeds the window limits, either high or low, for more than 16 PCLKs, the window comparator is set to trip the POE logic internally and thus shutdown the PWM signals. Since the PWM signals are "pulled" to an inactive state for the inverter, the drive is disabled to the Motor.

NOTE: in cases where the system may have lots of inertia causing the motor to continue to spin, the motor acts like a generator and can cause a "pump up" of VBus. The designer should evaluate his needs for active or passive braking to dissipate this energy.

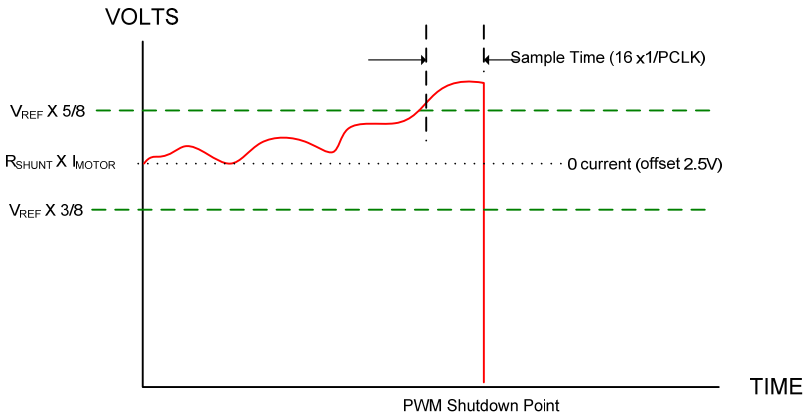

Figure 4: Over-current detection

3.4 Motor Voltage Calculations

Since VBus in this basic design will typically be a battery (or in some cases a low-cost power supply), the regulation or value of VBus may not be constant, as shown in Figure 3. We can add a simple resistor divider network to the VBus (battery voltage line in this case) and monitor its value. This allows us to add the following functions:

- High Battery detect
- Low battery detect
- Motor voltage correction for consistent drive

With these features we can make our product safer and prolong the life of the product. For example, if the product uses regenerative braking to recharge the battery, we can detect over-charge, over current. If the battery becomes too low and we apply this to the motor, it could mean higher than normal currents when driving the motor. By not allowing this to occur we can prolong the life of the motor.

In addition, if we do *not* monitor the VBus and correct the drive voltage accordingly, our IR compensation could become unstable (i.e. the algorithm trying to drive to a voltage beyond what is available on VBus).

So let's take a simple look how this operates. Figure 5 shows VBus versus time as the motor control operates. The blue line represents the battery voltage over time. So at t(0) we start with a freshly charged battery. In this case we have defined the VBus maximum (*VBUS_MAX*) to be 26 VDC. If at any time the VBus goes over this, we simply set a flag (*v_bus_err*) to indicate this VBus error.

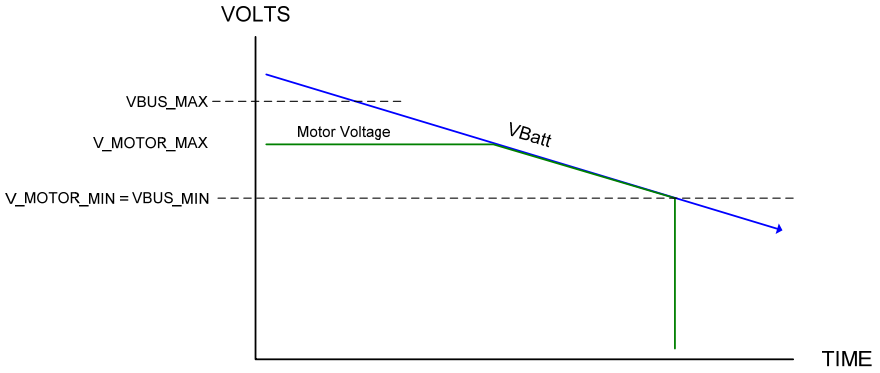

Figure 5: VMotor versus VBus

So we start the motor at maximum speed, that is we would like to apply maximum motor voltage as defined in motor specification, for example in this application, it is a 24VDC motor so we set VMOTOR_MAX at 24.0f in the customize.h So as the motor runs at maximum, we will hold at 24VDC. Since we have 26 VDC available, we must set the motor as a function of the voltage available (i.e. motor voltage is a function of the ratio of desired voltage versus available voltage).

First, we calculate the commanded Motor Voltage as a function of the reference speed

```
V_motor_commanded =  MOTOR_Ke * ref_speed;
```

Next, if compensation is on (IR_Comp_on == TRUE), we need to add back the amount dropped on the R of the motor:

```
V_Motor_Drive = V_motor_commanded + (I_Motor * MOTOR_R);
```

If you refer back to Figure 5, you can see at some point, the desired Motor Drive voltage will be higher than the available VBus. So we are tracking VBus down and limiting the Motor Drive voltage to the available VBus (i.e we limit the Motor Drive to whatever is available from the battery to avoid unstable IR compensation).

```
    if ( V_Motor_Drive > max_v_avail)
    {
      V_Motor_Drive = max_v_avail;
    }

    if ( V_Motor_Drive < -max_v_avail)
    {
      V_Motor_Drive = -max_v_avail;
    }
```

Finally, we need to convert to PWM count value and at the same time we will make the value a function of VBus:

```
V_motor_cnt = (int16_t) (V_Motor_Drive/v_bus * ((float) CARRIER_CONSTANT_2));
```

You can see that the PWM count is a function of the CARRIER_CONSTANT_2 which is ½ of the full carrier constant. So aside from Dead-time, this represents our maximum PWM value. Once we have the count, we determine whether U (+Motor drive) is bigger than V (-Motor drive) or vice versa depending on the sign of V_motor_count. This determines the direction so we calculate the final U and V count to drive the motor in the correct direction. If you look at the code, you can see we start with ¼ of the carrier constant which represents 50% duty cycle. If we are going CW, we add ½ of the count to U and subtract ½ of the count from V making U > V and thus driving the motor forward with the calculated V_Motor_Drive.

```
    if (0 > V_motor_cnt)
    {
        V_motor_cnt = (int16_t)-V_motor_cnt;
        pwm_u = (uint16_t)(CARRIER_CONSTANT_4 - (V_motor_cnt/2));
        pwm_v = (uint16_t)(CARRIER_CONSTANT_4 + (V_motor_cnt/2));
    }
    else
    {
        pwm_u = (uint16_t)(CARRIER_CONSTANT_4 + (V_motor_cnt/2));
        pwm_v = (uint16_t)(CARRIER_CONSTANT_4 - (V_motor_cnt/2));
    }
```

NOTE: For VBus error we do nothing except set flag. Designer must decide what to do at this point based on their individual system requirements.

3.5 PWM Drive

So we have seen how we calculate this, now let's take a quick look at the resulting motor waveform and resulting motor voltage. Figure 6 shows the actual PWM drive at the motor for CW operation. If you look at the actual Motor Drive part, you can see we are driving about 20 µS or about 35% of the time, this would equate to about 24V*.3 or 30% of the available drive voltage. Since the + side is driven longer this equates to positive drive (CW direction for the motor reference in this application note). Note that since we are using complementary drive, during the re-circulation the motor drive is ***not*** stopped, the motor is just briefly shorted out by either the high-side MOSFETs or the low-side MOSFETs. This basically creates re-generative braking in the motor, as less motor drive voltage is generated (i.e. motor being pushed by the load for example), more re-generative braking is created and the motor maintains speed. In fact, if the motor is being pushed fast enough the current will reverse and negative motor voltage (additional braking) will be produced.

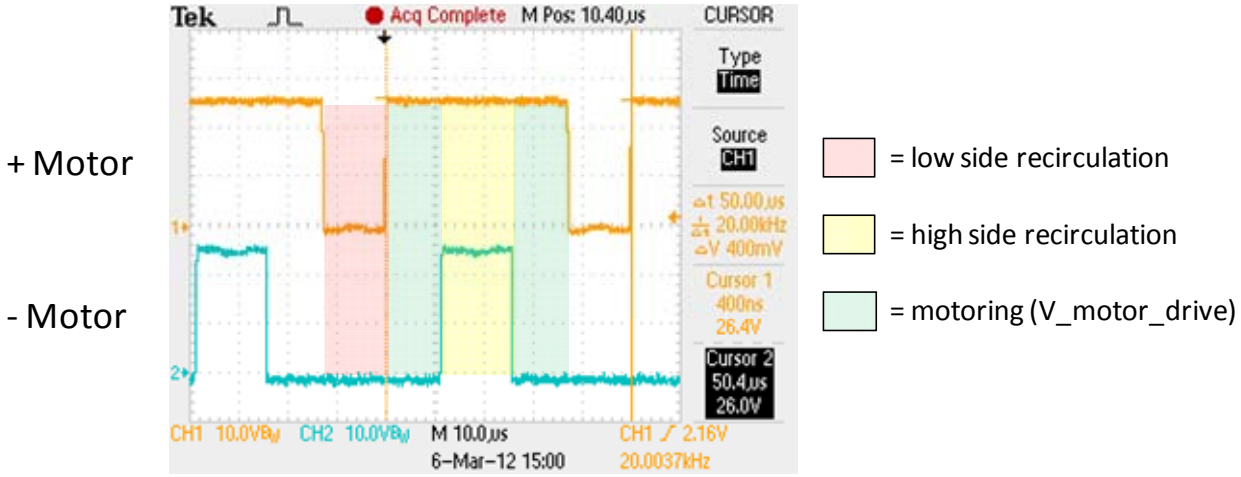

Figure 6: CW Motor Drive

For comparison, shows the motor being driven in a CCW direction. Note that in this situation, the resulting motor voltage gives us a longer period of drive on the "-" side of the motor.

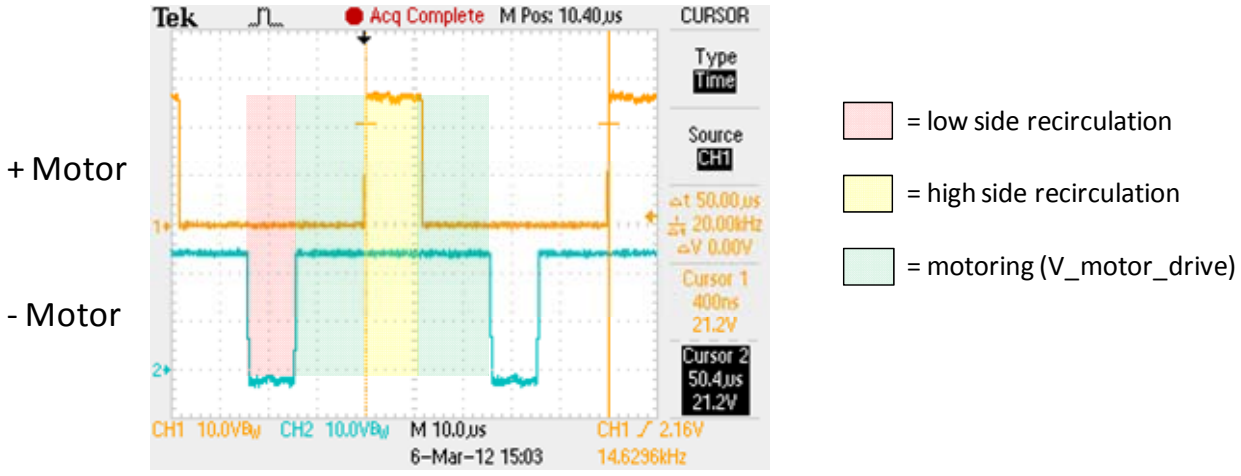

Figure 7: CCW Motor Drive

3.6 Motor State Machine

Now that we've looked into how the motor is driven, let's take one step up in our demo software and take a quick look at the simple State machine that determines how the Motor is controlled from a more abstract level.

If you look at Figure 8, it shows a very simple state machine that monitors the motor status and UI to determine operation.

We start out in the ERROR state to avoid the motor from Running before commanded (usually an undesirable characteristic), that is the motor cannot just "jump" when power is turned on. To leave the ERROR state in our demo, a speed of 0 must be commanded (i.e. positive command from User).

When in the START state and we see from the UI (or could be from some other control mechanism) that we have a non-zero commanded speed we move to START state.

The START state is a "transient" state that is used for any "housekeeping" that may be required before the motor actually moves. This might be anything from removing a brake signal to providing time to charge the bootstrap capacitors (as we show in the example). In our demo, the state machine is allowed to transition as fast we get back to it from the while(1) loop in main(). Transition could also be controlled by a clock or other mechanism, but it is not our intention to go into state machine theory in this application note.

So we transition to the RAMP state. The RAMP state is when there is a commanded speed from the user, but it does not match the reference speed of the motor control. The RAMP is based on definitions in the customize.h file. The reference speed is incremented or decremented until it matches the commanded speed. This is actually done in the motor control interrupt at a fixed rate (i.e. it is basically small steps based on the ramp value and the carrier rate of the motor timer). This allows smooth operation as defined by the user. In this case we only support a linear (sometimes called trapezoidal) ramp profile. Any change of speed requested by the user must go through the RAMP state to provide smooth transitions in motor speed, even when going from full forward to full reverse. Once the commanded and the reference speed match we go to the RUN state.

In the Run state we monitor the commanded speed. If it ever changes where it does not match the reference speed we move back to the RAMP state, again to avoid abrupt changes in motor speed. Even if we ask for speed of 0, it will be done by ramping, it will return to the RUN state where it will see we have "stabilized" at a speed of 0 for both commanded and reference, where it will go back to the STOP state.

If at any time the MCU detects an Over-current condition, it moves to the ERROR state. Exit from the ERROR state is possible if the UI commands a speed of 0. Of course your application may have more complex tasks in the event of an error, but this a very small motor in a demonstration application.

We have made provision for a BRAKE state, but we do not do any braking. Braking may be required in systems with large inertia where the energy from the spinning motor must be dissipated so that it does not over-charge the bus. In your system, the energy may be re-captured to use in charging the batteries or some other "green" energy feature.

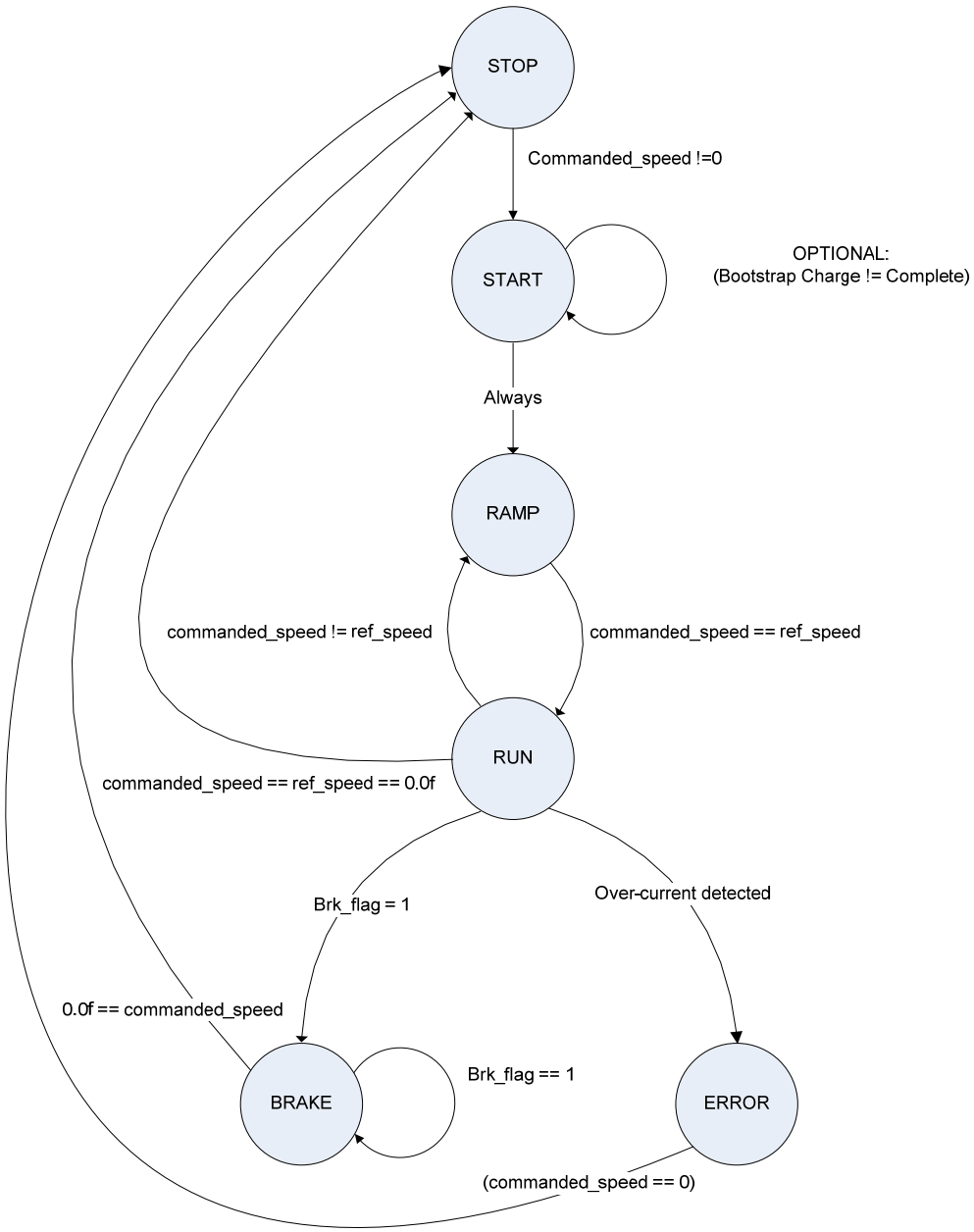

Figure 8: Motor State Machine

4. Hints, Tips, and Tricks

4.1 Tuning

This will not be a long dissertation in tuning Motor Drives, only a simple advisory.

IR compensation is basically a positive feedback system, the more you load, the more voltage is applied/generated. You can see that in many cases this can cause issues. In the case of IR compensation, the R can be thought of as the "gain". The bigger the R you put in your definitions file, the more the algorithm will add back the applied motor voltage to maintain. When the "gain" is too high, you may get the resulting "chatter" from Brushed DC motor drives that the compensation is set incorrectly. Many white papers can be found on the web for IR compensation adjustments, some of the best coming from Motor Control System manufacturers.

4.2 Instability

There are other causes on instability not related to "tuning" the IR compensation. One example is allowing the defined maximum voltage of the motor to exceed the available voltage on VBus. If you remember back to section 3.4 on Motor Voltage Calculations, after VBus gets at or below the Motors Maximum voltage, we track it down to motor minimum voltage at which point we flag this and stop the PWM. If we did not do this, the Algorithm would continuously try to raise the Motor voltage up to a point that it could not (i.e. if it had 22V on VBus, but it needs to command 24V to get to the speed desired).

4.3 Low-side Shunt operation

This demonstration code can run from the low-side shunt only, but extensive testing has not been done in this mode. To run form the low-side shunt, set thedefine in the HEW build dialog C/C++ in defines from USE_ADC1 to USE_ADC0.

5. Limitations of Testing

The software in this product has undergone *limited* testing (enough to demonstrate IR compensation). It uses minimalist drivers and simple state machines to implement the demonstration code, and is not intended for "production" as provided.

The Motor algorithm has been tested using limited frictional loads, but dynamometer testing has *NOT* been performed to determine stability and accuracy of the speed of the motor under all load conditions.

6. References

6.1 External

Brushed DC electric Motor, Wiki - http://en.wikipedia.org/wiki/Brushed_DC_electric_motor

Joliet Technolgies, DC Drive Fundamentals - http://www.joliettech.com/dc_drive_fundamentals.htm#ir-comp

eCircuit DC Motor Model - http://www.ecircuitcenter.com/circuits/dc_motor_model/dcmotor_model.htm

Carnegie Melon Control Tutorials for Matlab, Example DC Motor Speed Modeling

http://www.engin.umich.edu/group/ctm/examples/motor/motor.html

Linear Technologies High Side Current Sense Amplifier LT1999 Data Sheet

http://www.np.edu.sg/alpha/nbk/alphastu/DCMotor.html

Autotrol Motor 220-0106A, Mouser # 708-2200106A, http://www.autotrol.com/

6.2 Renesas

RX62T Motor Demonstration Kit Schematics

RX62T Hardware Manual, R01UH0034EJ0110 Rev.1.10

7. Glossary

Bootstrap capacitor – a small capacitor used to provide the gate voltage on the high-side gate drivers so that additional power supply is not needed.

Re-circulation – a condition where either both high-side MOSFETs or both low-side MOSFETs are on, in effect shorting the motor and allowing any current within the motor to re-circulate.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	May.29.12	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   - "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.1