# CM3 Revision 1

Khanh J. Phan

Last Updated: December 14, 2014

## Contents

## 1 Introduction

The proposed motor controller will use an AVR microcontroller to interface with the power stages and user inputs. The specific AVR microcontroller used is the ATmega328P-PU 28 Pin PDIP.

The firmware will consist of interrupt-based programming as opposed to polling. A single external-pin interrupt will be used to indicate to the AVR C that a hall signal has been received. The corresponding ISR will then look-up the appropriate commutation signal to commutate the motor.

The reference speed is calculated from a ADC interrupt handler which will set the appropriate duty cycle.

The measured speed can be calculated by timing the interval between two hall interrupts (and knowing which interrupts are read). Similarly, the direction in which the motor is spinning can be determined by confirming that the expected hall signals are received.

## 2 Firmware

The following interface functions are proposed for implementation in revision 1:

### 2.1 Initialization Functions

```
//Initialize the internal clock
void clock_init()
{
    //Oscillator
    //Prescaling
}

//Initialize the motor controller pinouts
```

```
void mc_init()
{
    //Power bridge connections (6 pins)
    //Hall connections (3 pins)
    //Timer
    //ADC
}
```

## 2.2 Regulation Functions

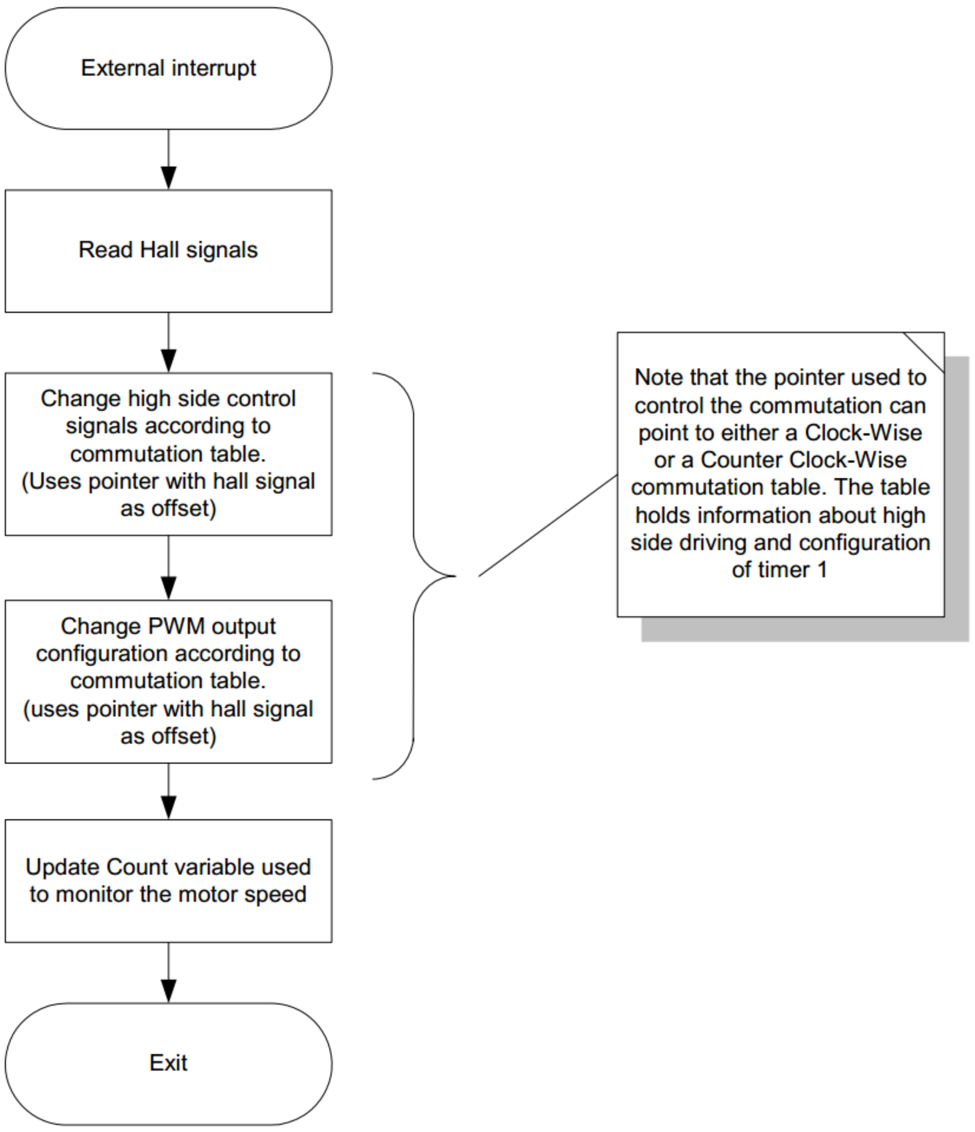
```
void mc_start (void)
{
    //Used to start the motor. The regulation loop is launched to set the duty
    //cycle. Then the first phase commutation is executed
}

void mc_stop (void)
{
    //Used to stop the motor by turning off the appropriate register pins
}

bool mc_isMotorRunning (void)
{
    //Returns the motor state, 'TRUE' for running and 'FALSE' for stopped
}

bool mc_isDirFwd (void)
{
    //Returns 'TRUE' for correct direction and 'FALSE' elsewise
}

void mc_refSpeedSet (uint8_t speed)
{
    //Set the reference speed for the motor controller to adjust to
    //Update the output compare registers of timer which controls the duty cycle
    //of the PWM output, thus the speed of the motor.
}

uint8_t mc_refSpeedGet (void)
{
    //Returns the reference speed
}

uint8_t mc_measuredSpeedSet (uint8_t measured_speed)
{
    //Saves the measured speed in the measured_speed variable
}

uint8_t mc_measuredSpeedGet (void)
{
    //Gets the measured speed.
}
```

## 2.3 Commutation

```
void mc_commute (void)
{
    //Update the PWM outputs depending on the hall signals read
}
```

## 2.4 Reading Hall Signals

**Figure 2-5.** Flowchart of the external interrupt handling the commutation


External interrupt
Read Hall signals
Change high side control signals according to commutation table. (Uses pointer with hall signal as offset)
Change PWM output configuration according to commutation table. (uses pointer with hall signal as offset)
Update Count variable used to monitor the motor speed
Exit
Note that the pointer used to control the commutation can point to either a Clock-Wise or a Counter Clock-Wise commutation table. The table holds information about high side driving and configuration of timer 1