Motor Control Using a 32-bit Cortex-M3 MCU

Paul Kimelman - CTO
Luminary Micro, Inc

Contents

- Electronic Motor control: What and Why?
- What are the kinds of motors?
- What is PWM and Closed loop control?
- How do I wire these up?
- What are the algorithms?
- What are the advantages and disadvantages of each?
- Why Cortex-M3 makes this easier
- How you can mix motor control with your application

Why Electronic Motor control?

- Varies by application
  - Improved efficiency (power use)
  - Improved torque (affects efficiency)
  - Control of speed
    - Adapt to load
  - Control of acceleration/deceleration
  - Precise control of position
  - Detect faults and stalls – not just overheat based
- Required for new styles of motors
  - Many are like old style, but need input not naturally found

What is Motor control?

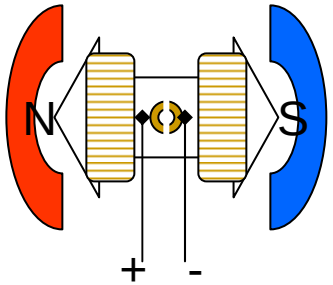
- Control of current to windings of a motor
  - It is all about current (in correct winding)
  - PWM used to drive current using on/off pulses
- Open loop – no feedback from motor
  - Feed the current when motor *should* need it for given speed
    - Rotor may get behind or stall – no way to know
- Closed loop – feedback from motor or drive circuit
  - Adjust input based on feedback
    - Maintain speed and manage acceleration/deceleration
    - Detect stall and fault – shutdown cleanly
  - Minimally: measure current into windings to understand what motor is doing
    - Current draw tells a lot about motor behavior
    - Back-EMF is winding acting as a generator when allowed to float
  - Maximally: get position information
    - *Absolute* or incremental, Hall effect or physical (ie. light strobe)
    - The more positions the better (but higher bandwidth input)
    - Two encoders 90° (quadrature) apart gets position and direction

Motor history

- Three main types historically
  - Brushed DC and Universal
  - AC 3-phase
    - Induction and Synchronous
    - Rely on 3-phase AC
  - AC 1-phase
- Speed control often unavailable or crude
  - Speed control of AC tended to be very expensive

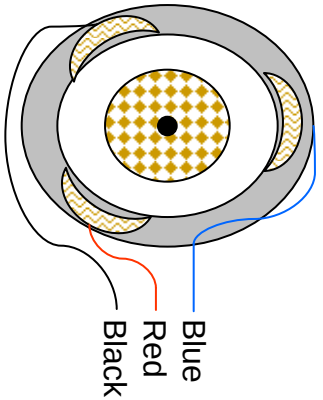
DC brushed motors

- Commutator and brushes
  - Current reverses at 180° of rotation
  - Normally, electromagnet spins (rotor)
    - Windings are around the armatures
    - 3 poles of armature is most common, to prevent stalling
  - Normally, permanent magnets stationary (stator)
- Speed defined by voltage
  - “Chopping” most common variable speed control
    - On/off pulses – defines apparent voltage
- Torque defined by current
  - Overcomes resistance to motion
- Lower efficiency
  - Brushes limit current and voltage, lossy
- Universal motors are a variation that works with AC
  - Universal motors get two-level speed control by pruning AC (one diode)



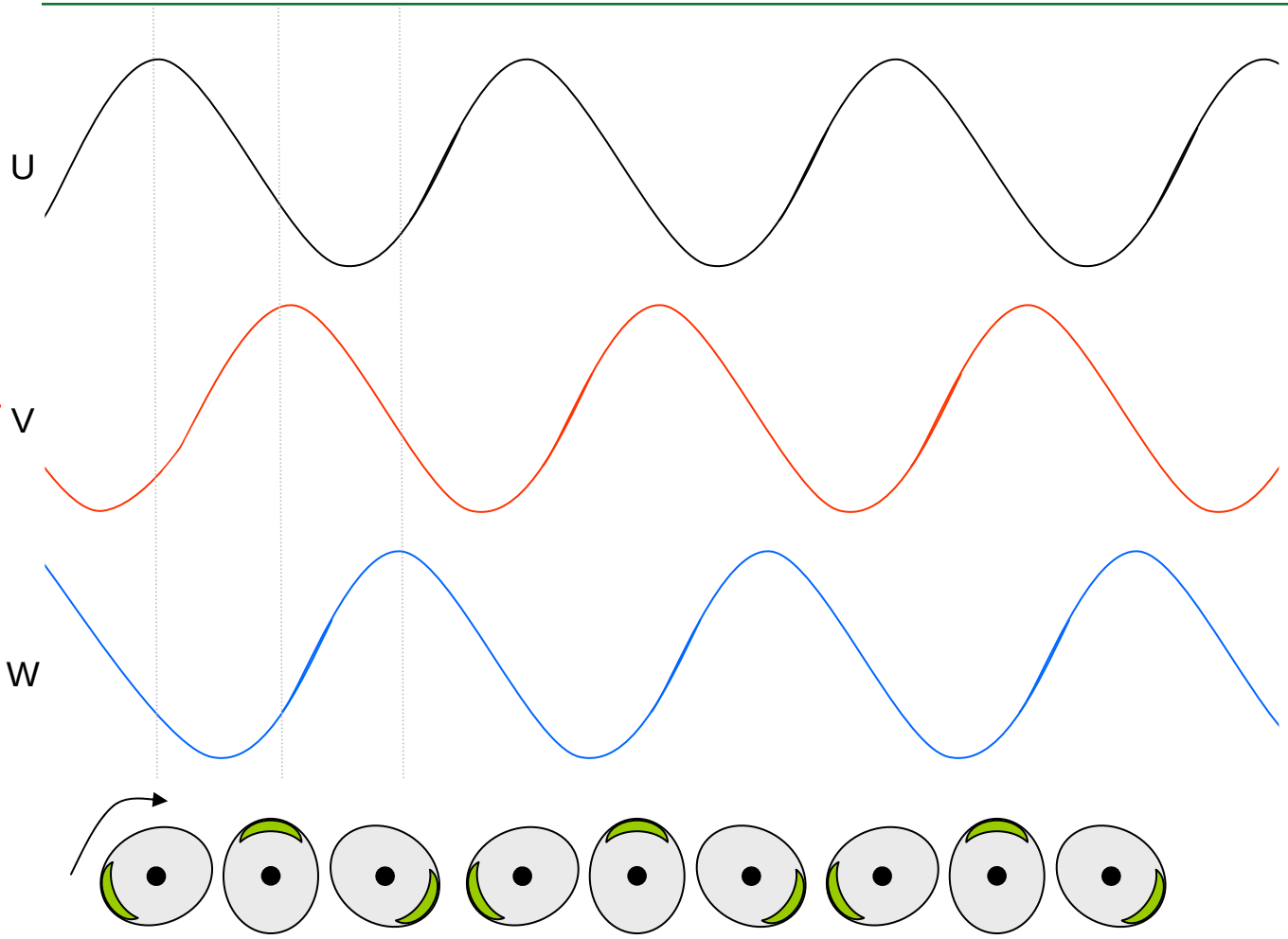
AC 3-phase

- Normal Design matches 3 phase AC into building
  - Windings are fixed (stator)
  - 3 sets of windings receive current 120° out of phase from each other
    - Forms a circle of magnetism
- Synchronous rotor style
  - Permanent magnets or continuously powered electromagnet (through slip rings)
    - Follows the charged windings – synchronously
    - Speed normally set by frequency
- Induction rotor style
  - Metal conductors in a “squirrel cage”
    - Picks up field from windings, which induces a current in conductors
    - Current in conductors creates magnetism: follows the charged windings
    - Not synchronous: if catches up, loses current, so “slips” (stays behind)
- Variable speed control is complex with fixed frequency
  - Triacs or through slip rings – loses torque quickly



Cross section of Squirrel cage rotor in 3-phase stator

3-phase concept



Stator fields around the circle (120° apart, as generalized sum of 3 windings)

UUU VVV WWW UUU VVV WWW UUU VVV WWW

AC 1-phase

- Most common is “shaded pole”
  - One main winding
  - Uses simple copper loops to delay phase
    - Loops used to start
    - Position of two lengths of shading sets direction
  - Very low torque – used for fans and other low load uses
- Other 1-phase motors create phases
  - Capacitor is most common
  - Starting is hardest part

Contents

- Electronic Motor control: What and Why?
- What are the kinds of motors?
- What is PWM and Closed loop control?
- How do I wire these up?
- What are the algorithms?
- What are the advantages and disadvantages of each?
- Why Cortex-M3 makes this easier
- How you can mix motor control with your application

What are the types of Motors now?

- 6 Basic kinds:
  - Brushed (traditional)
  - Brushless DC (BLDC)
    - Basically an AC 3-phase synchronous
  - Stepper
    - Specialist BLDC motor with *toothed* rotor/stator
    - Teeth form “steps” that windings can attract
  - Switched reluctance
    - A mix of BLDC and unipolar Stepper
  - AC 3-phase Inductor and synchronous
  - AC 1-phase (e.g. shaded pole)

Brushed DC motors

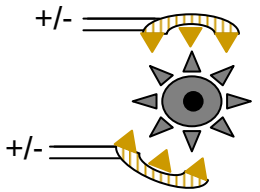
- Electronics just vary apparent Voltage
  - Chopping (PWM) is normal technique
    - Average voltage is controlled by on vs. off time
    - Noise reduced through filtering or higher frequency pulses
    - Need diode protection from winding discharge
  - Chopping gives highest current (torque)
    - Higher voltage used to charge windings more quickly
  - Can determine position and vary voltage/current
    - Current useless when between poles
  - Can never make as efficient as BLDC/AC
- Universal motors are not well suited for Electronics
  - Normal approach is DC bridge rectifier and treat as DC
  - Do not do well at lower speeds

Brushless DC (BLDC)

- Looks similar to a 3-phase AC synchronous motor
  - Normally 3 sets of windings (some smaller ones use 2 sets)
    - Normally power applied to 2 of 3 leads (or all to one and ½ to two)
  - Permanent magnet rotor (inside, outside, or parallel to stator)
  - Normally adds position sensor of one form or another
    - Hall sensor or Shunt to measure current flow
    - Incremental or absolute position encoder for position/direction
- Electronics sequence current to each winding set
  - Effectively simulating 3-phase AC, but only powers 2 legs
    - 2 legs in opposition (one positive, one negative)
    - Chopping (PWM) applied within sequence to get *effect* of sinusoid
    - Speed determined by frequency (sequencing)
    - Open loop control is problematic if a load on shaft
  - Torque determined by current (high voltage pulses get same effect)
  - Smoothness determined by apparent voltage matching speed

Stepper motors

- Similar to BLDC in stator side (but 2 windings)
  - Permanent magnet rotor teeth attract to winding's teeth
  - Precise steps based on number of teeth/alignment
    - Dozens to hundreds of teeth
  - Can hold a stopped position at tooth
    - Also ½ step position (between two)
    - Strength of hold is based on current
  - Far more sequences than BLDC for one rotation
    - Based on number of teeth/alignments
    - Considerably more processing power needed
    - Bipolar is most common: 2 pairs of wires
      - 4 states: forward/reverse polarity on each winding
      - ½ steps and micro steps by charging both windings
    - Unipolar is 6 wires (although commons usually joined, so 5)
      - No reversing, just drive one of each pair



Switched reluctance motors

- Mix of BLDC and Unipolar Stepper, but often large motors
  - Toothed rotor, but small number of teeth with strong magnets
  - 3 phased stator, but windings are not connected
    - 3 pairs of wires (for three windings)
    - Need special 3 legged half-bridge
    - Windings are only charged in one direction
- Inductance is near linear from min to max
  - Minimum when rotor tooth is farthest
  - Maximum when rotor tooth is aligned
  - Usually referred to as “triangle” inductance
- Generally need position feedback to make smooth
  - Sensorless done using current measurement
    - Works because of inductance profile

AC 3-phase

- Electronics create AC *equivalent* using PWM
  - So, AC in is converted to DC using bridge and caps 1st
  - PFC (active or passive) may be added if large motor
- AC induction is most common
  - Far more complex algorithms
    - Must compute speed of rotor and its flux
      - Flux is amount of induced charge
    - Cannot let rotor catch up or get too far behind
      - Control of slip is essential
    - Field Oriented Control (FOC) calculates position
      - Uses known constants for specific motor (you work out once)
      - Sinusoids computed to keep rotor at optimum slip
        - Requires feedback (closed loop) to verify
  - AC Induction rotor has no natural field, so active braking needed
- AC synchronous is basically same as BLDC

AC 1-phase

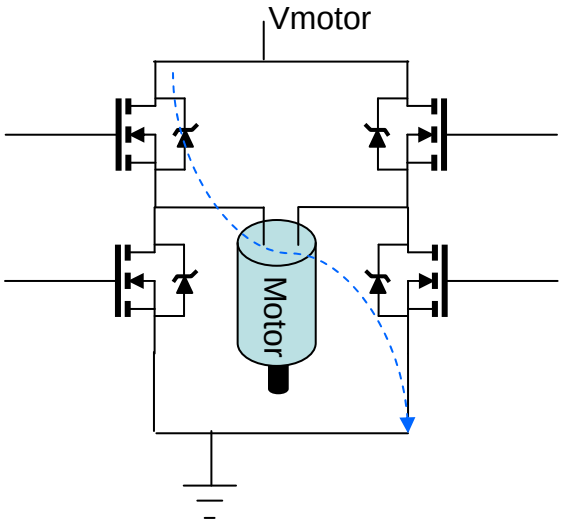
- Electronic speed control creates AC
  - Inverter using PWM most common
  - Creates frequency to control speed
    - Most motors do badly at lower and higher speeds
      - Property of phase shift (of shading or equivalent)
      - Vibration and heat can be a problem

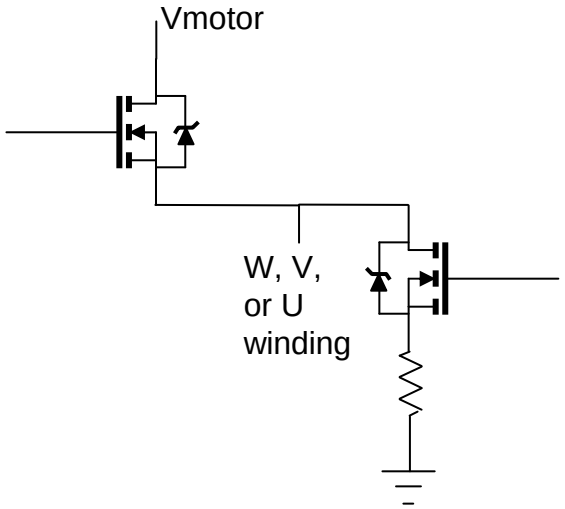
Contents

- Electronic Motor control: What and Why?
- What are the kinds of motors?
- What is PWM and Closed loop control?
- How do I wire these up?
- What are the algorithms?
- What are the advantages and disadvantages of each?
- Why Cortex-M3 makes this easier
- How you can mix motor control with your application

The bridge is the key to control

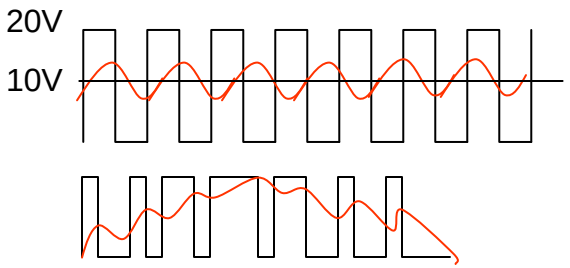
- DC H-bridge supports both directions of current flow
  - High to low in either direction
  - Uses fast switching MOSFETs
  - Diode protection for back-EMF and inductance
  - Need two H-bridges for bipolar steppers
- 3 leg Half Bridges for 3-phase
  - Windings connected together
  - Cannot have both transistors on - short
  - Higher voltage uses IGBT vs. MOSFET
  - 3-phases needs 3 half bridges
    - For each of U, V, and W winding





Chopper/PWM

- A chopper algorithm uses a PWM to create average voltage
  - The normal Chopper is used with DC motors to create the apparent voltage (across the windings)
    - Apparent/average voltage defines speed of rotation
  - The higher pulsed voltage of a chopper means more current per pulse
    - Current defines torque
- Example:
  - 50/50 duty cycle of 20V looks like 10V
  - Varied duty cycle can be made to look *like* a sine wave
    - More pulses makes more like sine wave



- Faster pulsing makes smoother
  - Smoother avoids noise and lowers heat
  - Can adjust based on feedback – adjusts to load

PWM applied to phases

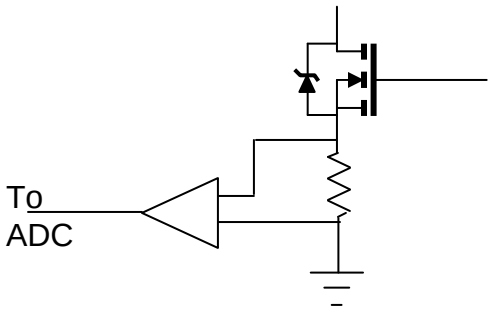
- PWM is then applied in sequence
  - Apply power to one leg
  - Other legs are unpowered, reverse powered, or some combination
- For bipolar Stepper motors, chopping applied to 4 states
  - Per winding: Forward, reverse
  - Different algorithms trade off torque and smoothness
  - Also, hold (in position)
- AC Induction and BLDC prefer sine waves
  - PWM to produce sine wave equivalent (approximation is enough)
  - Use feedback to make sure rotor is where it should be

Gate Drivers

- Gate drivers map from MCU PWM output to MOSFET/IGBT
  - Voltage matching and isolation
    - MCU uses normal output voltage – gate driver ups to match gate
    - MCU is on isolated power circuit to avoid noise issues
  - Short circuit protection – maybe
    - MCU algorithm should prevent turning on high/low same side
  - Dead-time for shoot-through protection
    - MCU or Gate drivers needs to allow for transistor shutoff timing
    - Preferred to have this in PWM (MCU) so can be tuned easily
  - Possible fault detection – over-current
    - Gate driver or MCU (ADC) may detect over-current
    - Important to protect motor
    - Stalled motor may draw way too much current

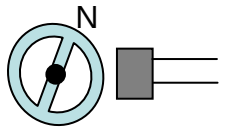
Closed loop detection

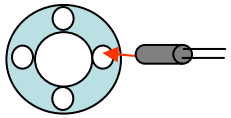
- Current sense circuit used to detect winding uptake
  - Shunt is normally used off low side
    - Prevent over-current
    - Control PWM more optimally
    - Rough close-loop model if no position sense
  - Hall sensor may also be used
- Back-EMF (sensorless)
  - Measure output of currently un-powered winding
  - Winding acts like a generator
    - Result of rotating magnetic field passing through it
  - Must wait long enough to get a measurement
    - When 1st de-powered, will get a negative spike from inductor dumping its charge (collapse of field)
  - Output voltage corresponds to speed of rotation
    - Need to convert to voltage range that ADC can handle
      - Voltage comparator gives square wave at target V
      - Need to factor out bus voltage



Closed loop detection continued

- Position encoder is used to detect one or many positions
  - Hall sensor or keyhole for one or two positions (N and S for example)
    - Must interpolate actual position based on time
    - Cannot tell direction normally
  - Absolute (optical) encoders are older method
    - Too few positions
  - Incremental encoders more common
    - Optical most common, but field methods used too
    - Two sensors (90° apart), gets position, speed, and direction
      - Counter must be fast enough – ideally in HW
      - More positions, more accuracy, especially under load





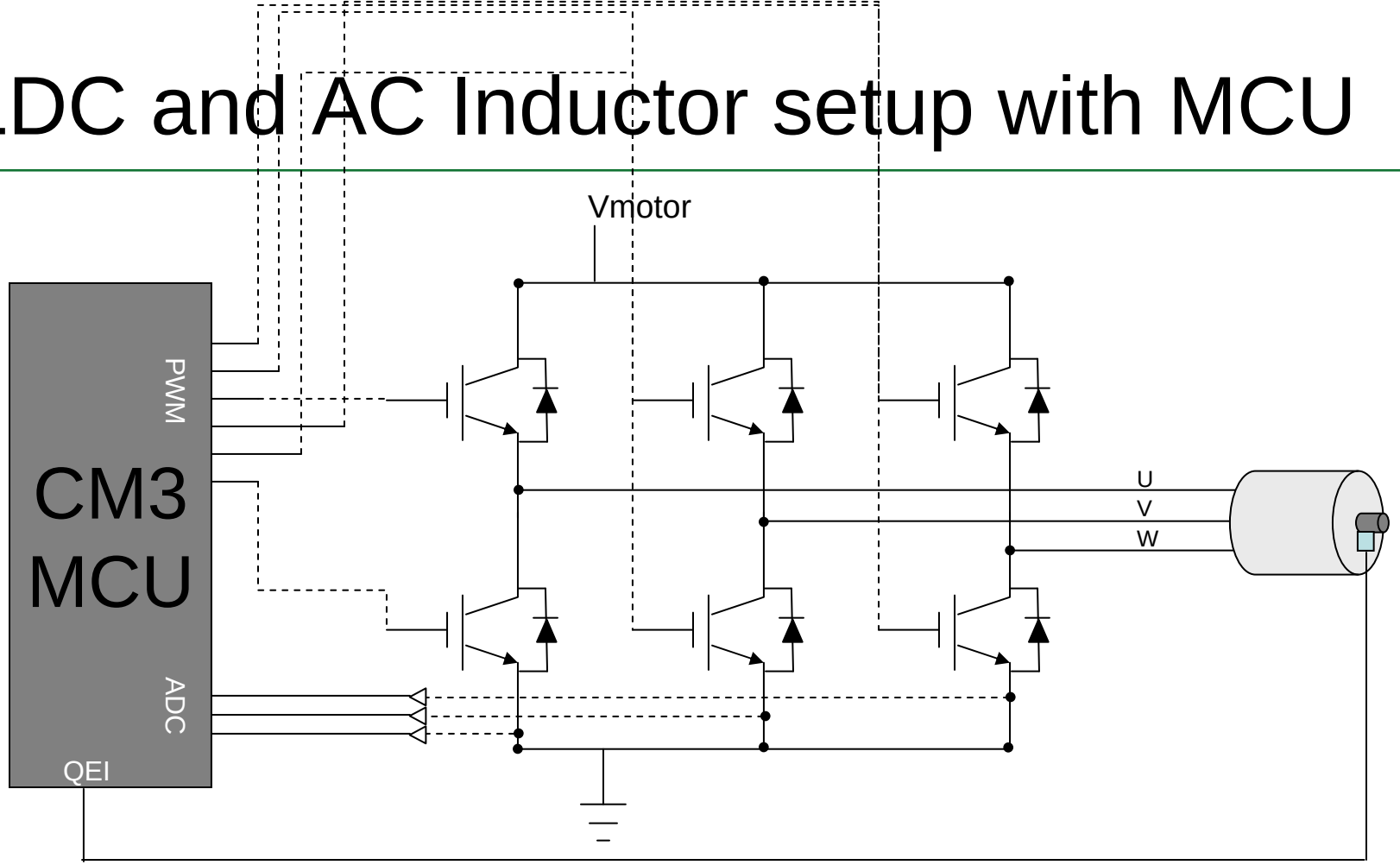
Closed loop detection continued

- Tachometer (or tach-o-generator) or resolver used
  - Normally just a DC motor or spinning winding
  - Yields speed of shaft and at least one position
    - Usually from asymmetrical rotor/stator position
  - May have sine wave output to get precise position
  - Used for very large motors
  - Used when motor too fast for other methods
  - Can detect direction from current direction

Contents

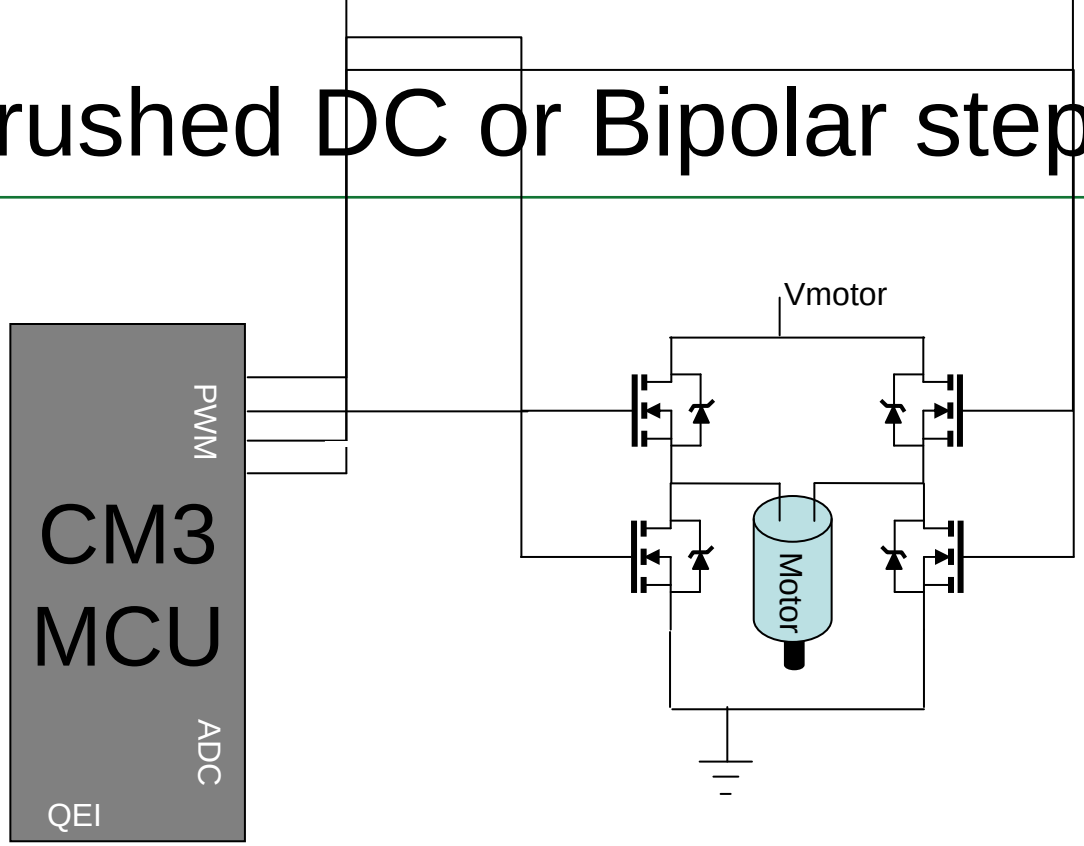
- Electronic Motor control: What and Why?
- What are the kinds of motors?
- What is PWM and Closed loop control?
- How do I wire these up?
- What are the algorithms?
- What are the advantages and disadvantages of each?
- Why Cortex-M3 makes this easier
- How you can mix motor control with your application

BLDC and AC Inductor setup with MCU



- Note: gate drivers, resistors, capacitors, etc not shown
  - 6 PWMs drive IGBTs through gate drivers
  - ADC senses current on each winding (low side)
  - QEI gets position (to 1/1000th most likely) and direction

Brushed DC or Bipolar stepper with MCU



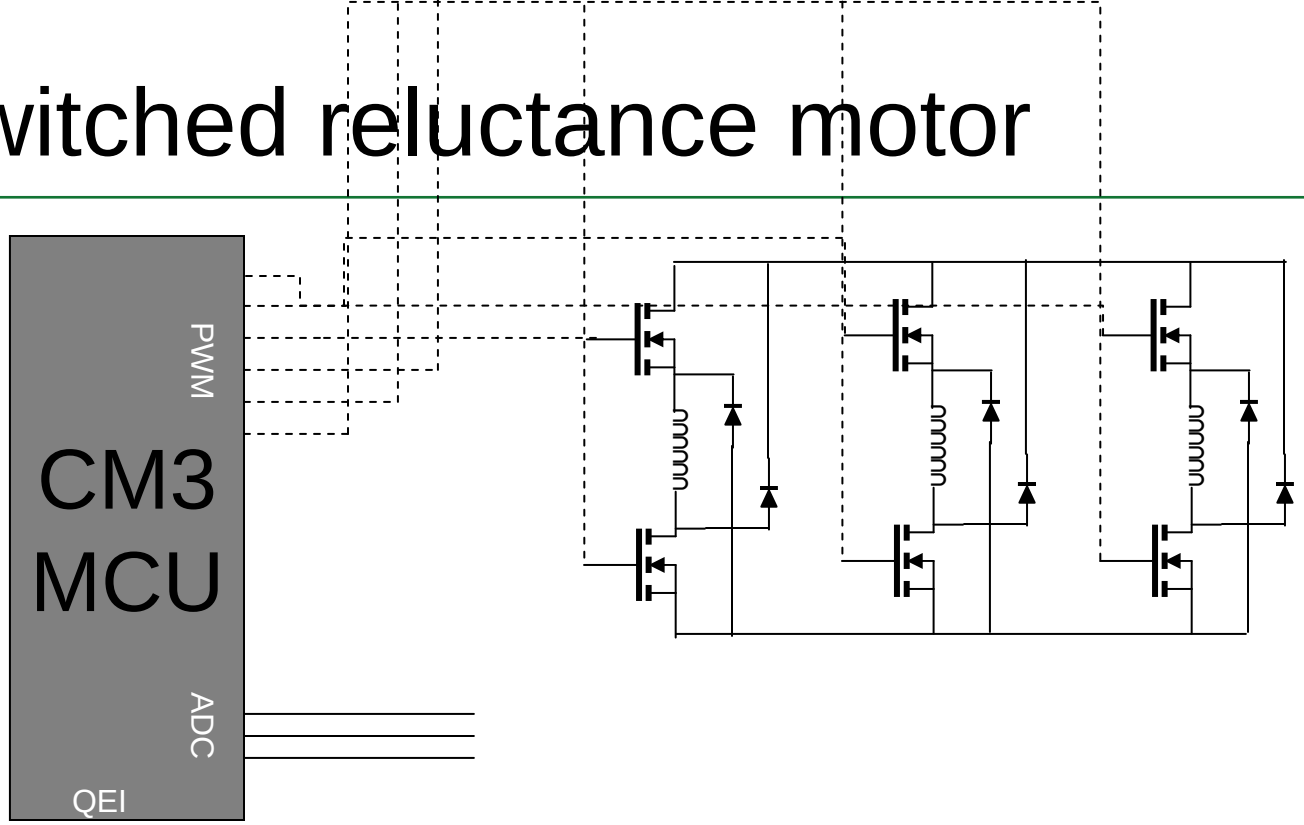
Bipolar stepper uses two such H bridges

- One for each winding
- Gate driver allows 3 PWM channels per (H, L, Enable)

Feedback options

- ADC to sense current
- QEI Encoder to provide rotor/shaft position

Switched reluctance motor



Notes:

- Coils shown for connections to motor winding pairs
- Note unusual diode positioning
- Advantage of this model is no short circuit possible
- Feedback of position is preferred (QEI)
    - Sensorless possible through current measurements to ADC

Contents

- Electronic Motor control: What and Why?
- What are the kinds of motors?
- What is PWM and Closed loop control?
- How do I wire these up?
- What are the algorithms?
- What are the advantages and disadvantages of each?
- Why Cortex-M3 makes this easier
- How you can mix motor control with your application

Algorithms - brushed

- PWM Chopper: open loop
  - Just picks apparent voltage and assumes works
  - Works fine if load is well defined and constant
  - Risk of load caused stall
- PWM Chopper and variations: closed loop
  - Simple PWM algorithm just adjusts periods on average
    - Works well when load changes slowly
  - Finer grained Chopping adjusts more often
    - Monitor and adjust based on requirements
- Acceleration and Deceleration are fairly well defined
  - Easy to adjust using error correction for closed loop

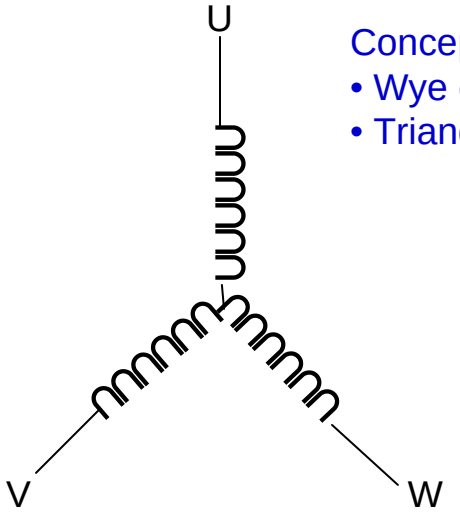
Algorithms - stepper

- Types:
  - 4 sequences for full step – two windings on: push/pulls to position
    - OK for low speed, when true *steps* needed, and hold – high torque
  - 4 sequences for wave – one on: pull to position
    - Smoother, lower torque, problems at high speed, low power draw
  - 8 sequences for ½ step – one on, then both on (between), then one
    - Quieter through most speeds, good torque, mix of wave and full
  - >8 sequences for micro-step: variations of power/on-times
    - Much quieter and smoother, but loses considerable torque
- Open Loop PWM – least CPU
  - Just drives sequences and assumes working
  - Rotor will tend to "stick" at tooth positions, so noisy
- Closed loop Chopper – most CPU, especially at speed
  - Constant adjustments
- Feedback based PWM (accelerate and then adjust) – decent CPU
  - Makes adjustment at start of step – excellent characteristics

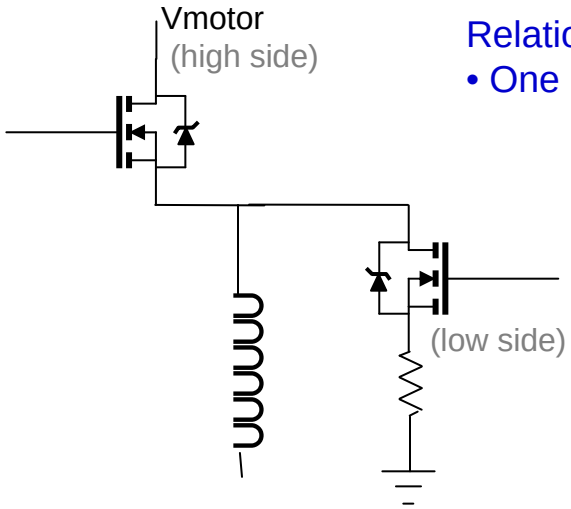
Algorithms – 3-phase (BLDC and AC)

- Simple sequenced chopping
  - OK for small motors/speeds, but tends to be noisy, especially EMI
- Trapezoid Space vector – sawtooth/triangle used to determine PWM periods
  - Approximates a rough sine wave (big steps) in one winding at a time
  - ± 30 ° of quadrature, so moderately good torque
  - OK at higher speeds, terrible at low speed (often clicks), high EMI
- Sinusoidal Space vector – generate sine wave in 2 windings
  - Gives decent sine wave approximation, good torque
  - Cleaner sequencing at low speed, fails at higher speeds
  - Needs highly accurate feedback to work at all
- Field Oriented Control (FOC) – determine stator current needed, given rotor's position
  - Excellent characteristics – adapts well since about adapting
  - Needs a lot of computation, but not so time sensitive
  - Works well at all speeds, allows adjustment of torque

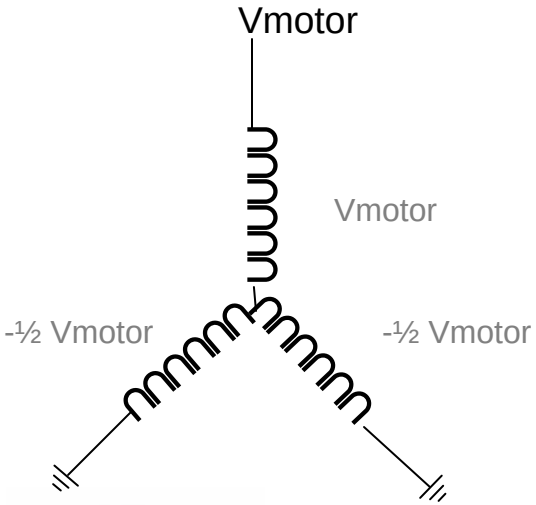
Actual 3-phase as used in PWM



Conceptual drawing
- Wye configuration
- Triangle acts the same



Relationship to Half bridge
- One leg shown

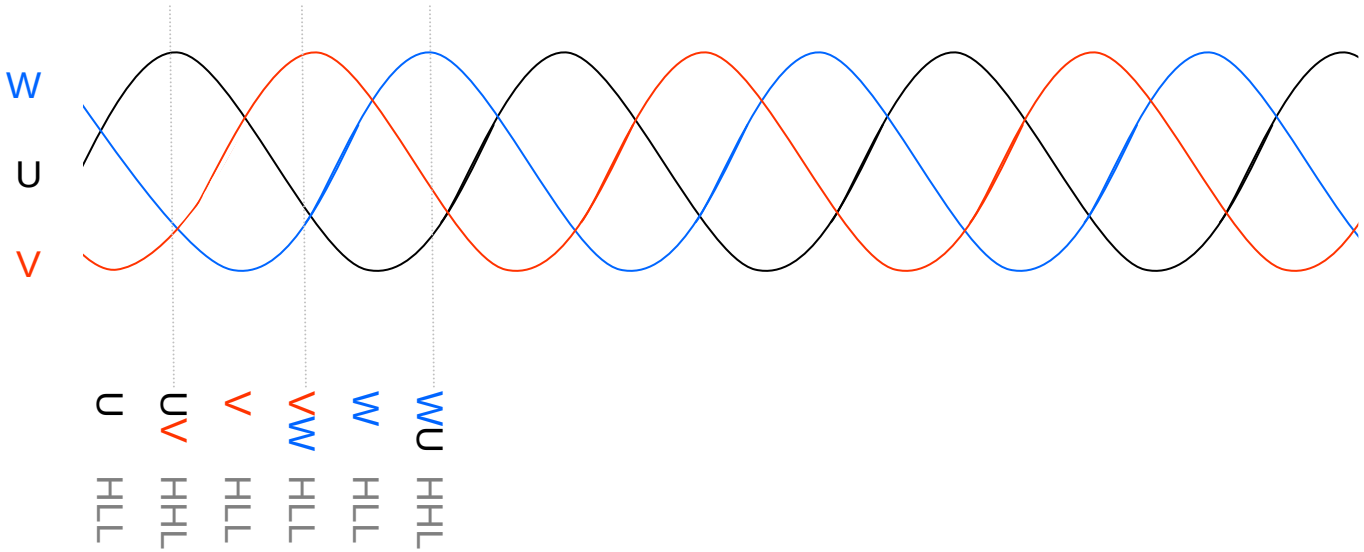


Basic: Full Voltage through one leg, ½ inverted to the other two
- High side transistor on for U
- Low side transistors on for V and W
- This forms one of 6 active space vectors
  - The other two are zero vectors (all high or all low)
  - There are 3 HLL: equivalent to top of peaks
  - There are 3 HHL: equivalent to mid points
    - Mid points are where next phase is rising

Complex: Full voltage to two, nothing to the other
- High side on for one, Low side for another, neither for the 3rd
  - Used to split 6 space vectors

Sequences of space vectors

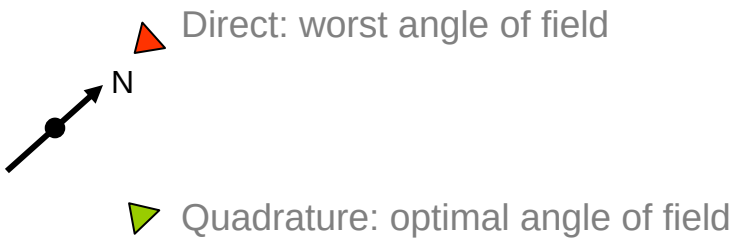
- The sequencing action minimizes switching (and its cost)



- Zero vectors may be used at inflections
  - HHH after HHL and/or LLL before HLL
- Modulation simulates sine wave in targeted winding
  - Over-modulation (net sum >V) is useful strategy for more torque
  - Over-modulation means Vmotor is enough greater than winding rating

Optimal winding model

- The optimal field relative to the rotor is 90° (quadrature)
- Angle of stator field is sum of each winding
  - Magnitude is generally 1 (each has a fraction of current from 0.0 to 1.0)



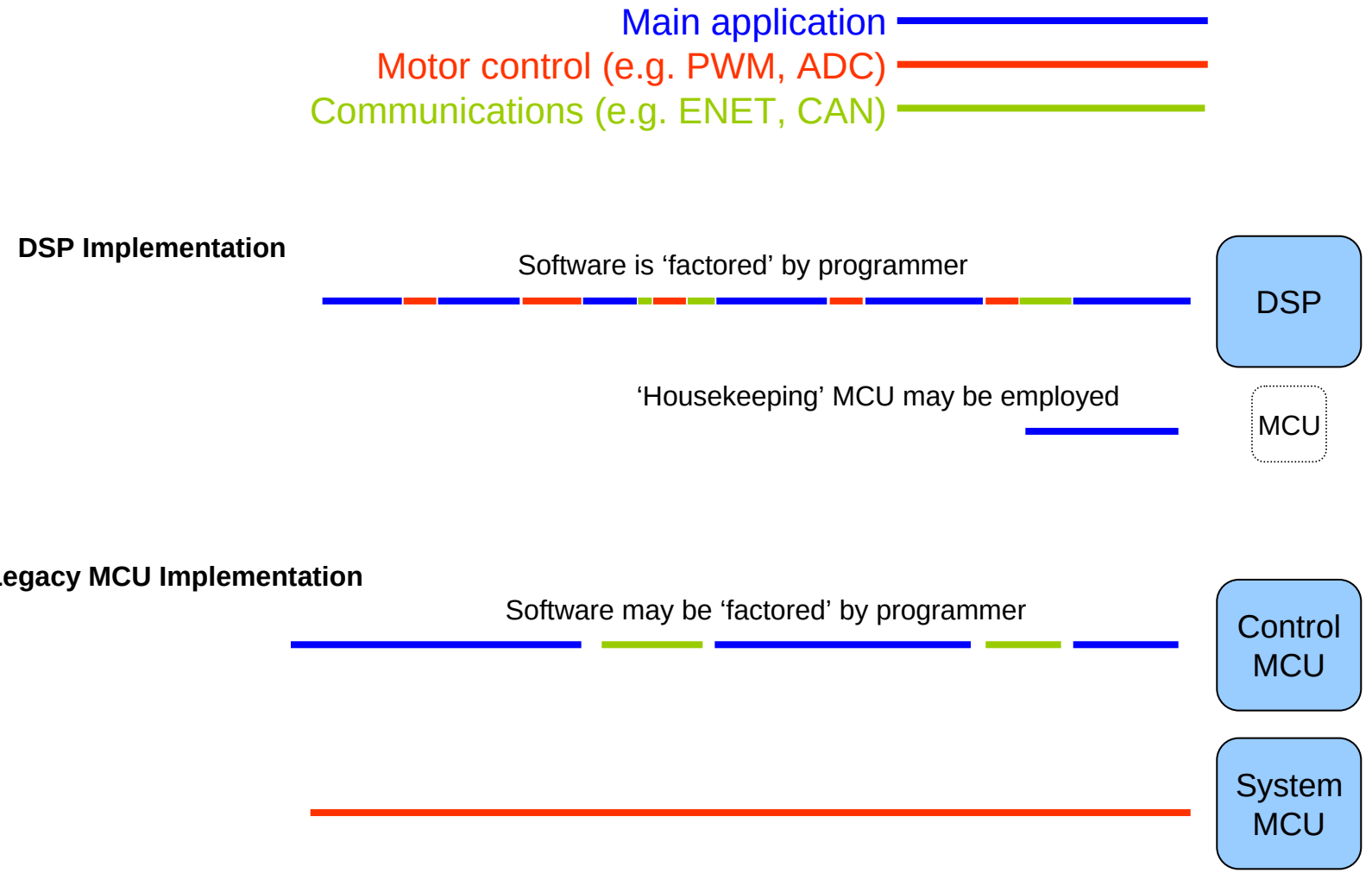
Field Oriented Control (FOC)

- FOC focuses on rotor's frame of reference
  - Uses Clarke transform to convert stator's 3-phase frame to 2-phase,
  - Uses Park transform to convert from stator's frame to rotor's
    - Collapses to single D and Q component
      - D is direct (in line with rotor, and so no impact) - Need to minimize
      - Q is quadrature (90° to rotor, so maximum impact) - Need to maximize
  - Defines corresponding Sin and Cos from these (Cos is just 90° OoP)
  - Uses iPark transform to convert from rotor's frame to stator's
    - Uses iClarke to get to stator's 3 120° frame components
    - Normally 0 input to direct space vector (want to minimize)
    - Maximal input to quadrature space vector – unless need to back off
      - PWM generated to the 2 windings in q's space vector
      - PWM ratio between two keeps close to quadrature angle

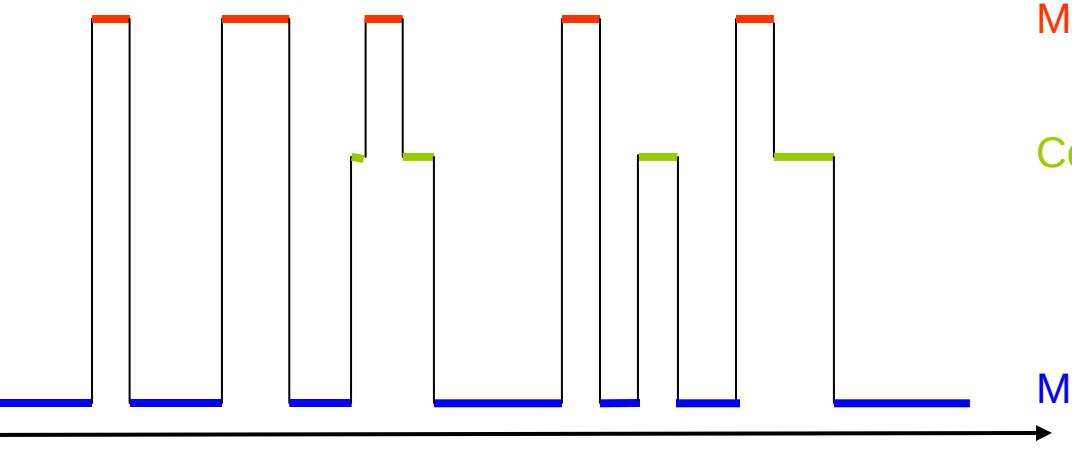
Contents

- Electronic Motor control: What and Why?
- What are the kinds of motors?
- What is PWM and Closed loop control?
- How do I wire these up?
- What are the algorithms?
- What are the advantages and disadvantages of each?
- Why Cortex-M3 makes this easier
- How you can mix motor control with your application

How it works on Traditional DSP/MCU



How it works on Cortex-M3



- Main application runs as foreground (base level)
  - Easy to write since no “factoring” – just normal application or RTOS based
  - Can use PLC style state-machine poll loop safely: ISRs keep data available
- ISRs for Motor control are highest priority(ies)
  - PWM, ADCs, Timer(s), Fault (may be highest), Temp sensor, etc
- ISRs for communications below that
  - Ethernet, CAN, and/or serial
- May use other priorities as needed
  - Very fast interrupt response time, true nested interrupts, priority masking, easy ISR setup all contribute to making an easy solution
  - Application uses priority masking vs. interrupt-disable if needs critical region

Cost of algorithms on Cortex-M3

- Cortex-M3 well suited to this - very fast interrupt response
  - Time from feedback to result is crucial – rotor still moving!
  - Use of fast prioritizable ISRs allows an application to run also
  - Fast math, including MUL (1 cycle) and DIV (3-12 cycles)
  - Able to do fixed point well: has USAT and SSAT when wanted
- MCU design is a big factor though
  - Need good multi-channel PWM device with low overhead to reprogram often
    - If 6 wait states to write registers, a lot of time lag and loss
    - Need to consider when update synchronized, if dead-band handled, faults
  - Fast ADC with at least 3 channels: sequencing important
    - Current comparison for error adjust crucial to optimizing
    - Time to lock and time to get result will affect algorithm response
      - Risk of more error adjust than acceptable
      - PWM triggering ADC ensures reliable timing
  - QEI/counter in HW important
    - Dropped pulses cause serious problems at high speed
    - Less load on CPU when using high res encoder
    - Get current position when needed with high accuracy (timer gives delta)

Why that does not work for other processors

- DSPs, ARM7/ARM9 and others too slow on interrupts
  - High overhead uses too much processor time to service high rate motor control interrupts
    - True 12 or 13 cycles to 1st real user instruction vs. 20, 40, or more
      - May be able to get in faster if only one ISR, but nesting is important (comms, faults, etc)
    - Much of processor headroom lost in overhead
    - With non-deterministic behavior, need to allow for worst case
  - Most common approach is then to make motor control the foreground task
    - Now must poll for communications and application functions
      - Or large int disable (critical section) to protect motor control
    - Likely need to “factor” application so motor control does not miss time windows

Example Stepper cost numbers on Stellaris

- 50 MHz part (zero wait states on Flash and RAM)
  - CPU Load percent using between 20 and 2K steps/second, then 3K, then 5K s/sec
  - Using Luminary RDK board and SureStep motor
  - PWM at 20KHz
  - Motor control code size between 4.5K and 7.5K including init data
    - Smaller is one algorithm, larger has all algos with runtime selection
  - Motor control data size around 850 bytes (carries configuration data too)

Algo	Full-step	Wave	Half-step	Micro-step
Open PWM	0 to 2%, 3%, 3%	0 to 4%, 3%, 4%	0 to 3%, 4%, 6%	0 to 12%, 18%, 31%
Chopper	8 to 59%, -, -	4 to 60%, 61%, 62%	8 to 81%, 81%, 82%	6 to 66%, 83%, -
Feedback PWM	12% to 13%, 13%, 15%	10 to 11%, 12%, 14%	11 to 13%, 14%, 17%	11 to 24%, 30%, 42%

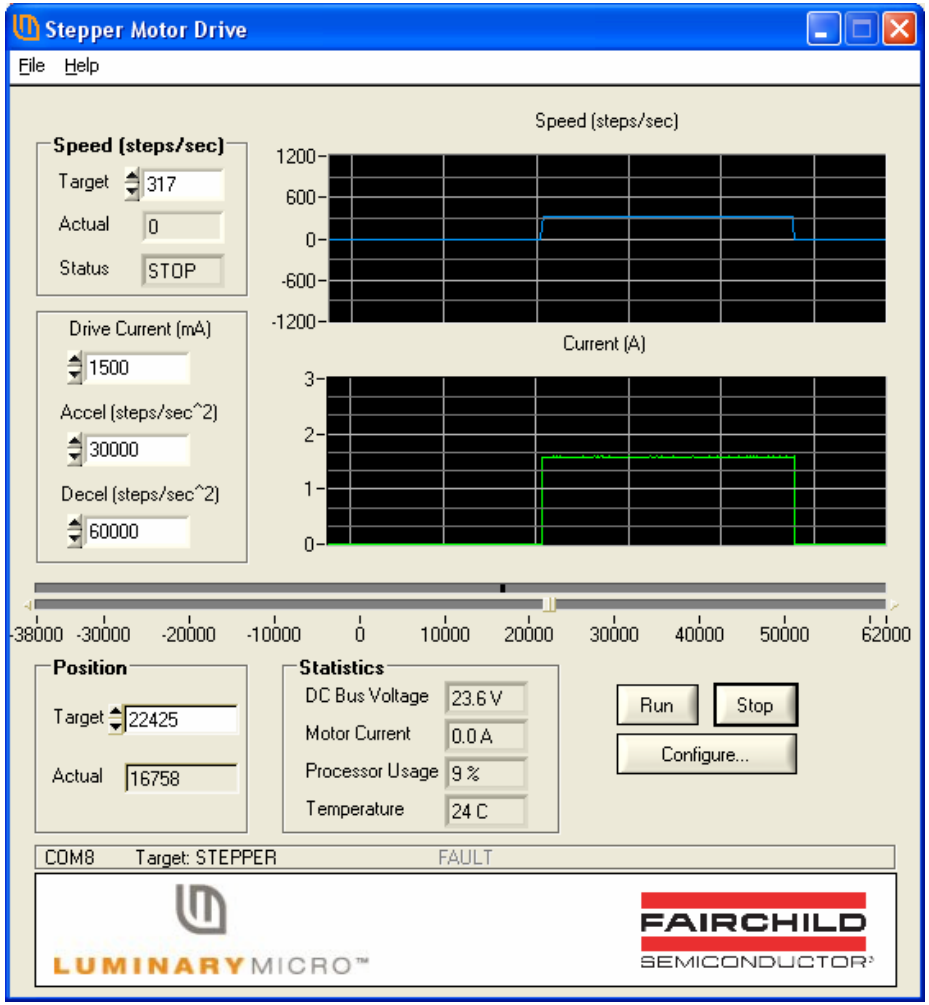
Example AC Inductor cost numbers on Stellaris

- 50 MHz part (zero wait states on Flash and RAM)
  - Using Luminary RDK board and Selni appliance motor (up to 20Krpm)
  - PWM at 20KHz
  - CPU Load percent for sine-wave and space vector algorithms
    - Which algorithm and closed vs. open has only small effect on CPU load
  - Motor control code size is 10.5K or less (that includes mixed algorithms)
  - Motor control data size is about 750 bytes or less (that includes configuration)

PWM clocks per update	8KHz PWM	12.5KHz PWM	16KHz PWM	20KHz PWM
1 per	19%	30%	38%	47%
2 per	14%	21%	26%	33%
3 per	12%	18%	22%	28%
4 per	11%	16%	20%	25%
5 per	10%	15%	19%	24%

Tuning

- Adjusting algorithm to particular motor and application is crucial
  - A GUI that helps do this is shown
  - It monitors many factors and provides a trace of use
- Verifying that a motor control algorithm can meet your needs is only half of it
  - Need to verify it does not use too much processor time
    - Application has to run too
  - Or, does not work well with motor and load
    - May degrade quality of control



Tune and then install into Application

- Can start with reference board or own board
  - Get your own motor working well unloaded
  - Then test loaded (in use if safe, else rigged load)
    - Verify required uses (e.g. steady, acceleration, braking, hold, etc)
  - Then, install application
    - Verify application meets its timing requirements
    - SWV can be used to see how things are playing together
  - If not on own board so far, move to it
- Reference schematic makes it much easier to design
  - Substituting parts is fine, but understand implications
  - AC needs careful isolation
    - MCU normally lives on AC side, so need isolation for comms
    - Magnetics for ENET may be enough, opto-isolators for others

Conclusion

- Understand type of motor needed for application
  - Torque, starting torque, position accuracy, speed accuracy, etc
- Understand how wired up
  - Based on motor and closed vs. open loop
- Understand algorithms and tradeoffs
  - Torque, smoothness, noise, processor load
  - Tuning based on actual application and motor
- Designing application in light of easier split of work
  - Cortex-M3 allows motor control to be done entirely with ISRs
  - May reduce from two or three MCU/DSP to one
- <plug for="LuminaryMicro">
  - Demonstration of ACI, Stepper, BLDC in Luminary booth
    - Please feel free to visit or ask questions
  - RDK boards with professional motors: $199 for Stepper, $379 for ACI
    - Fully working software available for free from Luminary Micro
    - Reference schematics and Gerbers also available for free
  - </plug>