Použití mikroprocesorů pro řízení pohonů s BLDC motory

Ing. Jaroslav Lepka
Ing. Pavel Grasblum, Ph.D.

10. – 11. listopadu 2011

Tato prezentace je spolufinancována Evropským sociálním fondem a státním rozpočtem České republiky.

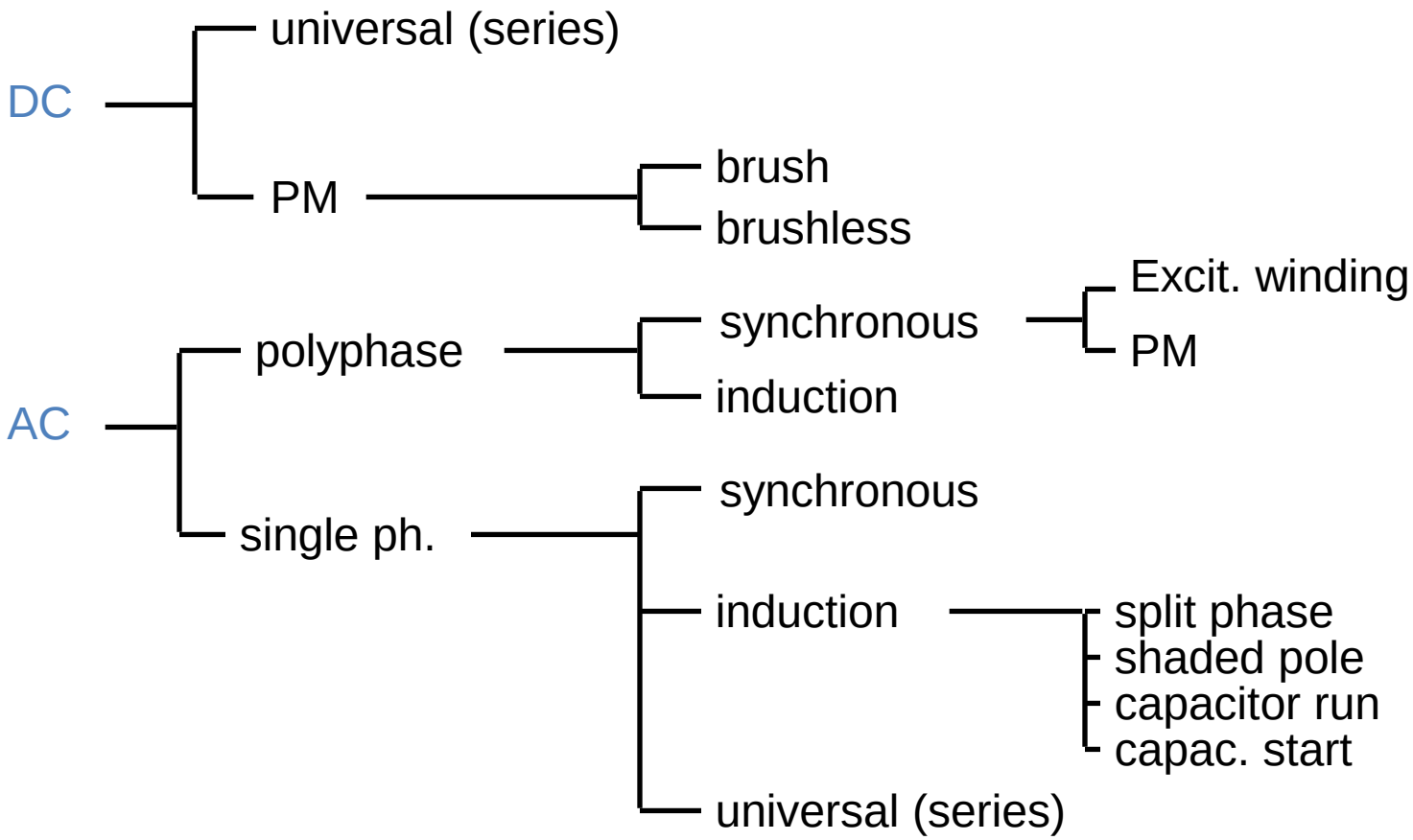
Agenda

- Basic Terms
- BLDC Motor Theory
- Sensorless Technique of BLDC Motors
- Microcontroller MC56F8006
- Freescale Software Library

Agenda

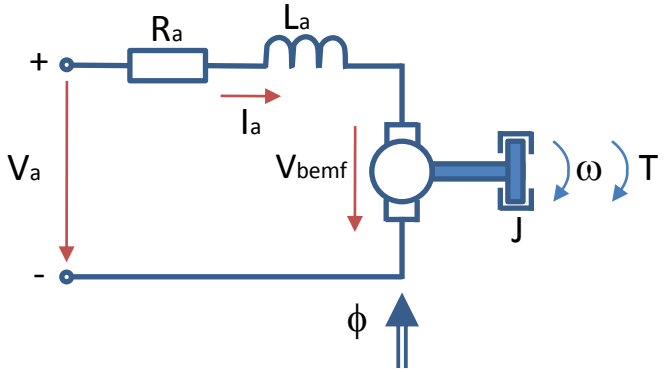
- Basic Terms
  - Separately excited DC motor
  - Operational characteristic of the drive
  - Unipolar versus Bipolar switching
  - Independent versus Complementary switching
  - Edge versus Centre aligned PWM
- BLDC Motor Theory
- Sensorless Technique of BLDC Motors
- Microcontroller MC56F8006
- Freescale Software Library

Electric Motor Type Classification



Stepper & Switched Reluctance

Separately Excited DC Motor



Equation describing dynamics of electric circuit

$$V_a = R_a \cdot I_a + L_a \cdot \frac{dI_a}{dt} + V_{bemf}$$

State equations – motor dynamics

$$\frac{dI_a}{dt} = \frac{1}{L_a} \cdot \left(V_a - R_a \cdot I_a - V_{bemf}\right)$$

$$\frac{d\omega}{dt} = \frac{1}{J} \cdot \left(T - T_L\right)$$

Steady state operation

$$V_a = R_a \cdot I_a + V_{bemf}$$

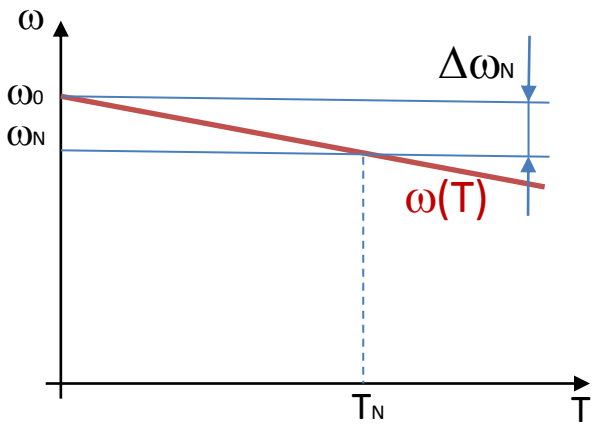
$$\omega = \frac{V_a}{c \cdot \phi} - \frac{R_a}{\left(c \cdot \phi\right)^2} \cdot T$$

$$\omega_{0N} = \frac{V_{aN}}{c \cdot \phi}$$

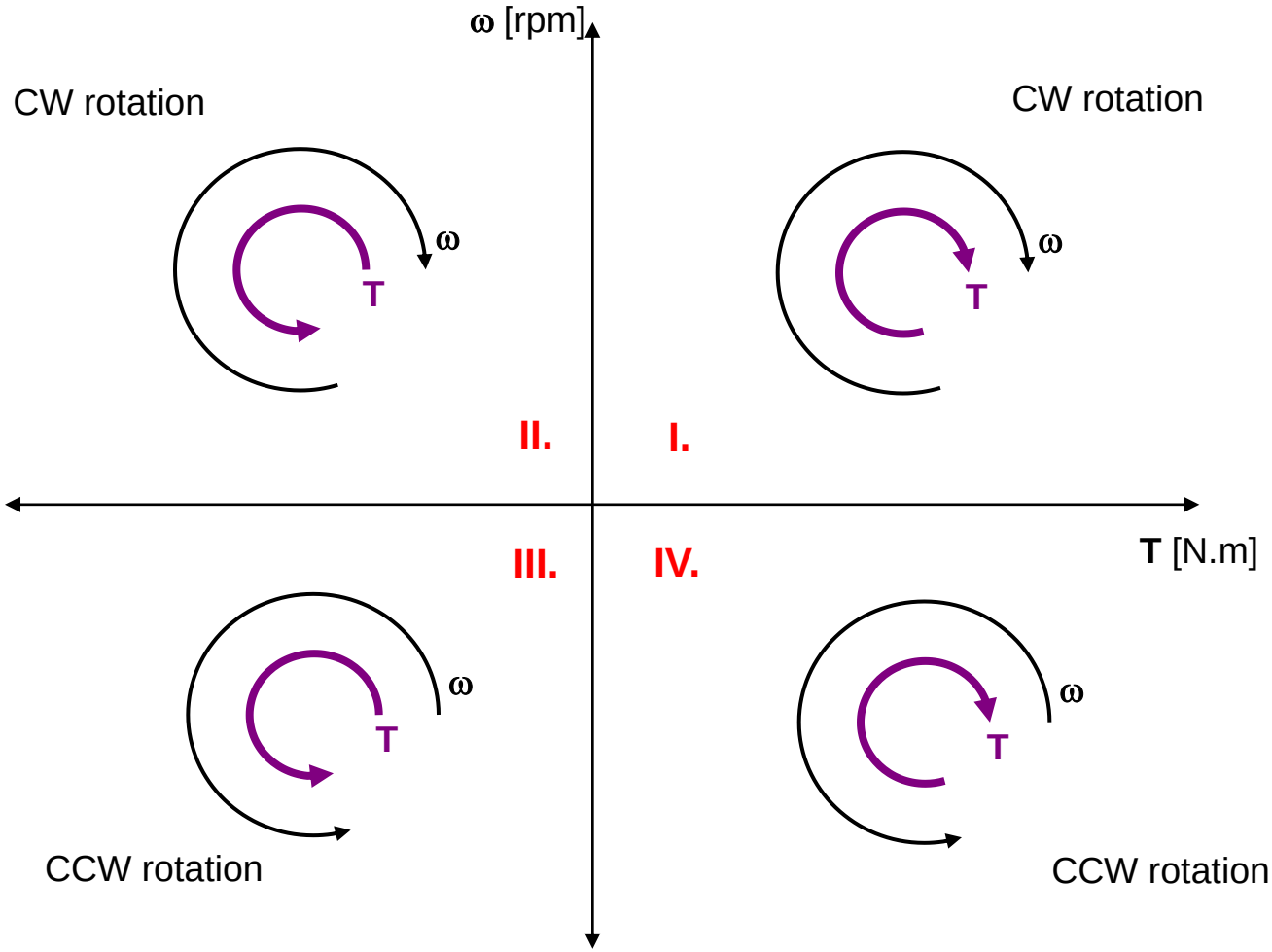
$$\Delta\omega_N = \frac{R_a}{\left(c \cdot \phi\right)^2} \cdot T$$

$$V_{bemf} = c \cdot \phi \cdot \omega$$

$$T = c \cdot \phi \cdot I_a$$

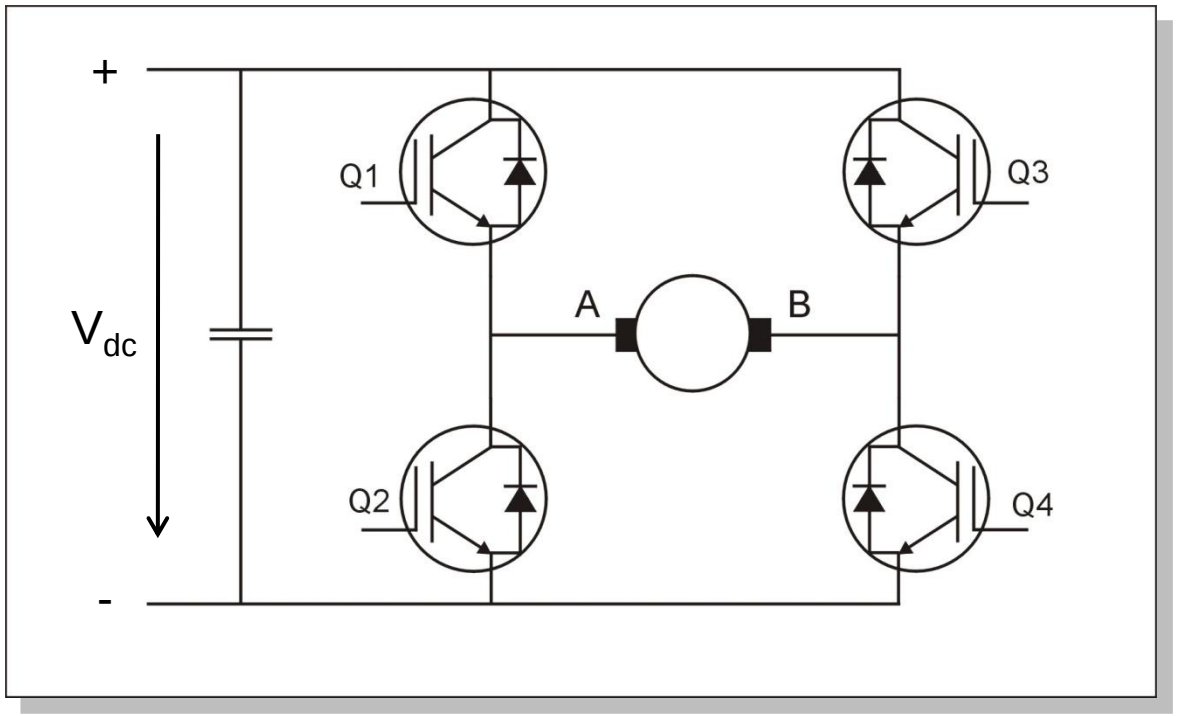


Operational Characteristic of the Drive



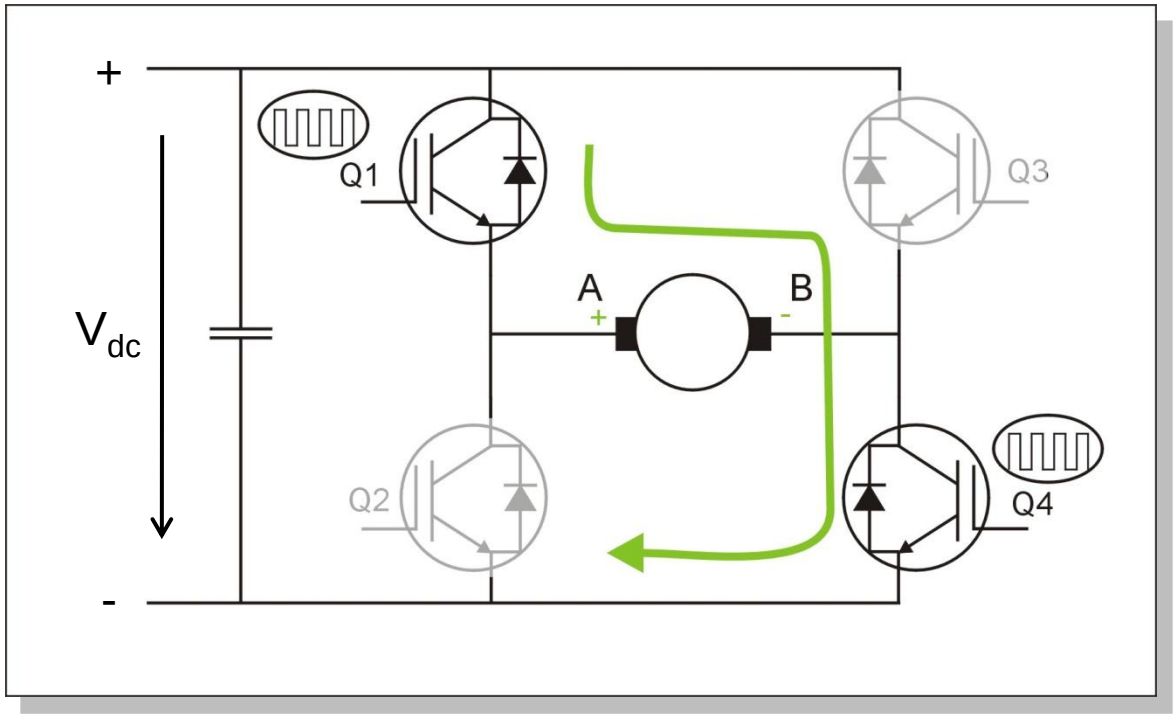
Unipolar versus Bipolar Switching

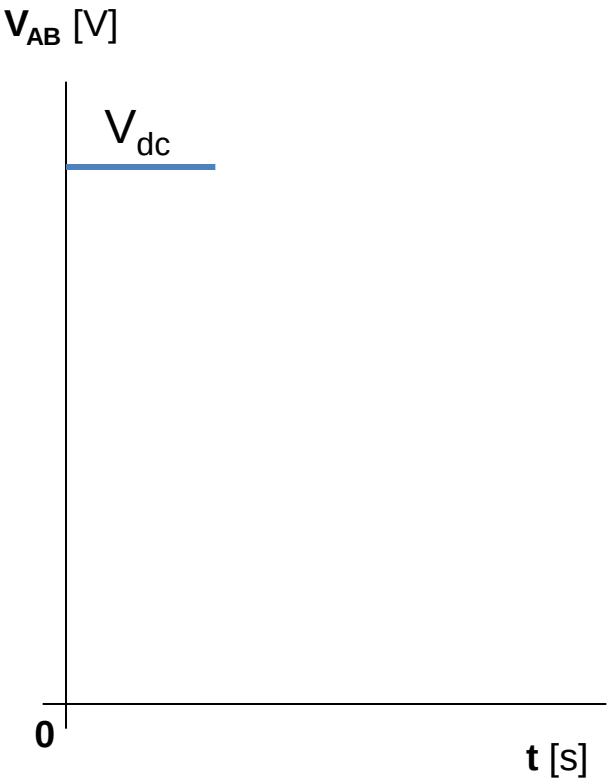
- The terms “unipolar” and “bipolar” are related to how a motor can see voltage on its terminals.



Unipolar versus Bipolar Switching

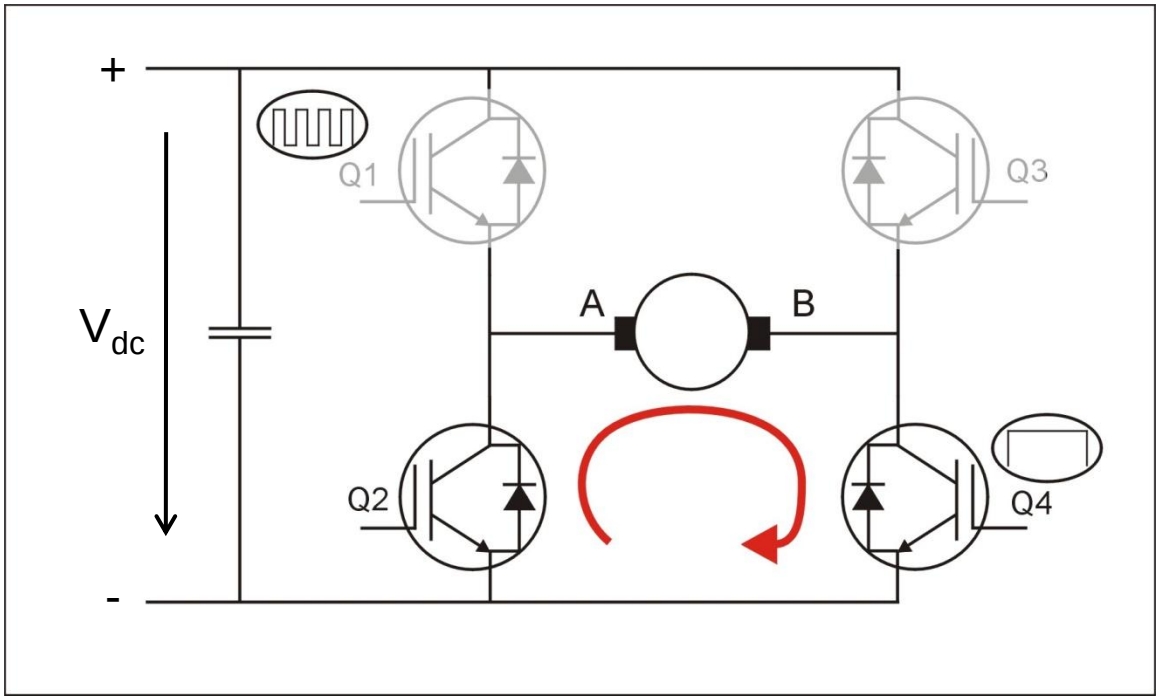
- Unipolar switching

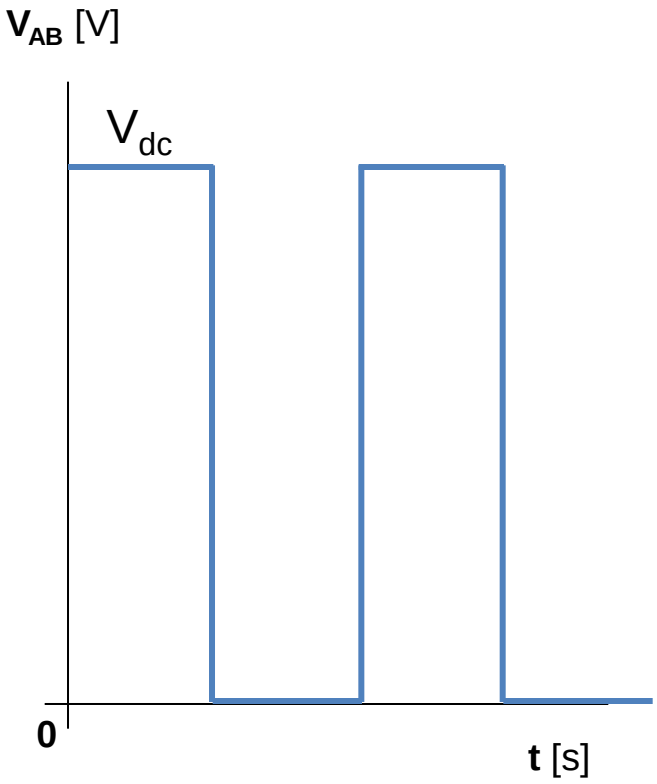




Unipolar versus Bipolar Switching

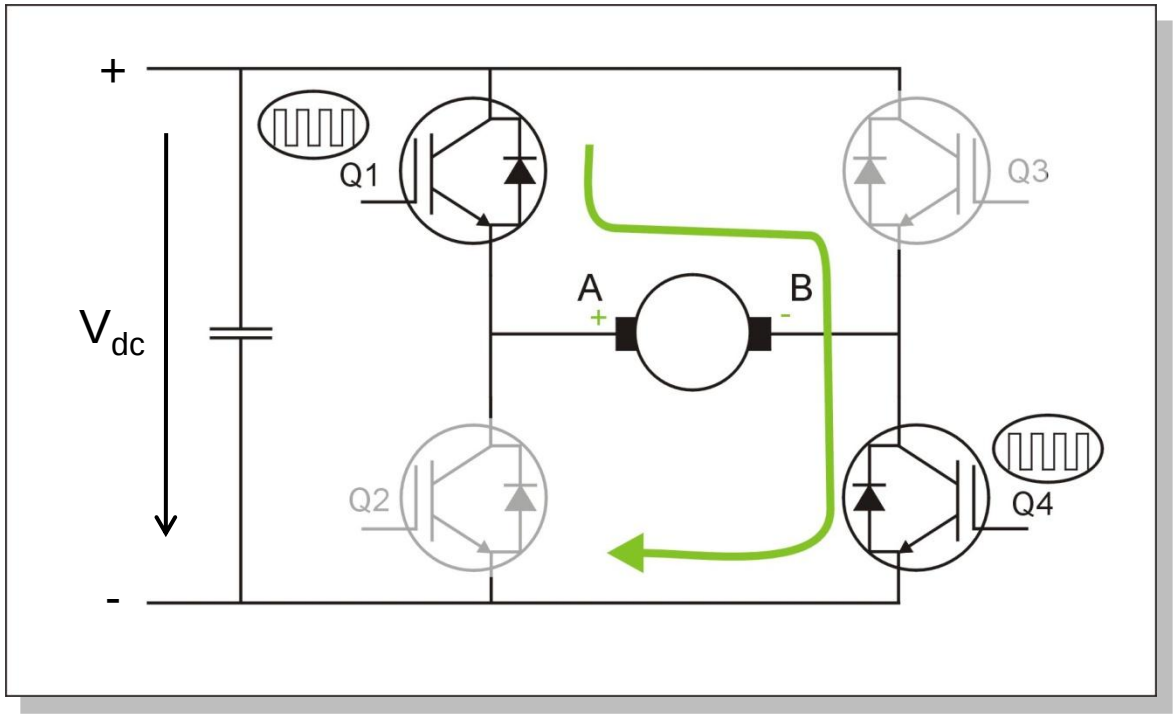
- Unipolar switching

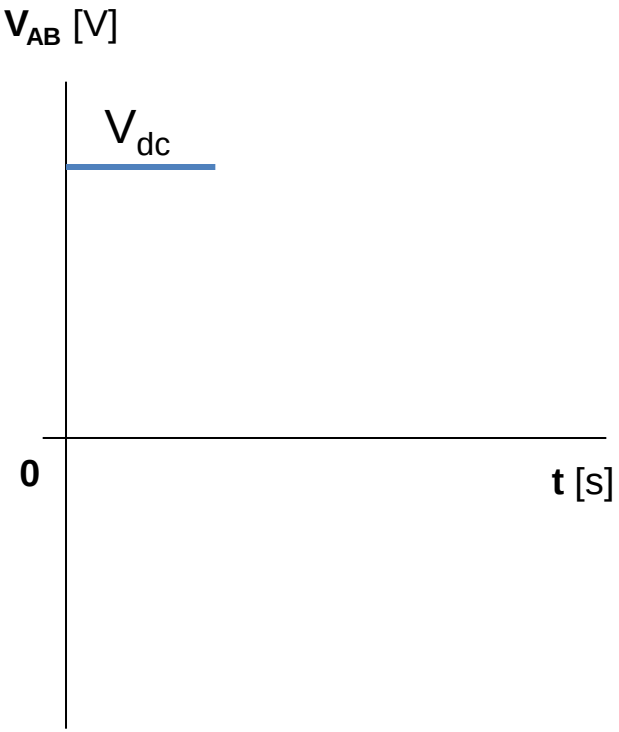




Unipolar versus Bipolar Switching

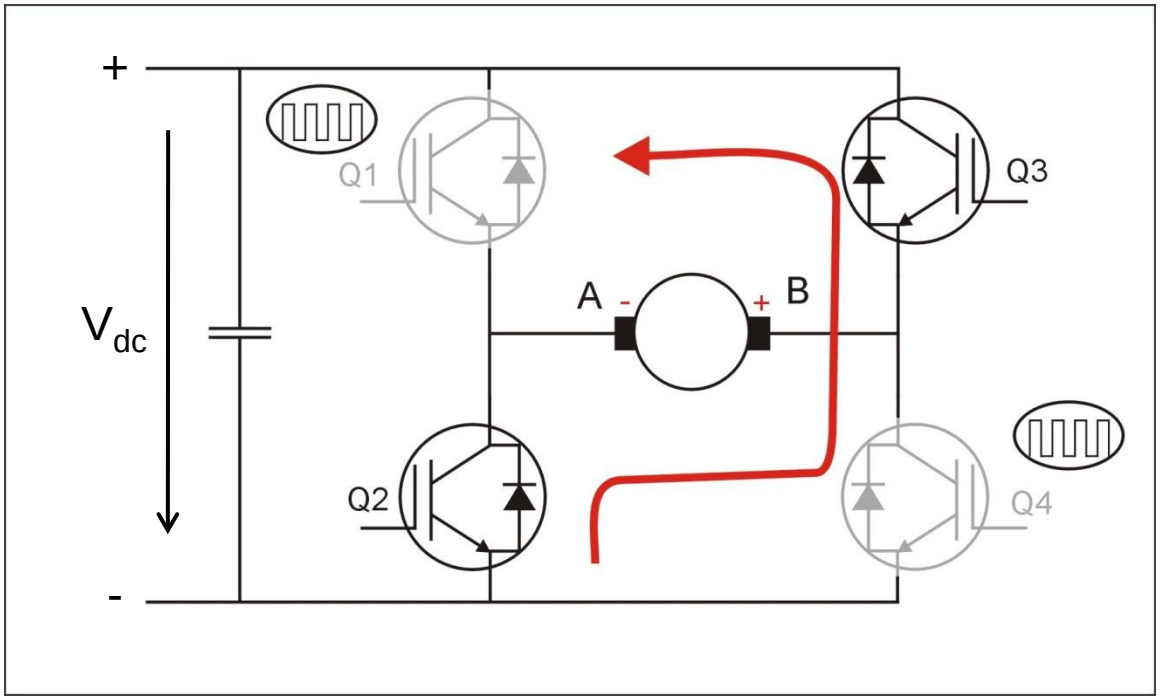
- Bipolar switching

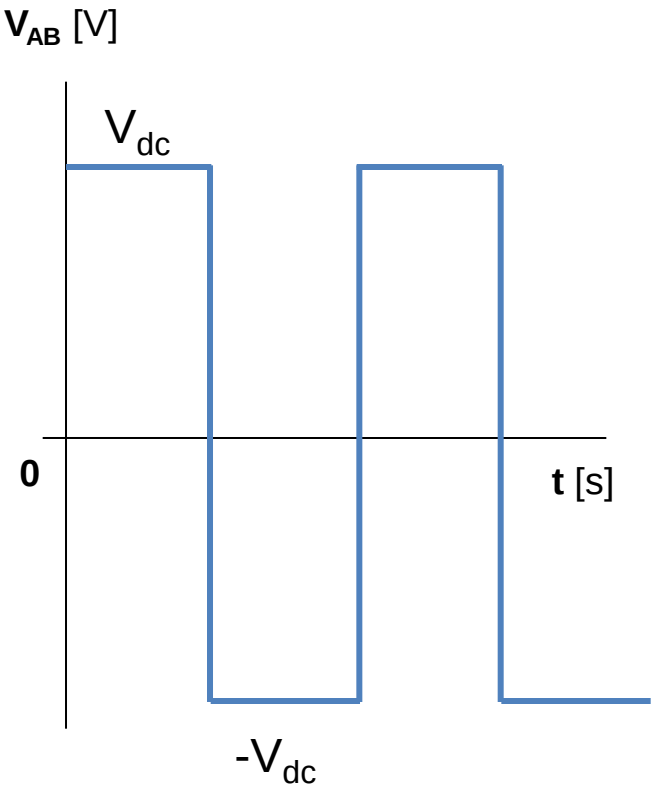




Unipolar versus Bipolar Switching

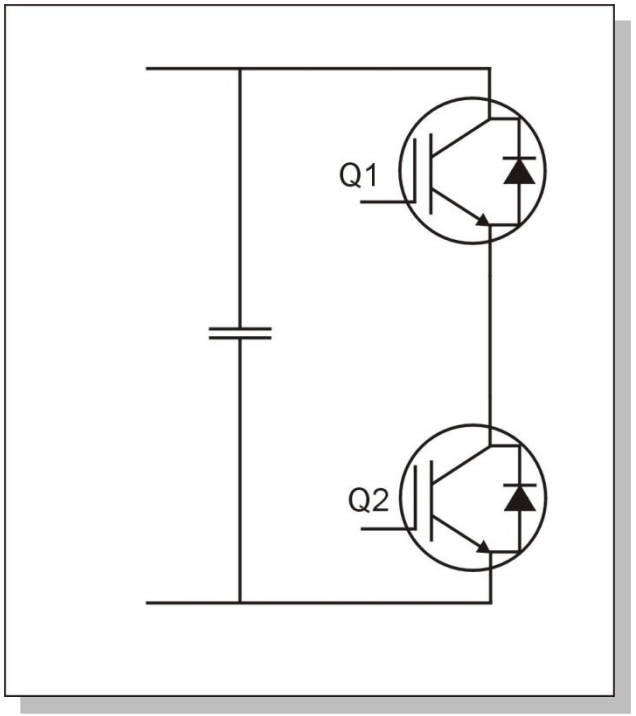
- Bipolar switching





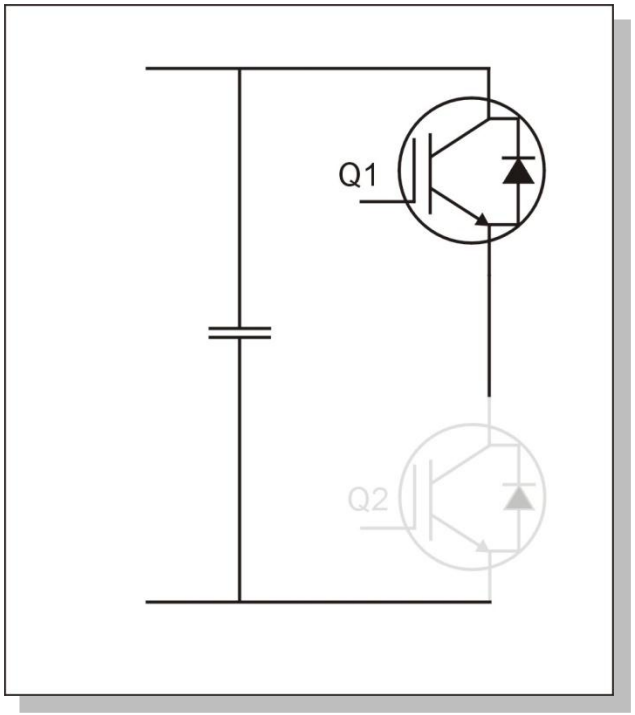
Independent versus Complementary Switching

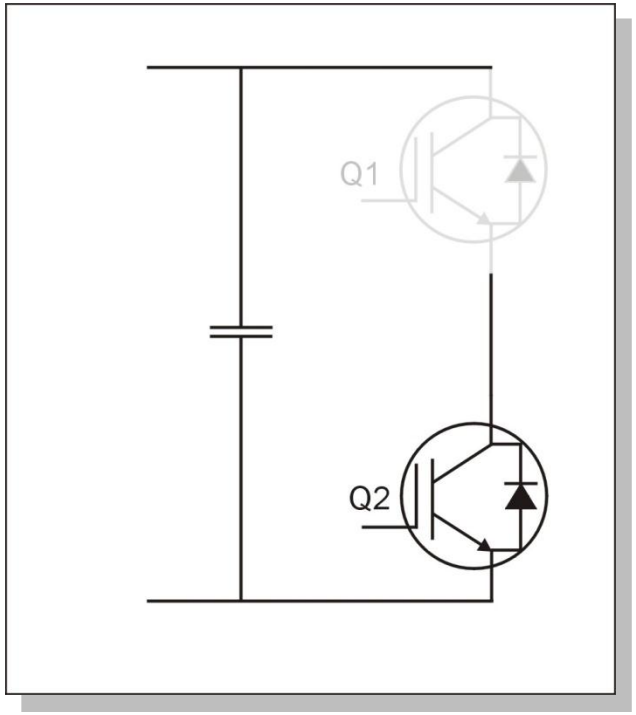
- The terms “independent” and “complementary” are related to how the transistors are switched in one phase.



Independent versus Complementary Switching

- Independent switching

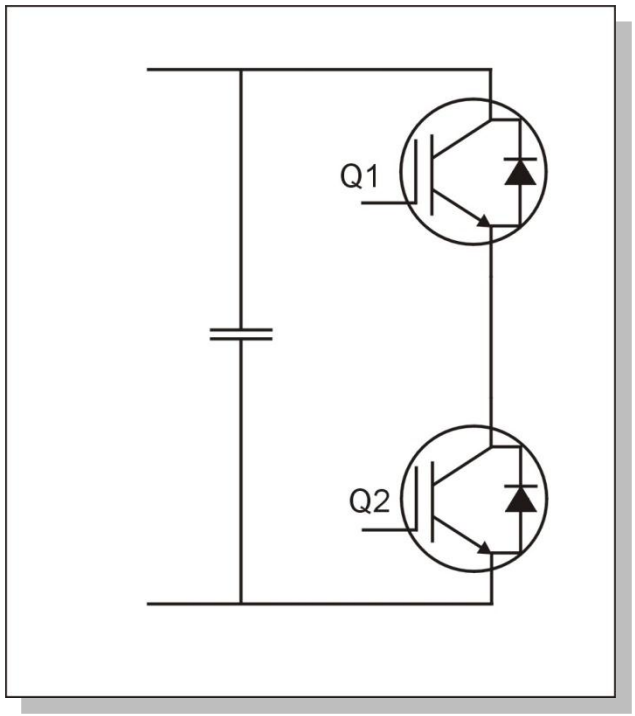




The top or bottom transistor is switched during whole period.

Independent versus Complementary Switching

- Complementary switching



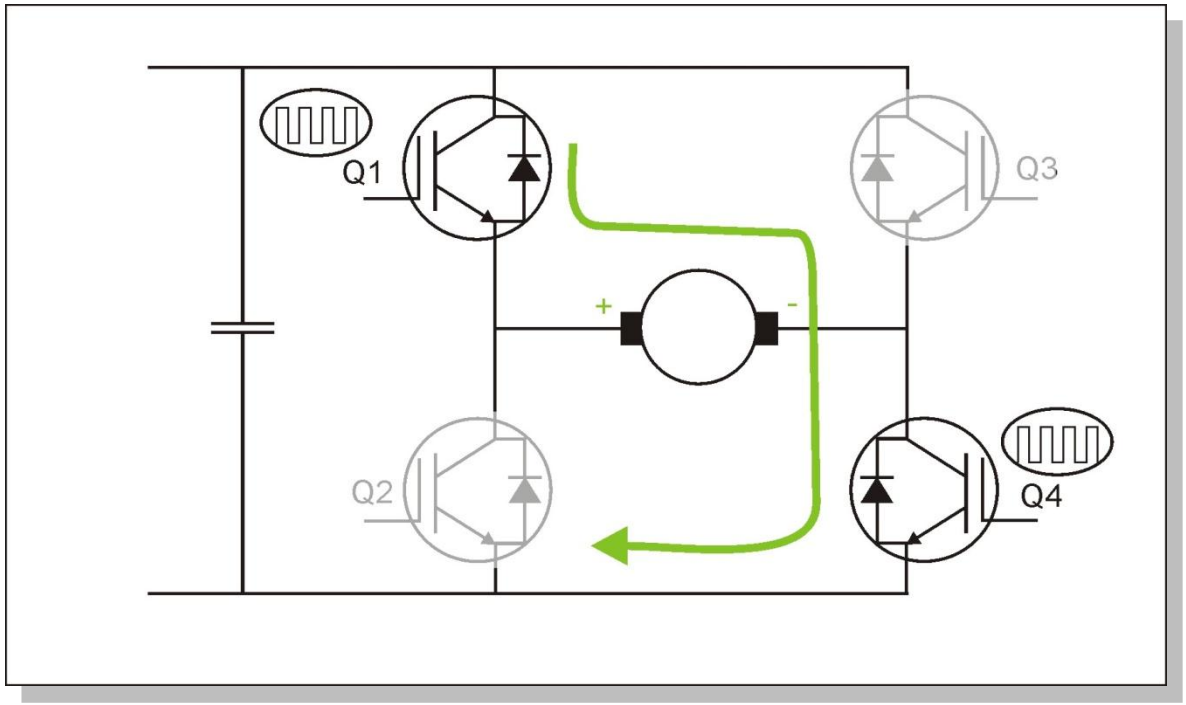
Both transistor are switched in complementary manner during whole period

Pulse Width Modulation Technique

- DC Motors – Single Quadrant Operation
  - Single Switch and Freewheeling Diode
- DC and BLDC Motors – 2 & 4 Quadrant Operation
  - H-Bridge & 3-Phase Bridge
    - Independent Bipolar Switching
    - Independent Unipolar Switching
    - Complementary Bipolar Switching
    - Complementary Unipolar Switching
    - Independent/Complementary Unipolar Switching

Pulse Width Modulation Technique

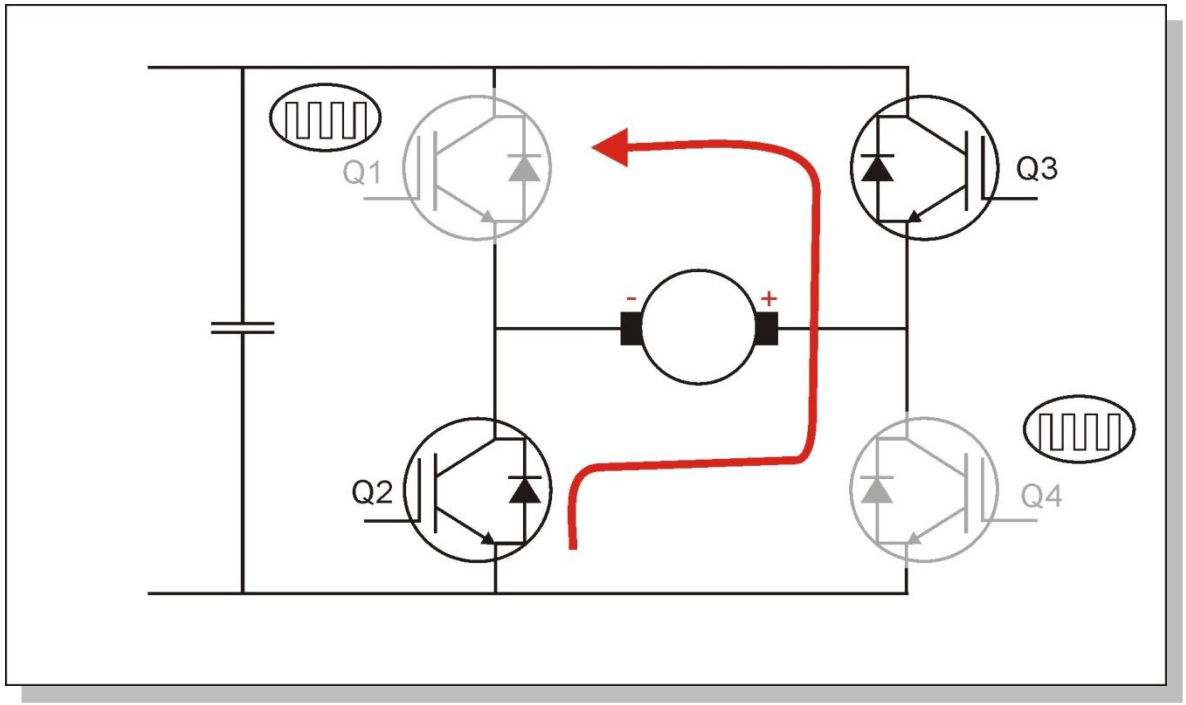
- DC and BLDC Motors – 2 Quadrant Operation
  - Independent Bipolar Switching – Acting Transistors



Q1=Q4=PWM

Pulse Width Modulation Technique

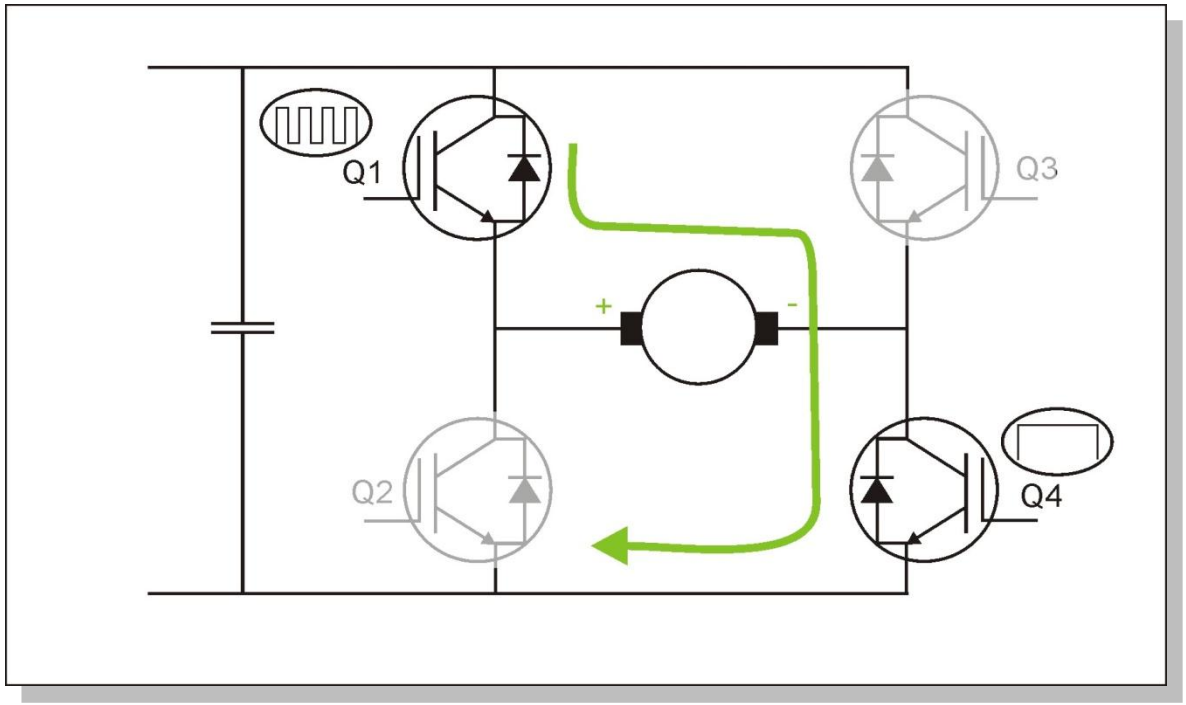
- DC and BLDC Motors – 2 Quadrant Operation
  - Independent Bipolar Switching – Freewheeling Diodes



Q1=Q4=PWM

Pulse Width Modulation Technique

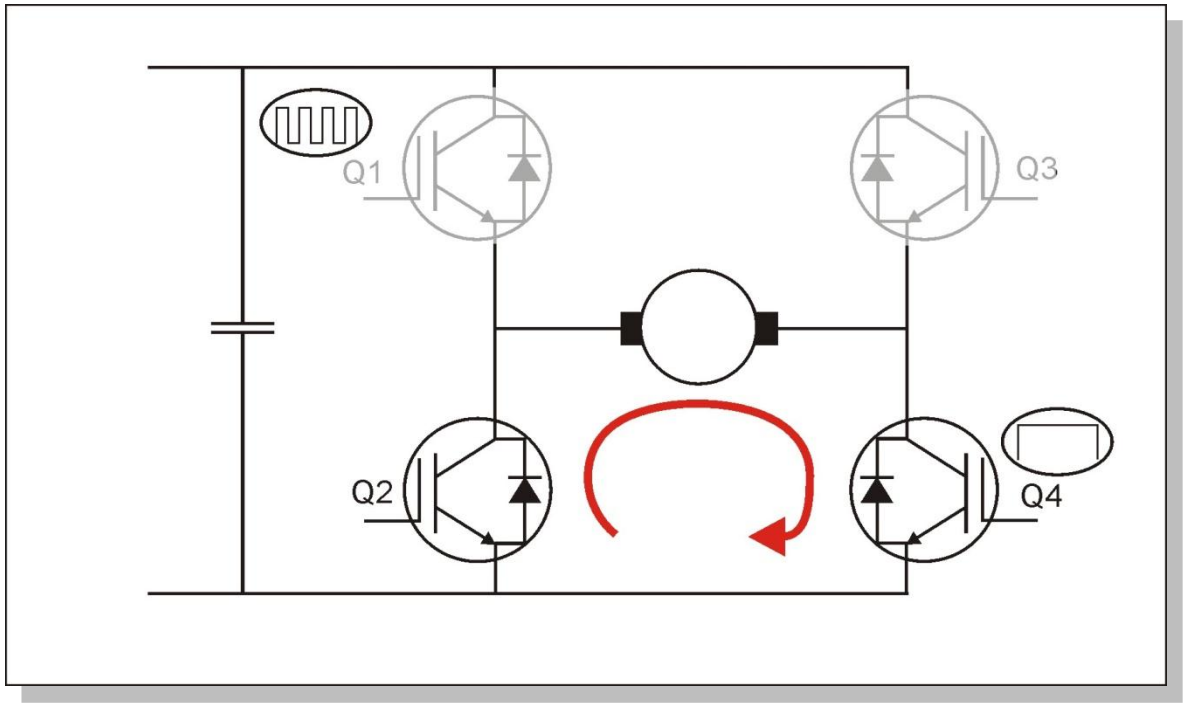
- DC and BLDC Motors – 2 Quadrant Operation
  - Independent Unipolar Switching – Acting Transistors



Q1=PWM; Q4=ON

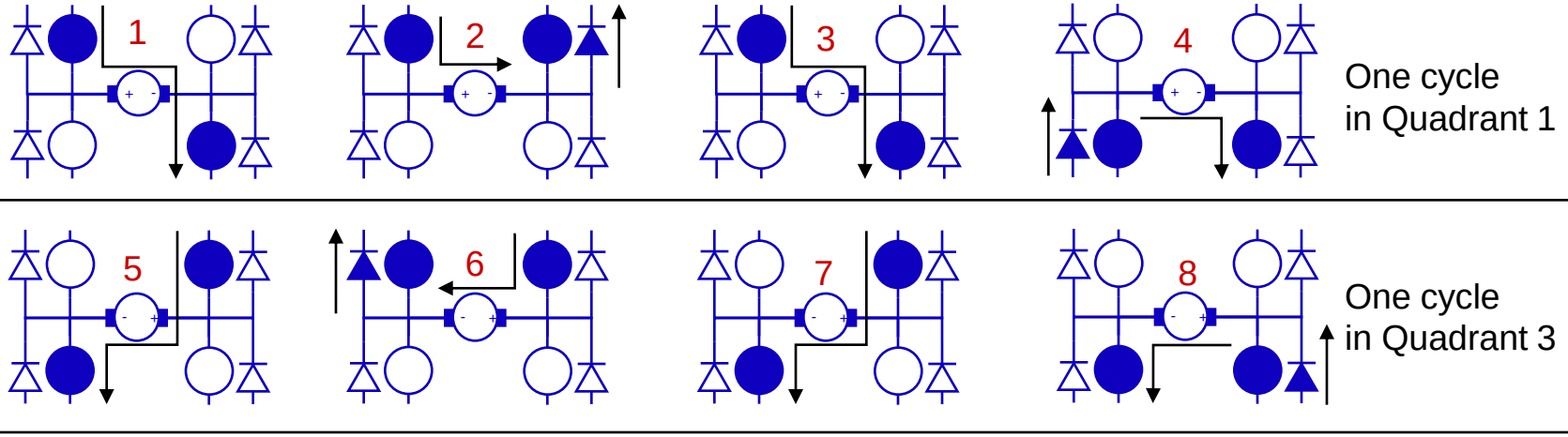
Pulse Width Modulation Technique

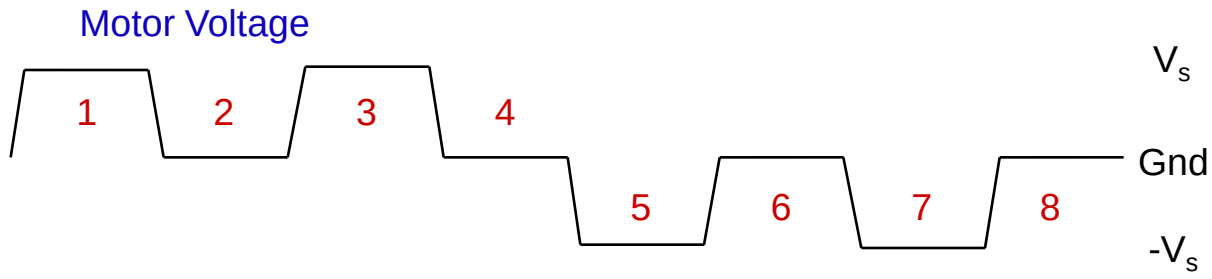
- DC and BLDC Motors – 2 Quadrant Operation
  - Independent Unipolar Switching – Freewheeling Diodes



Q1=PWM; Q4=ON

Unipolar Switching Modes





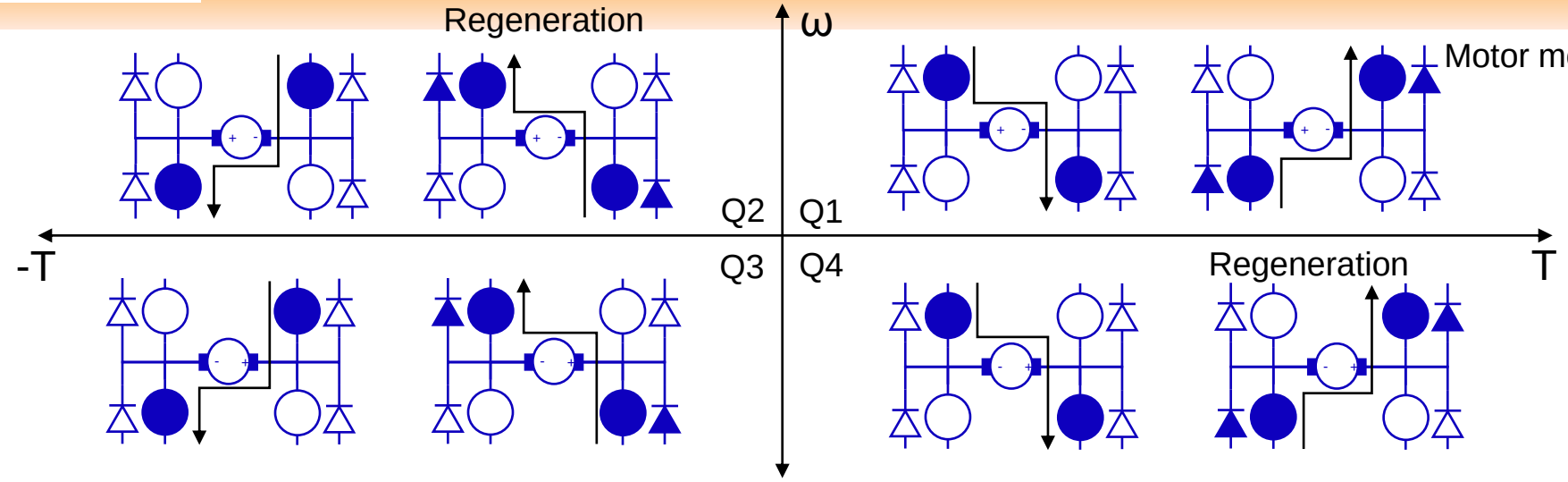
Advantages

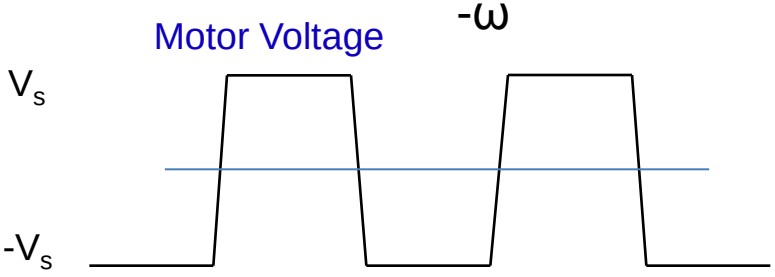
Control in all 4 quadrants.
Motor PWM frequency doubled.

Disadvantages

Deadtime generation is now required between four PWM signals.
Requires center aligned PWMs.

Bipolar Switching Modes





Advantages

Full control in all 4 quadrants.
No current re-circulation to complicate current detection and limiting.

Disadvantages

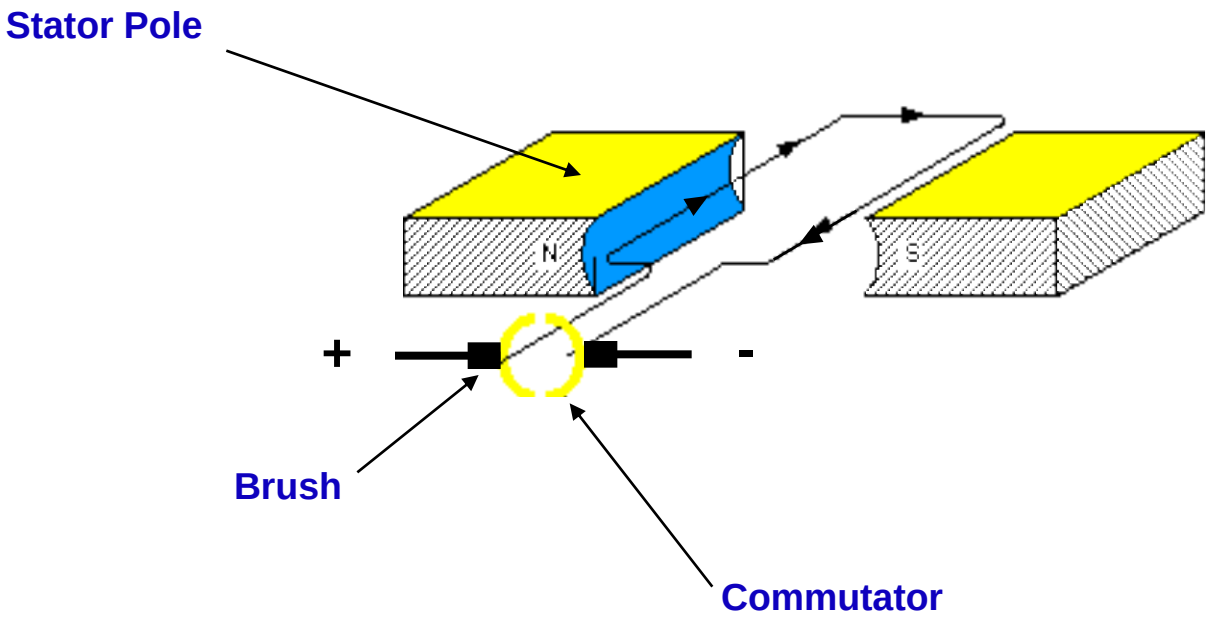
Deadtime generation is now required between two PWM signals.
ΔV seen by motor is 2Vs.

Agenda

- Basic Terms
- BLDC Motor Theory
  - BLDC motor principle
  - Basic control techniques
  - BLDC versus PMSM motor
  - Six step commutation
  - Commutation table derivation
- Sensorless Technique of BLDC Motors
- Microcontroller MC56F8006
- Freescale Software Library

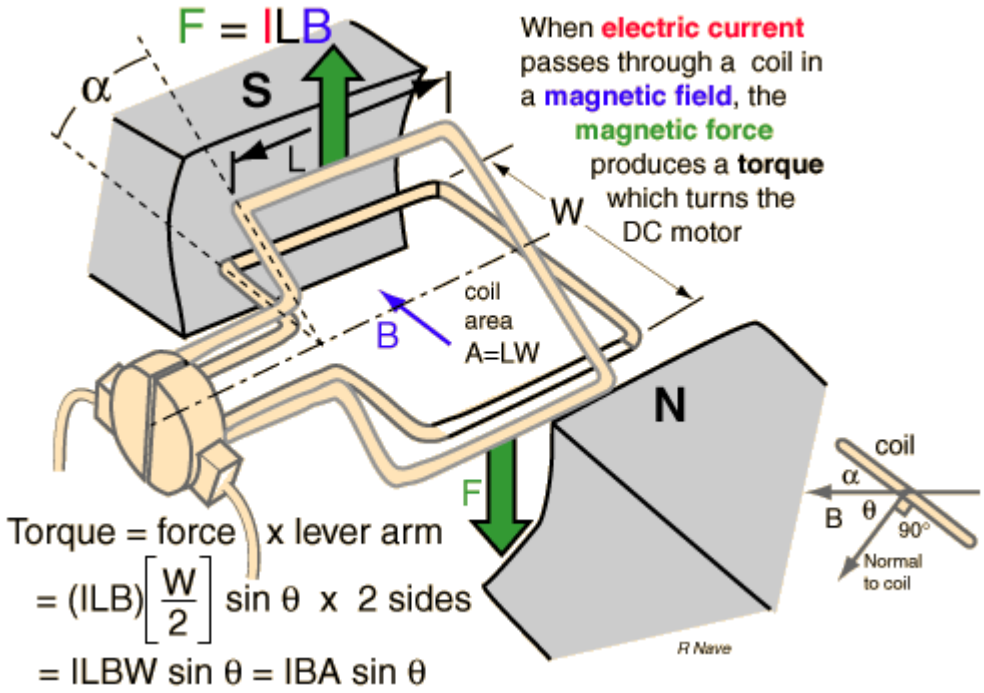
Brush DC Motor Principle

- Basic Structure



Brush DC Motor Principle

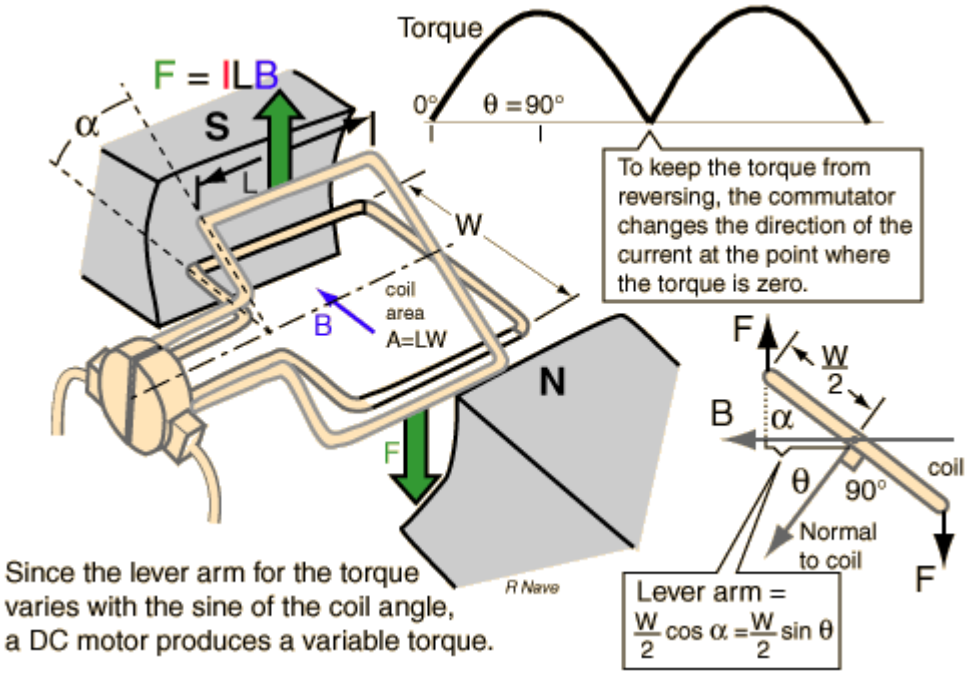
- Brush DC motor rotation



Source: http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/motdc.html#c1

Brush DC Motor Principle

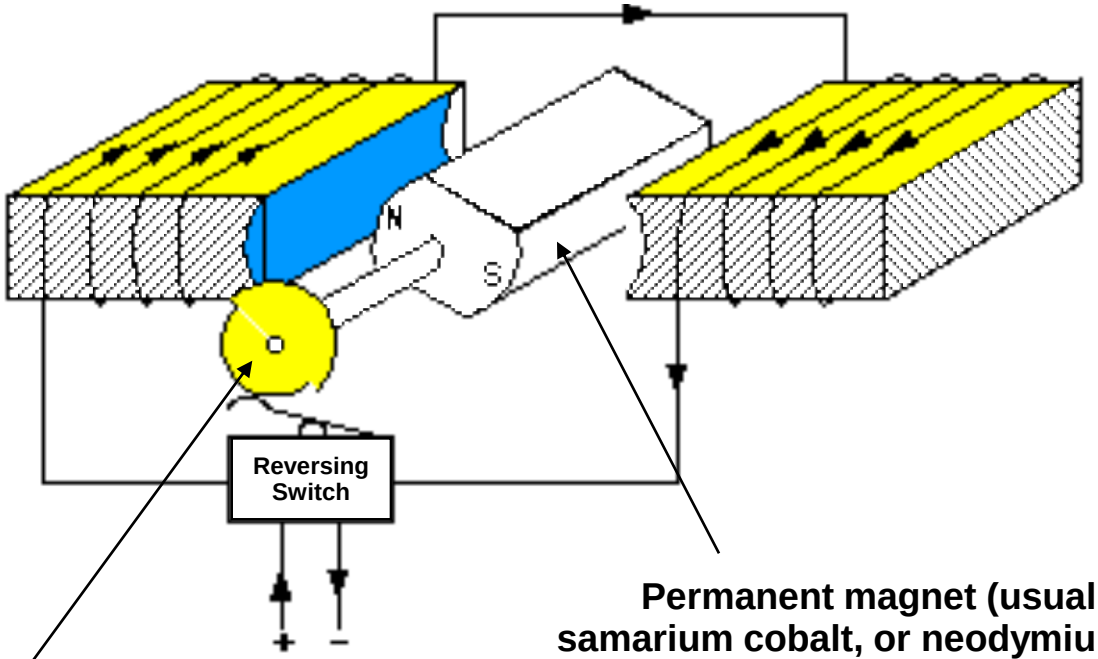
- Brush DC motor rotation



Source: http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/motdc.html#c1

Brushless DC Motor Principle

- Basic Structure



Permanent magnet (usually ferrite, samarium cobalt, or neodymium iron boron)

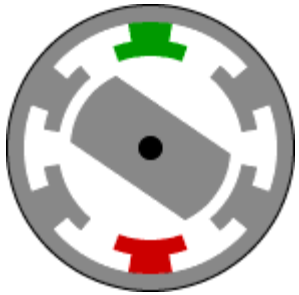
Requires mechanism to sense rotor position to commutate field properly

This is usually a hall effect sensor array or an encoder

Brushless DC Motor Principle

- Brushless DC motor rotation

- BLDC motor rotates because of the magnetic attraction between the poles of the rotor and the opposite poles of the stator.
- If the rotor poles are facing poles of the *opposite* polarity on the stator, a strong magnetic attraction is set up between them.
- The mutual attraction locks the rotor and stator poles together, and the rotor is literally yanked into step with the revolving stator magnetic field.



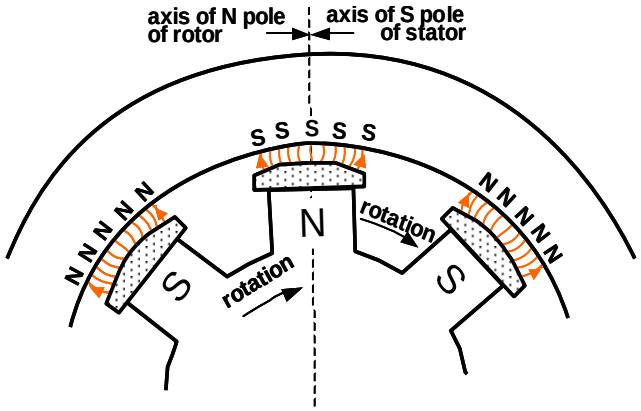


Figure 1- No-load condition

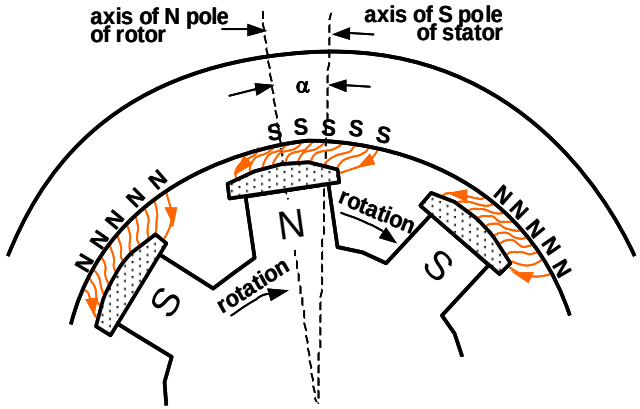
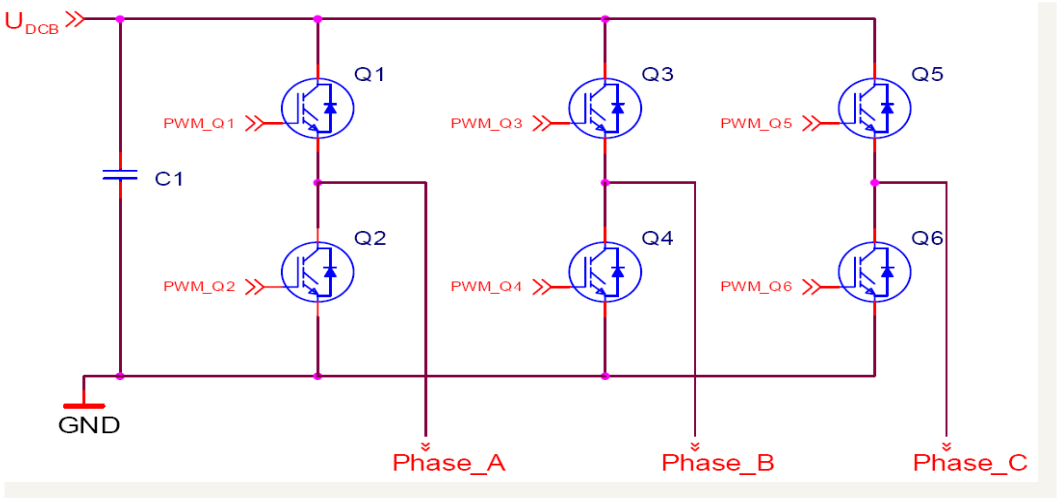


Figure 2- Load condition

Brushless DC Motor Control

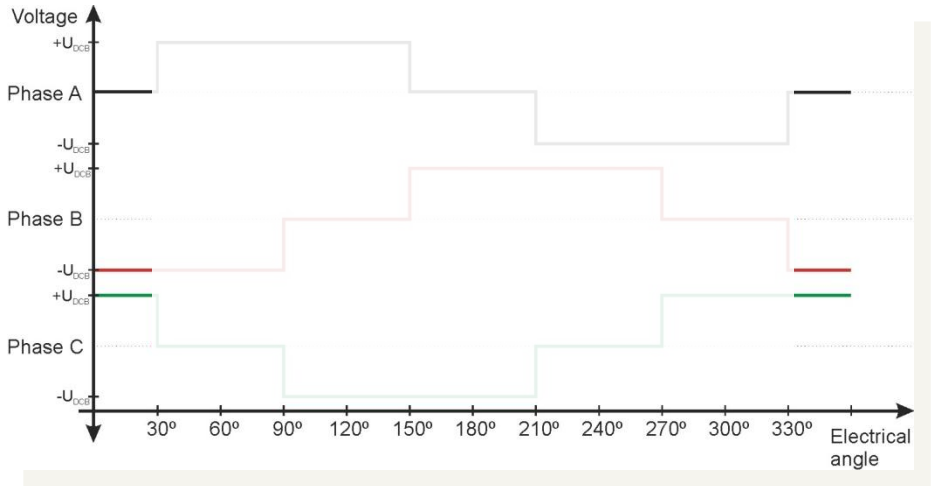
- Six Step BLDC Motor Control
  - Voltage applied on two phases only



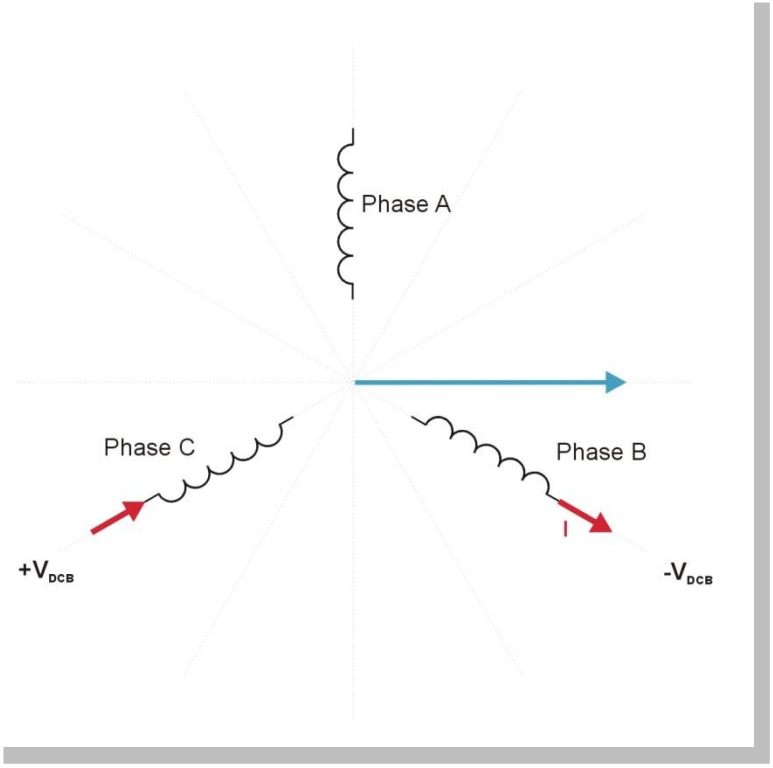
Power Stage

Brushless DC Motor Control

- Six Step BLDC Motor Control
  - Voltage applied on two phases only

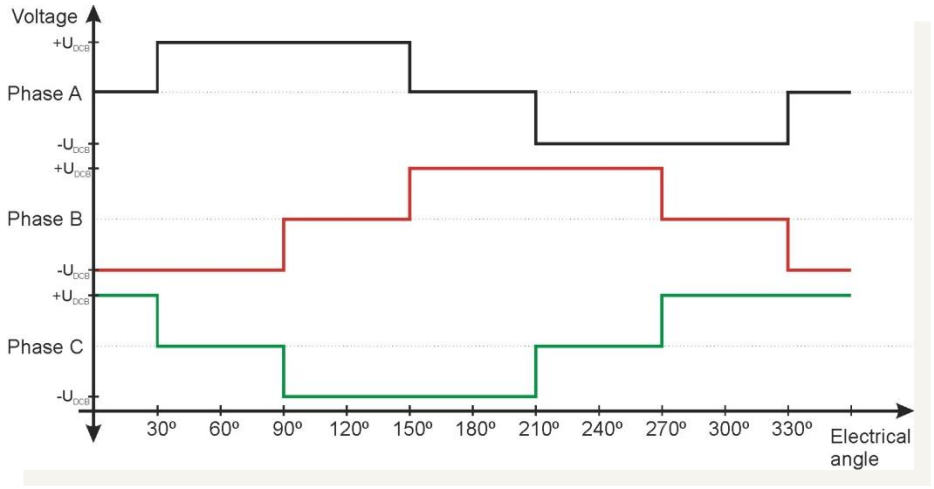


Phases voltage

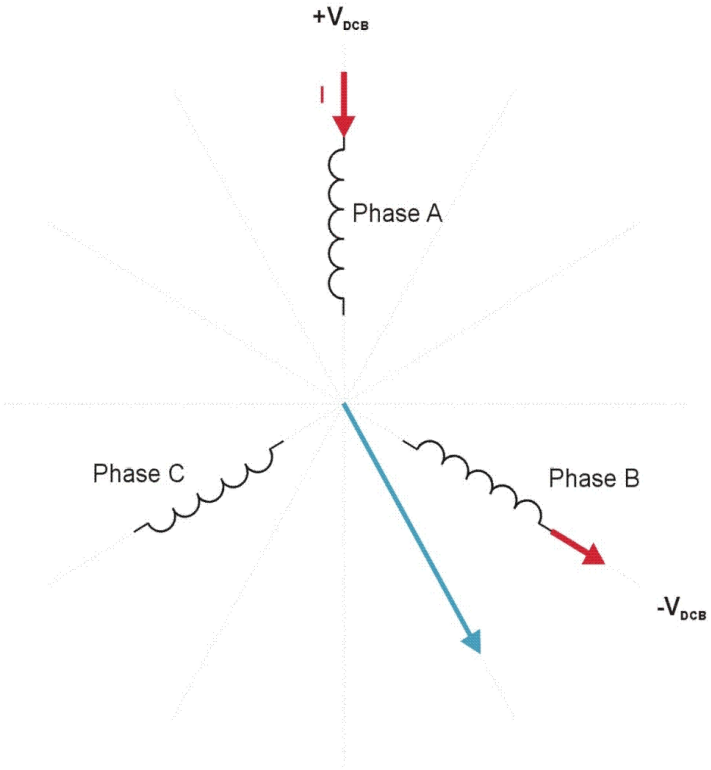


Brushless DC Motor Control

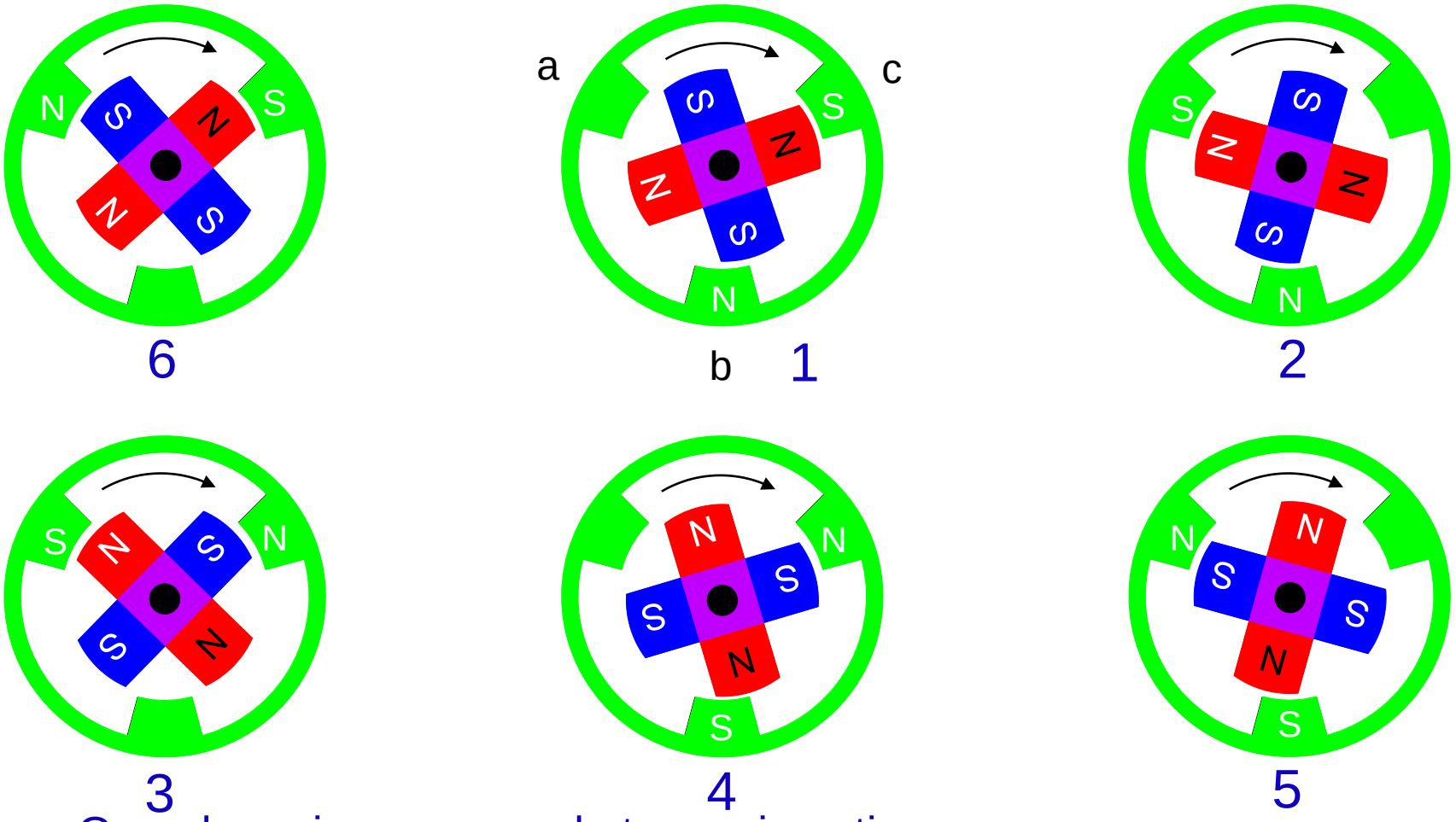
- Six Step BLDC Motor Control
  - Voltage applied on two phases only
  - It creates 6 flux vectors
  - Phases are power based on rotor position
  - The process is called Commutation



Phase voltages



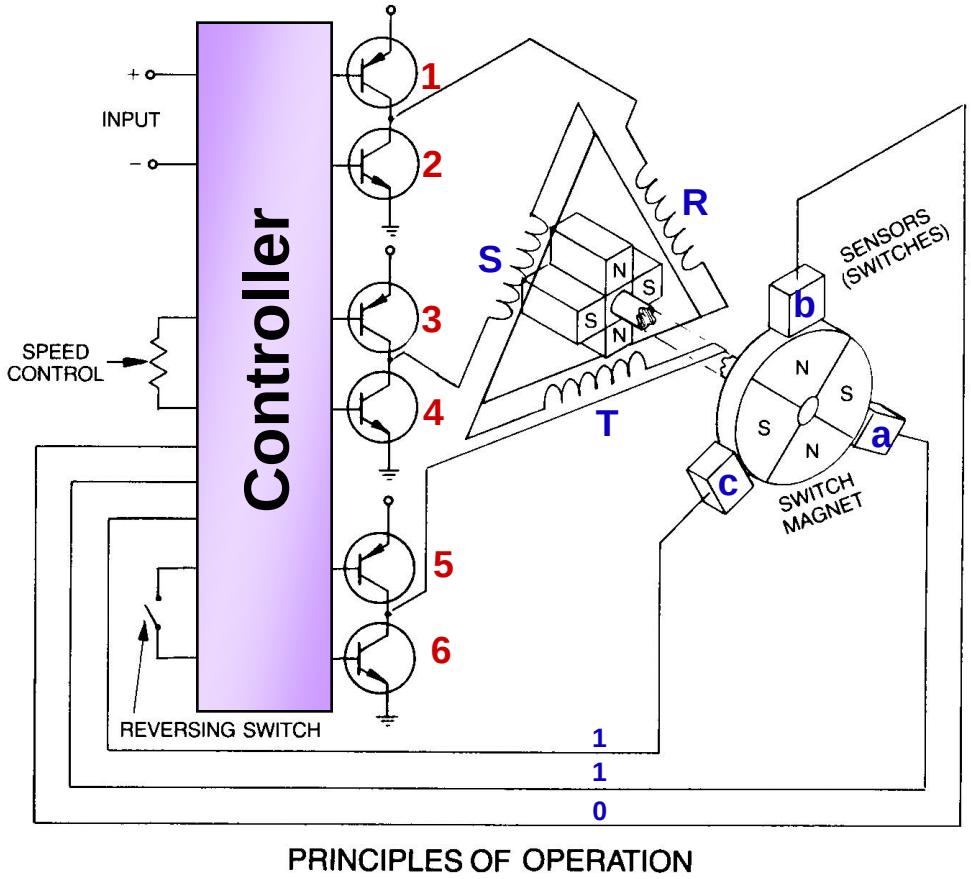
Commutation of a Brushless DC Motor



One phase is unpowered at any given time.

Brushless DC Motor Control

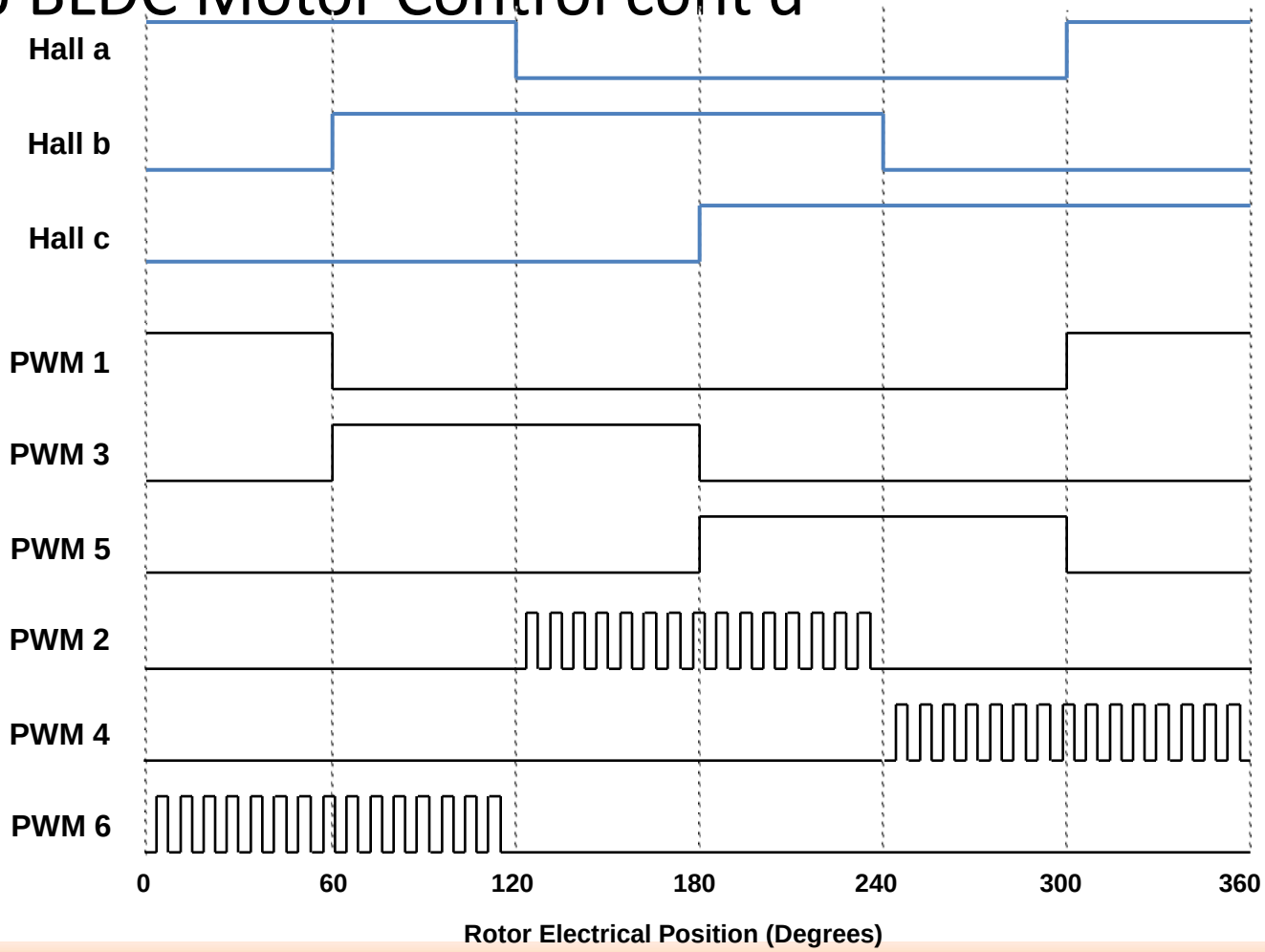
- Six Step BLDC Motor Control cont'd



Source: Eastern Air Devices, Inc. Brushless DC Motor Brochure

Brushless DC Motor Control

- Six Step BLDC Motor Control cont'd



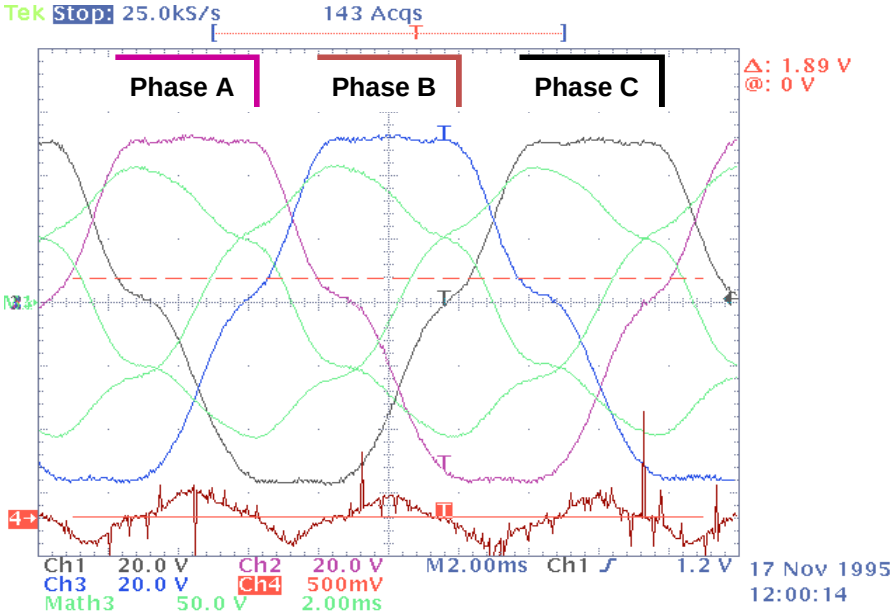
Brushless DC Motor Control Summary

- Six step control versus sinusoidal control

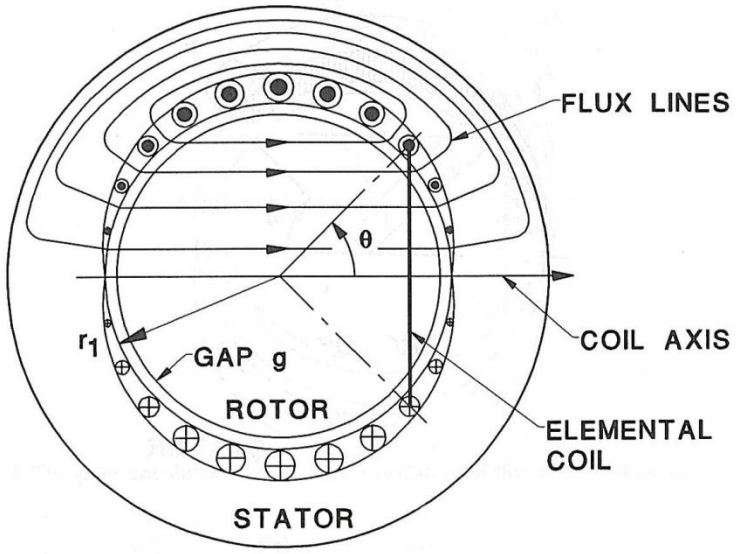
Six step control	Sinusoidal control
+ Simple PWM generation	− More complex PWM generation (sinewave has to be generated)
− Ripple in the torque (stator flux jumps by 60°)	+ Smooth torque (stator flux rotates fluently)
− A little noise operation (due to ripple in the torque)	+ Very quite
+ Simple sensor	− Requires sensor with high resolution

Brushless DC Motor Control

- BLDC Motor versus PMSM Motor
  - The both motors have identical construction. The difference is in stator winding only. The BLDC has distributed stator winding in order to have trapezoidal Back-EMF. The PMSM motor has distributed stator winding in order to have sinusoidal Back-EMF.



Trapezoidal Back-EMF voltage



Sinusoidal winding distribution

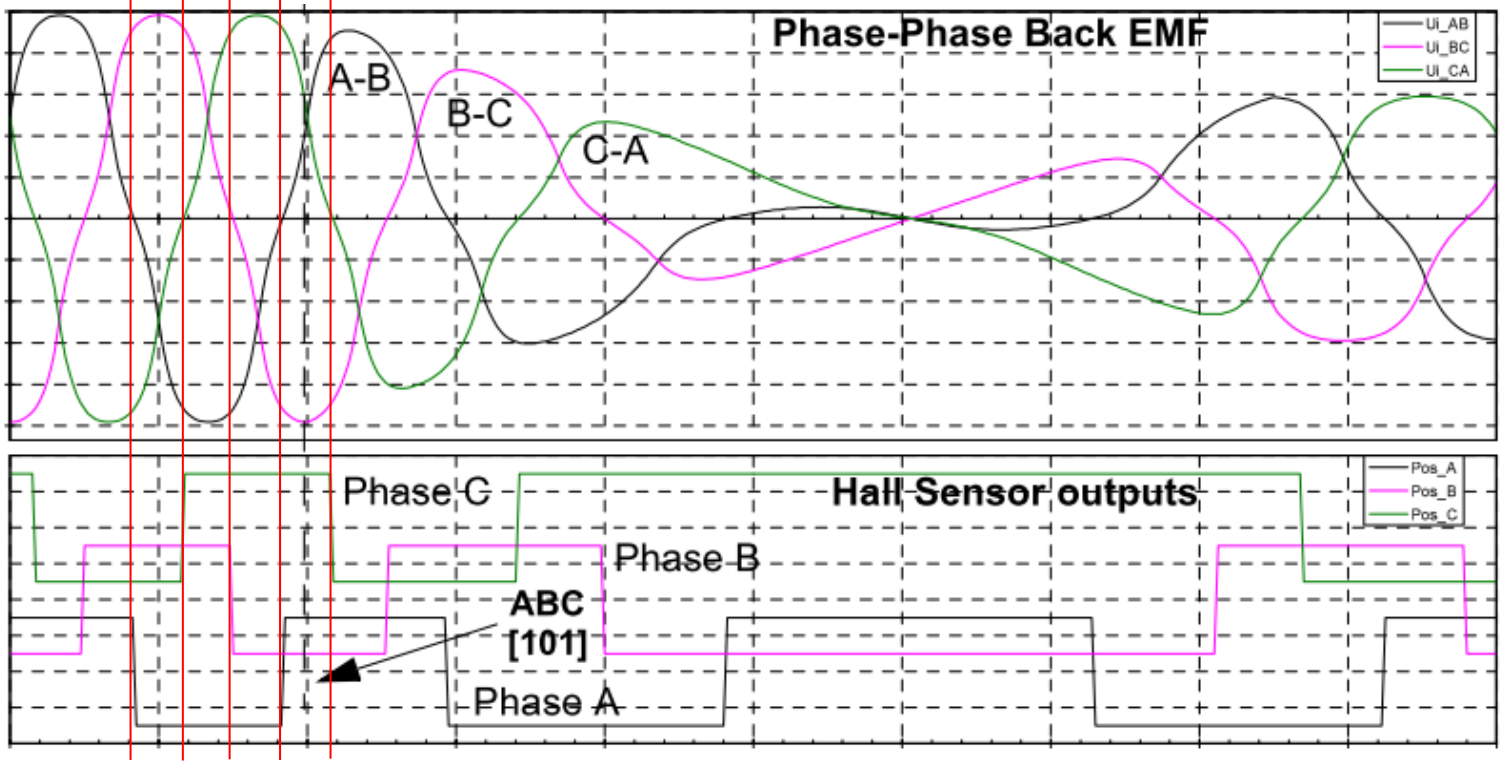
Source: Hendershot J. R. Jr, Miller TJE: Design of brushless permanent-magnet motors

Commutation Table

- The commutation table is fundamental for six step commutation BLDC control algorithm
- Motor commutates and thus rotates according to the commutation table
- The commutation table must be created depending on the actual motor configuration

Commutation Table

- Hall Sensors sense rotor flux and are aligned to phase to phase Back EMF voltage

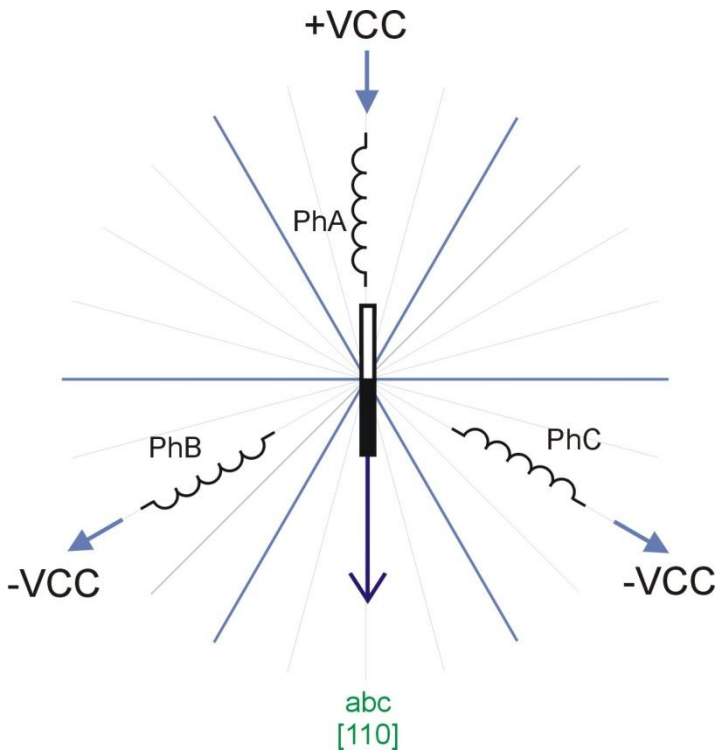


How to Get Commutation Table?

- There is a simple method using power supply with current limit
- Step 1 – Preparation for Hall Sensors measurement
  1. Mark all phases and all sensors as you want
  2. Set current limit of power supply to 20-30% of nominal motor current
  3. Choose direction of motor rotation (clockwise, counterclockwise)
  4. Connect any phase to “+” terminal
  5. Connect remain two phases to “-” terminal (All phases are always connected to power supply)

How to Get Commutation Table?

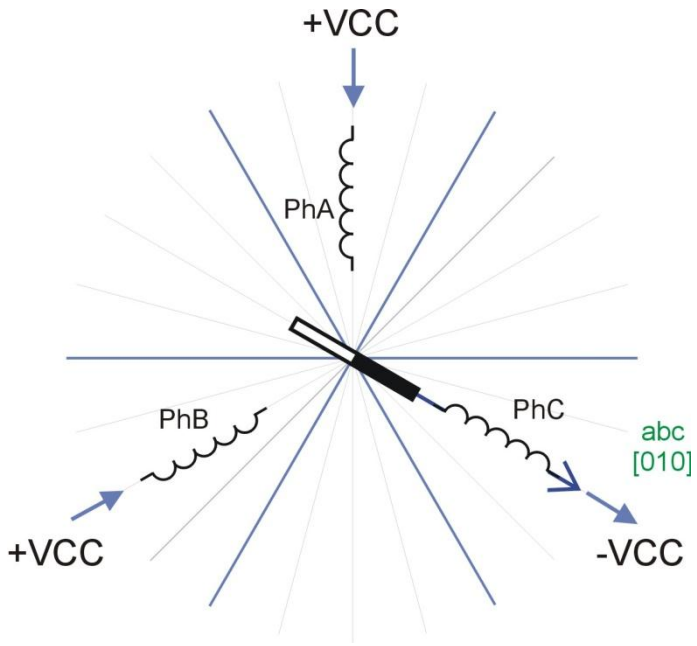
- Step 2 – Hall Sensors Measurement



Phase			Hall Sensor		
A	B	C	a	b	c
+	-	-	1	1	0

How to Get Commutation Table?

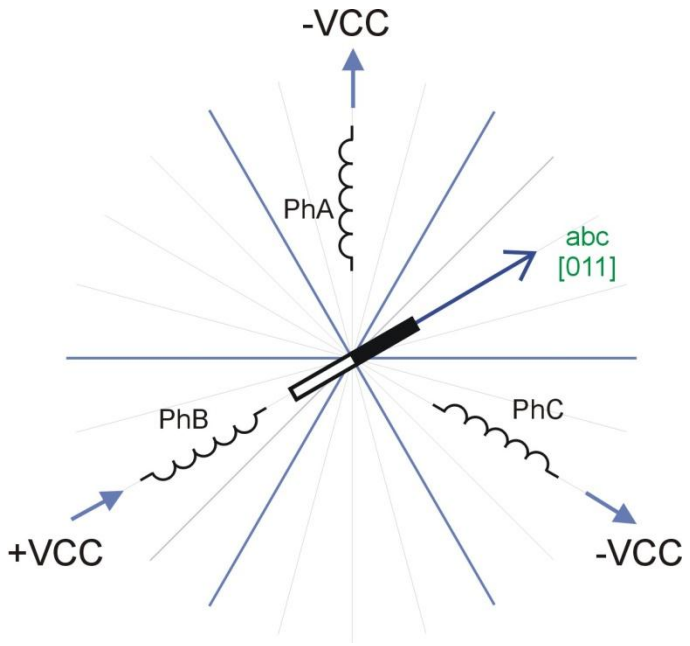
- Step 2 - Hall Sensors Measurement



Phase			Hall Sensor		
A	B	C	a	b	c
+	-	-	1	1	0
+	+	-	0	1	0

How to Get Commutation Table?

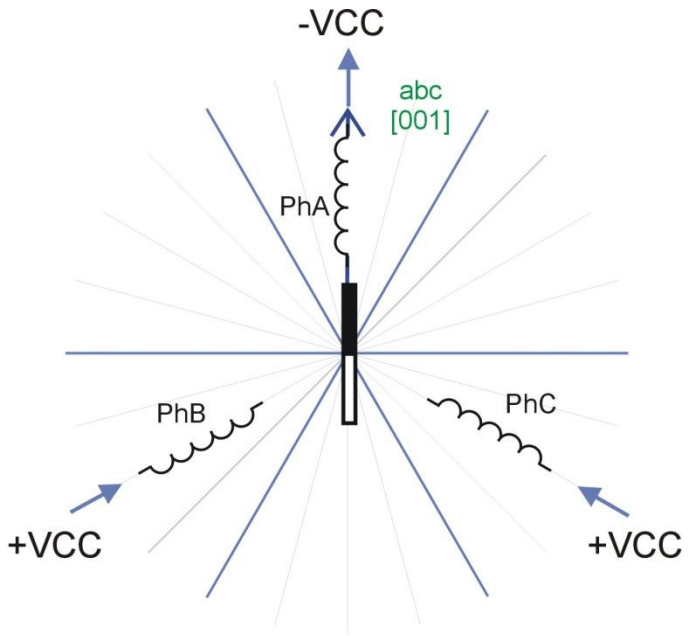
- Step 2 - Hall Sensors Measurement



Phase			Hall Sensor		
A	B	C	a	b	c
+	-	-	1	1	0
+	+	-	0	1	0
-	+	-	0	1	1

How to Get Commutation Table?

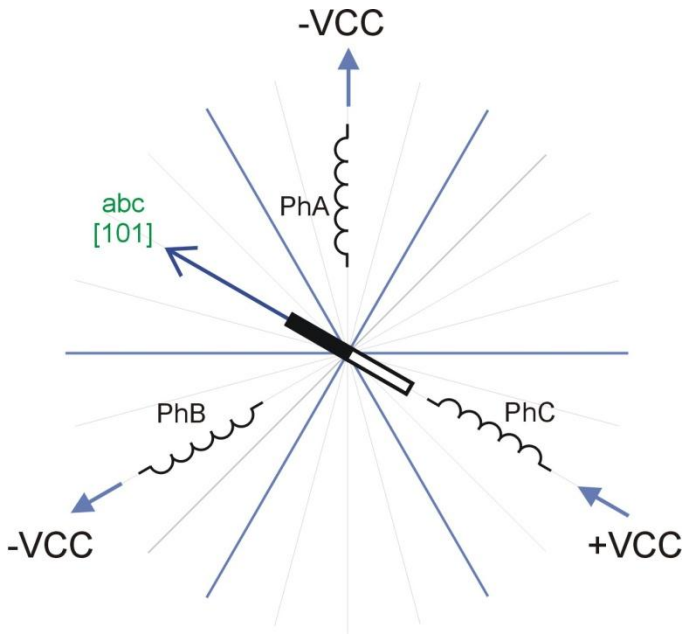
- Step 2 - Hall Sensors Measurement



Phase			Hall Sensor		
A	B	C	a	b	c
+	-	-	1	1	0
+	+	-	0	1	0
-	+	-	0	1	1
-	+	+	0	0	1

How to Get Commutation Table?

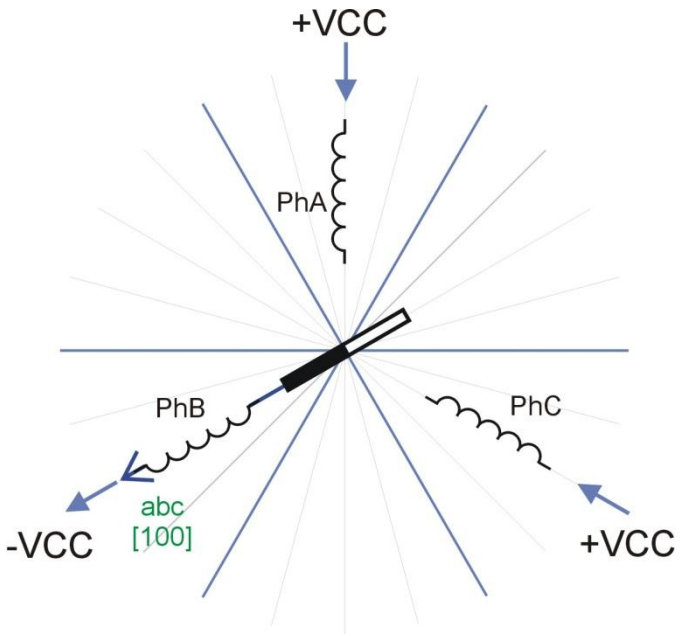
- Step 2 - Hall Sensors Measurement



Phase			Hall Sensor		
A	B	C	a	b	c
+	-	-	1	1	0
+	+	-	0	1	0
-	+	-	0	1	1
-	+	+	0	0	1
-	-	+	1	0	1

How to Get Commutation Table?

- Step 2 - Hall Sensors Measurement

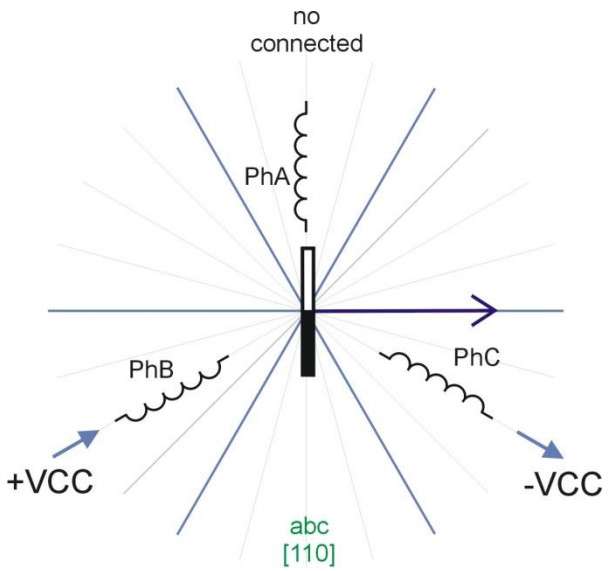


Phase			Hall Sensor		
A	B	C	a	b	c
+	-	-	1	1	0
+	+	-	0	1	0
-	+	-	0	1	1
-	+	+	0	0	1
-	-	+	1	0	1
+	-	+	1	0	0

This is not commutation table!!!

How to Get Commutation Table?

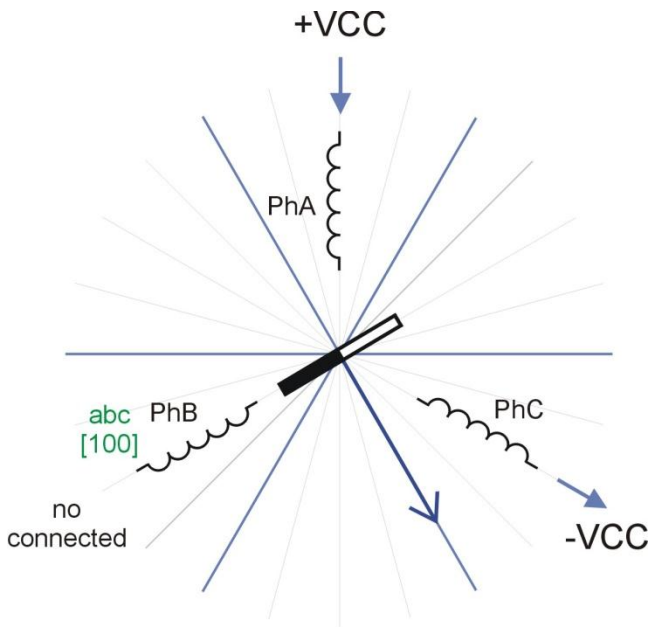
- Step 3 - Making Commutation Table



Hall Sensors			Phase		
a	b	c	A	B	C
1	1	0	NC	+	-
0	1	0			
0	1	1			
0	0	1			
1	0	1			
1	0	0			

How to Get Commutation Table?

- Step 3 - Making Commutation Table



Hall Sensors			Phase		
a	b	c	A	B	C
1	1	0	NC	+	-
0	1	0	-	+	NC
0	1	1	-	NC	+
0	0	1	NC	-	+
1	0	1	+	-	NC
1	0	0	+	NC	-

This is our commutation table

Agenda

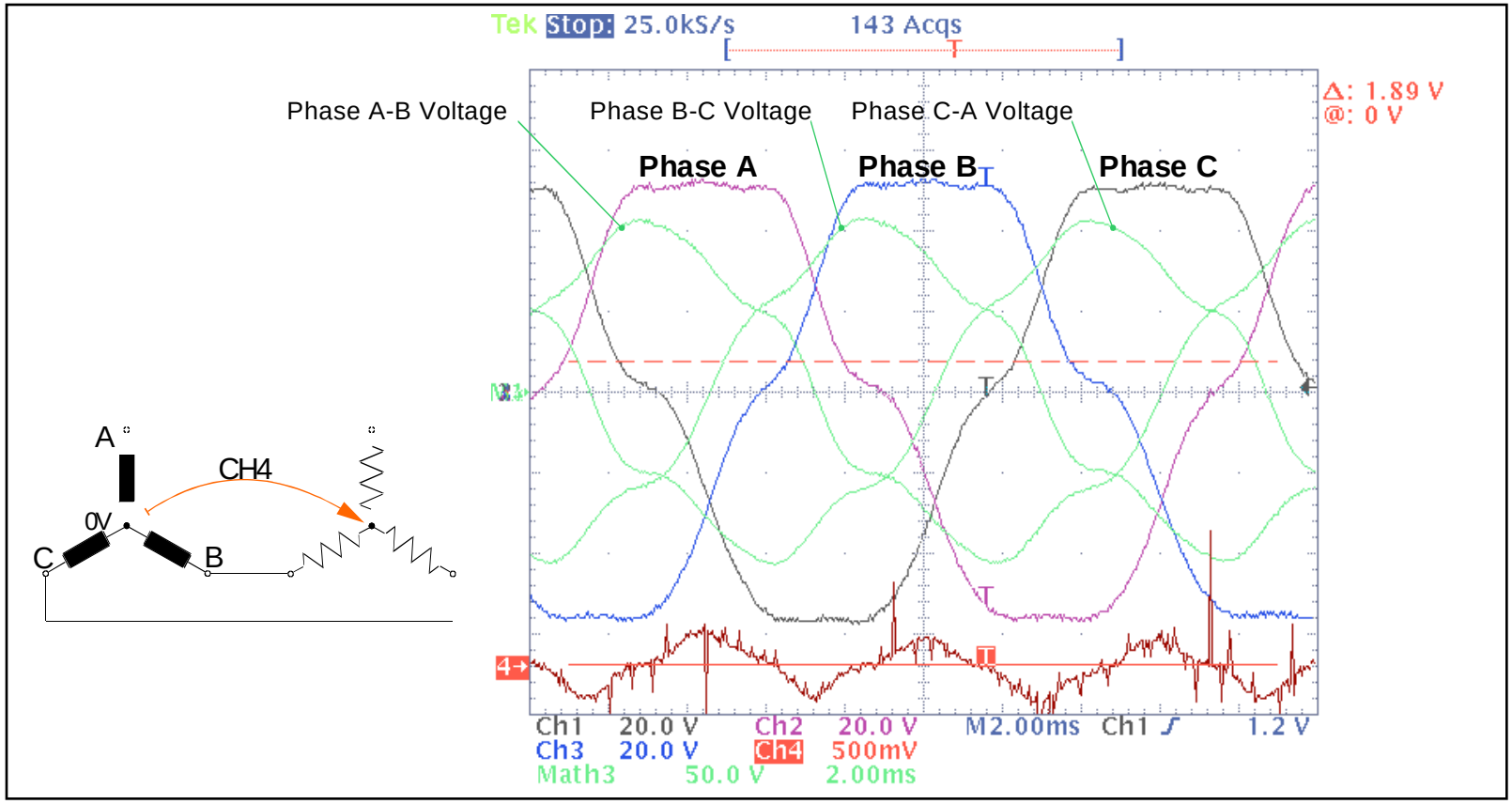
- Basic Terms
- BLDC Motor Theory
- Sensorless Technique of BLDC Motors
  - Principle of sensorless control
  - Rotor position estimation based on BEMF sensing
  - Zero crossing detection method
  - Implementation of zero crossing measurement
- Microcontroller MC56F8006
- Freescale Software Library

Techniques for BLDC Sensorless Position Estimation

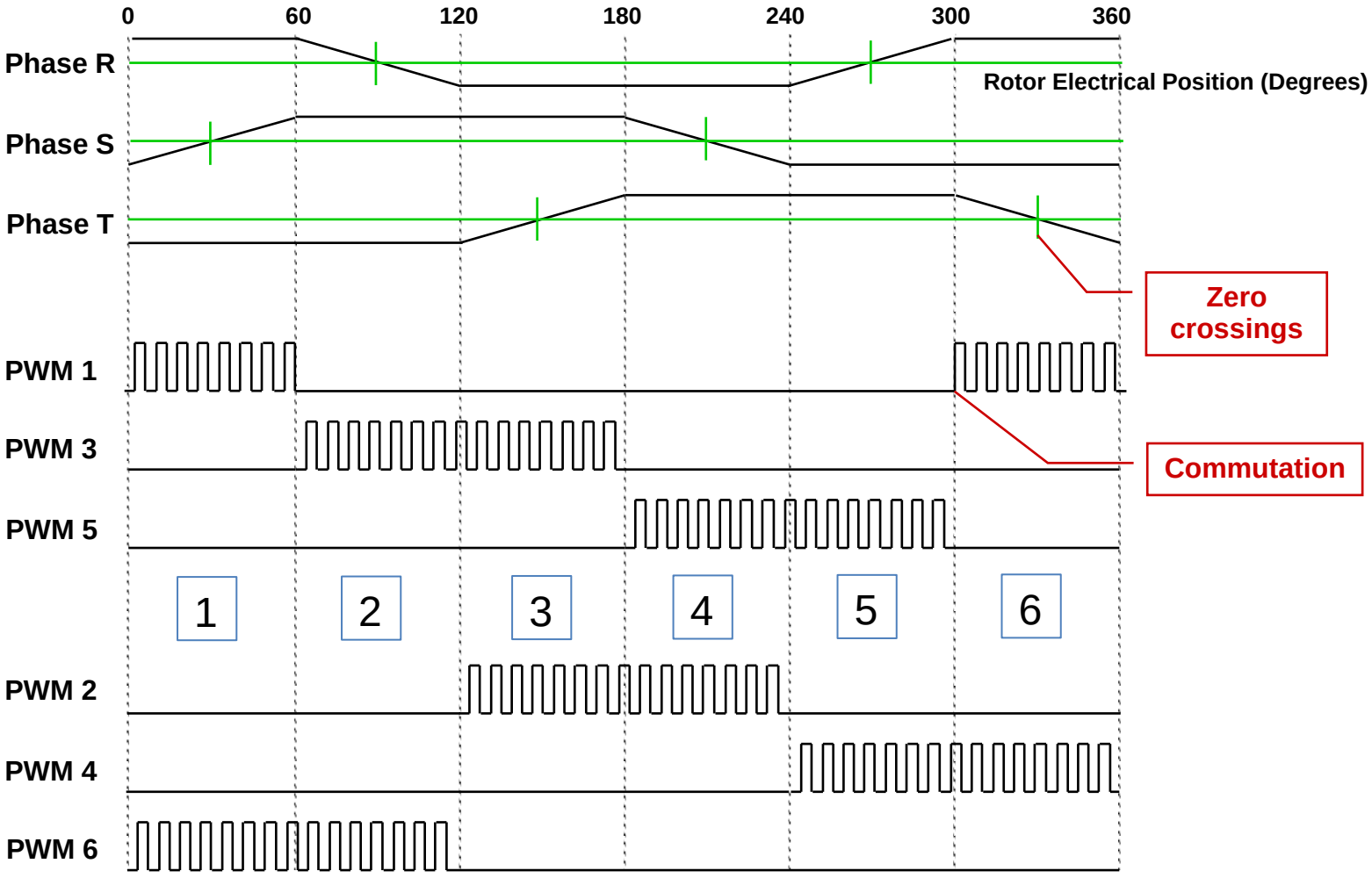
- Based on BEMF
  - Speed range from 5-10% up to 100% of nominal speed
    - The BEMF must be high enough
  - Techniques:
    - Based on BEMF Zero Crossing
    - Advanced Other BEMF estimation techniques
      - System Observers
      - Measurement non-conductive phase with oversampling
- Based on Motor Inductance Saliency
  - Speed range from standstill to about 20% of nominal speed
  - Techniques
    - Simple techniques - based on current transient measurement
    - Advanced techniques - with current injection
      - Requires powerful core and exact current sensing

- Combination of the two techniques could be used to control the BLDC motor from standstill to nominal speed

BLDC Motor Position Sensing based on Back-EMF Voltage

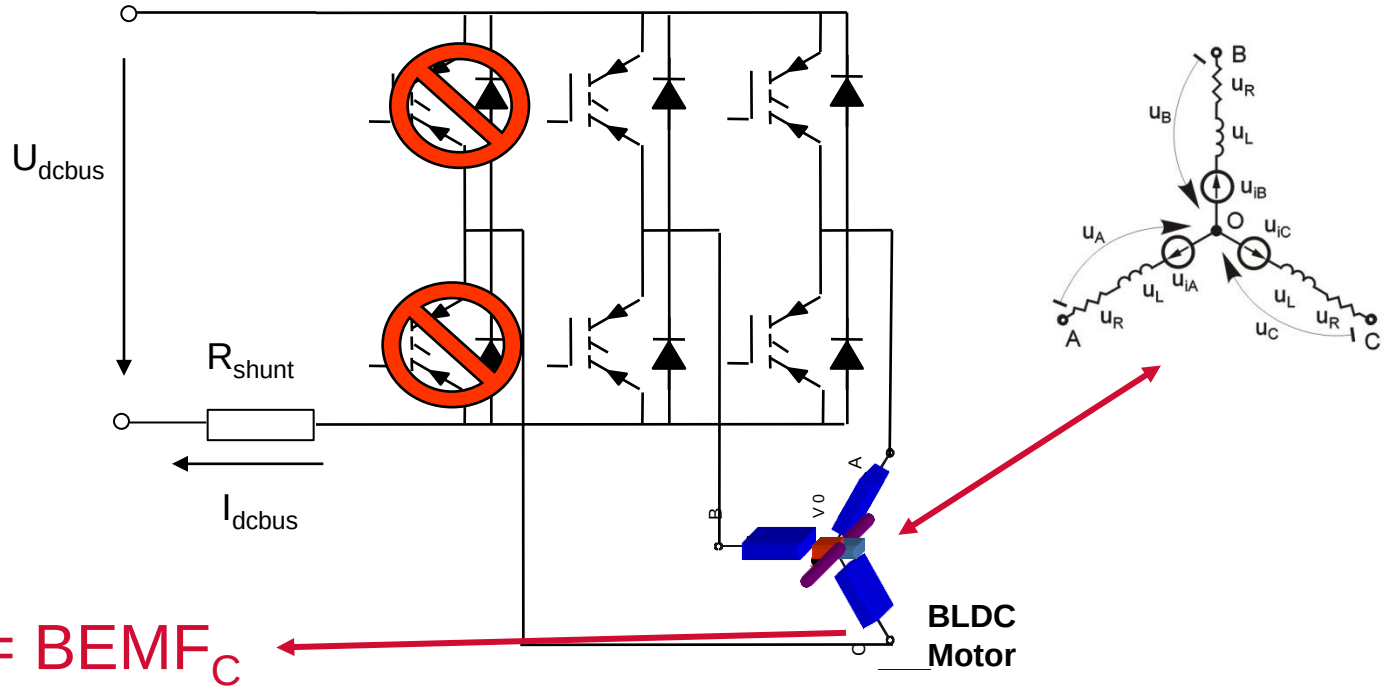


BEMF Voltage Zero Crossing Sensing



BEMF Voltage Zero Crossing Sensing Condition

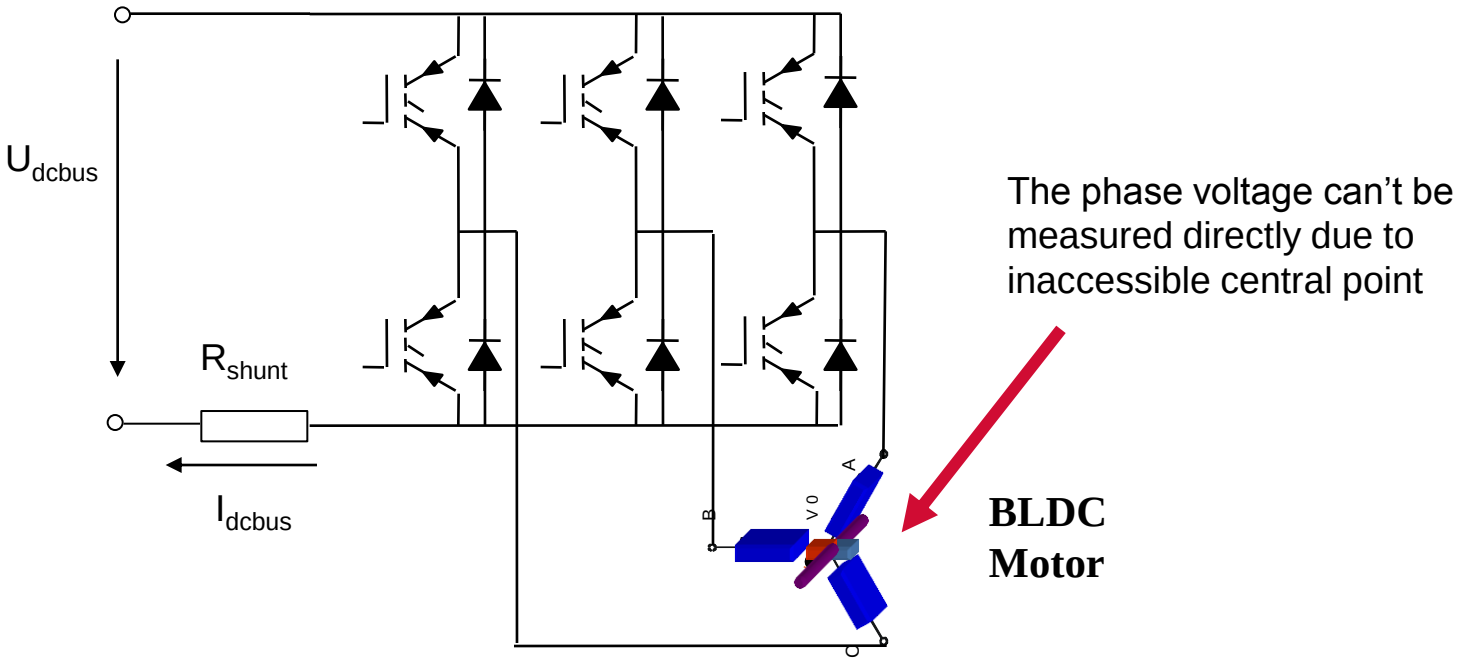
1. The measured phase is disconnected from power source
2. There is no current flowing through measured phase



$U_C = BEMF_C$

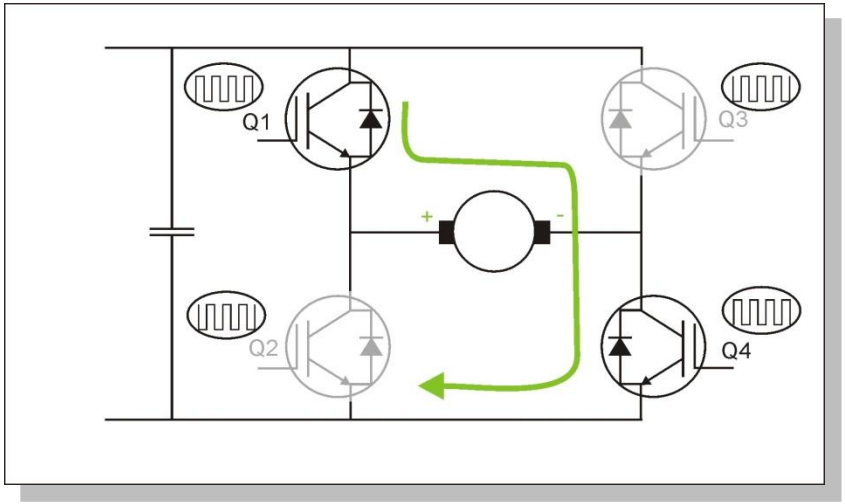
BEMF Voltage Zero Crossing Sensing Condition

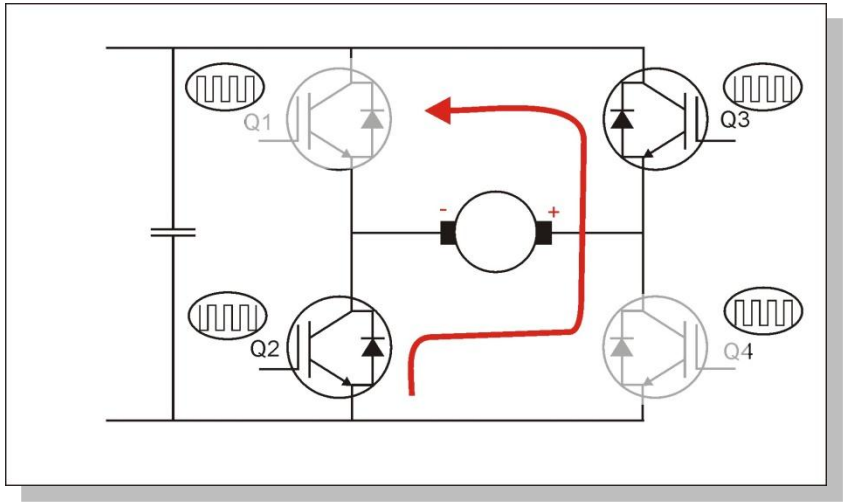
- BLDC Central Point Is Not Usually Accessible



Impact of PWM Modulation on BEMF Voltage Sensing

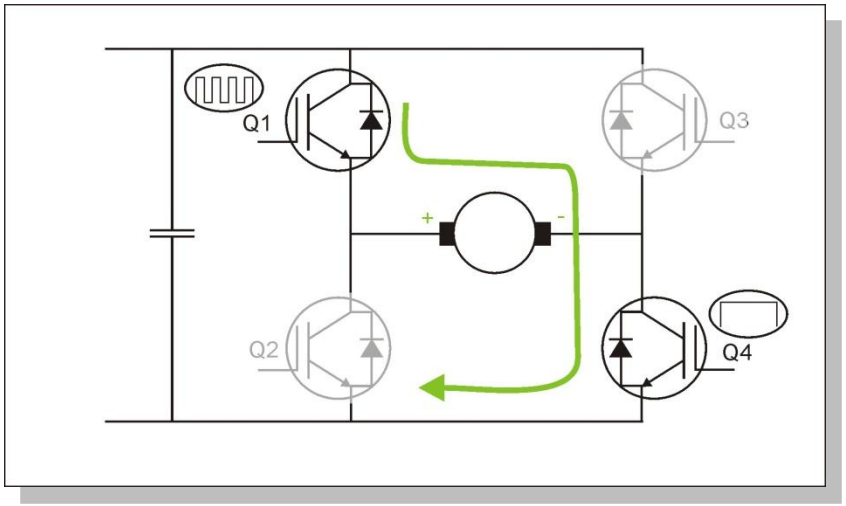
- Bipolar Switching
  - One phase is always connected to positive V_{DCB}, second one is always connected to negative V_{DCB}
  - The conditions for BEMF sensing are the same over whole PWM period

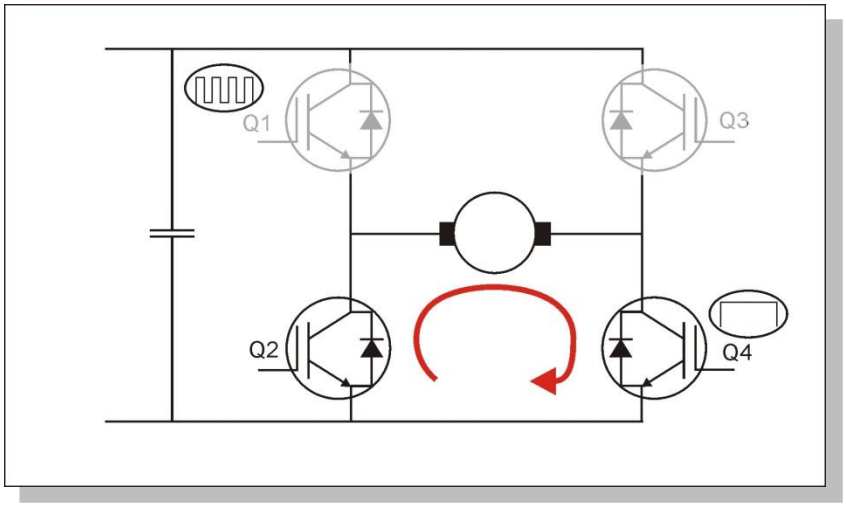




Impact of PWM Modulation on BEMF Voltage Sensing

- Unipolar Switching
  - One phase is always connected to negative V_{DCB}, second one alternates between positive and negative V_{DCB}
  - The conditions for BEMF sensing change in dependency on state if Q1



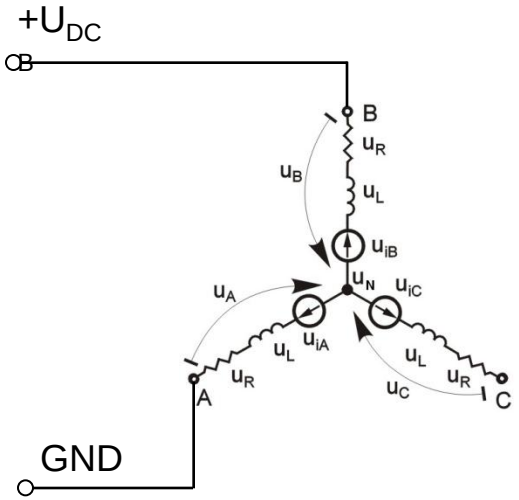


Impact of PWM Modulation on BEMF Voltage Sensing

- Two possible circuit configurations have to be considered for BEMF Sensing:
  - Both transistors in diagonal are switched ON.
  - The only bottom transistor is switched ON. The top transistor is switched OFF.

BEMF Voltage Zero Crossing Methods

- Both transistors in diagonal are switched ON.



$$u_N = u_{DCB} - ri - L\frac{di}{dt} - u_{iB} \qquad u_N = ri + L\frac{di}{dt} - u_{iA}$$

$$u_N = \frac{u_{DCB}}{2} - \frac{u_{iB} + u_{iA}}{2}$$

$$u_{iA} + u_{iB} + u_{iC} = 0 \quad \Rightarrow \quad u_N = \frac{u_{DCB}}{2} + \frac{u_C}{2}$$

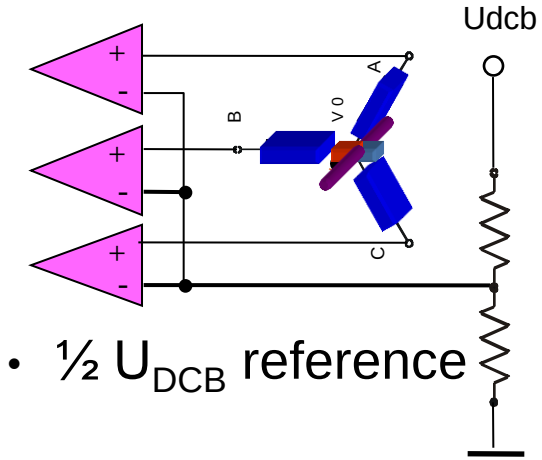
$$u_C = u_N + u_{iC}$$

$$u_C = \frac{3}{2}u_{IC} + \frac{u_{DCB}}{2} \quad \Rightarrow \quad u_C = \frac{u_{DCB}}{2}$$

zero at
zero crossing

BEMF Voltage Zero Crossing Methods

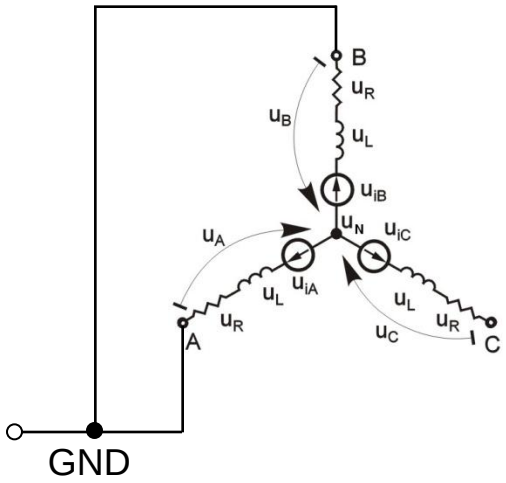
- Method 1: Comparison of BEMF voltage with half of DC bus voltage
- Assumptions:
  - The measured phase is disconnect from power supply
  - There is no current in measured phase
  - Can be sensed when both transistors in diagonal are switched ON
- This method can be used for both bipolar even unipolar PWM



- ½ U_{DCB} reference

BEMF Voltage Zero Crossing Methods

- Only bottom transistor is switched ON.



$$u_N = 0 - ri - L\frac{di}{dt} - u_{iB} \qquad u_N = ri + L\frac{di}{dt} - u_{iA}$$

$$u_N = -\frac{u_{iB} + u_{iA}}{2}$$

$$u_{iA} + u_{iB} + u_{iC} = 0 \quad \Leftrightarrow \quad u_N = \frac{u_C}{2}$$

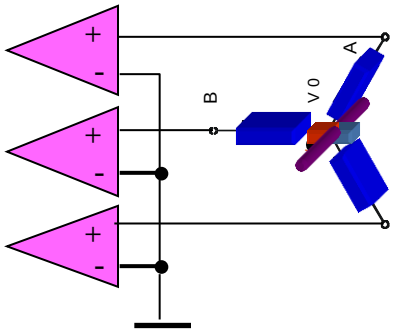
$$u_C = u_N + u_{iC}$$

$$u_C = \frac{3}{2}u_{IC} \quad \Leftrightarrow \quad u_C = 0$$

zero at
zero crossing

BEMF Voltage Zero Crossing Methods

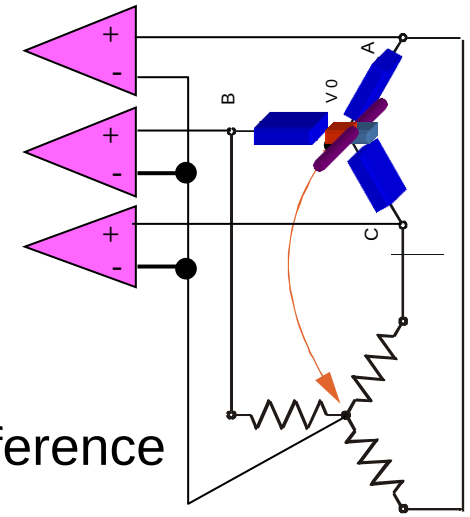
- Method 2: Comparison of BEMF voltage with zero voltage
- Assumptions:
  - The measured phase is disconnect from power supply
  - There is no current in measured phase
  - Can be sensed when bottom transistor is switched ON
- This method can be used for unipolar PWM only



- GND reference

BEMF Voltage Zero Crossing Methods

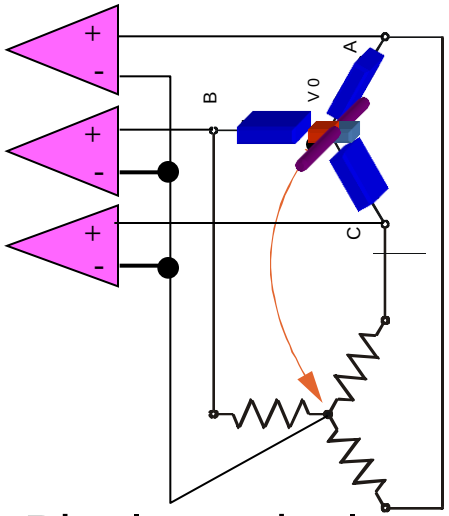
- Method 3: Comparison of BEMF voltage with virtual central point
- Assumptions:
  - The measured phase is disconnect from power supply
  - There is no current in measured phase
  - Can be sensed any time
- This method can be used for both bipolar even unipolar PWM



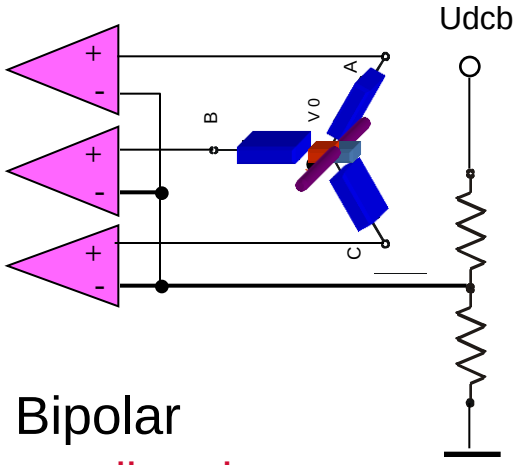
- Virtual CP reference

BEMF Voltage Zero Crossing Methods

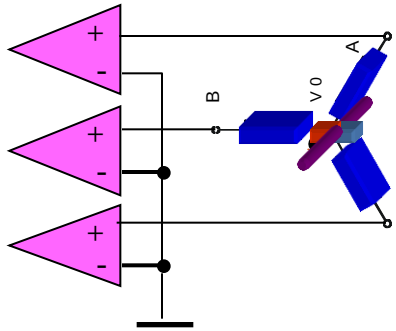
- BEMF Voltage Zero Crossing Methods - Summary
  - Virtual CP reference



    - Bipolar, unipolar
      - All cycles
    - Drawbacks
      - reference disturbed
      - offset error (LV)
  - ½ U_{DCB} reference



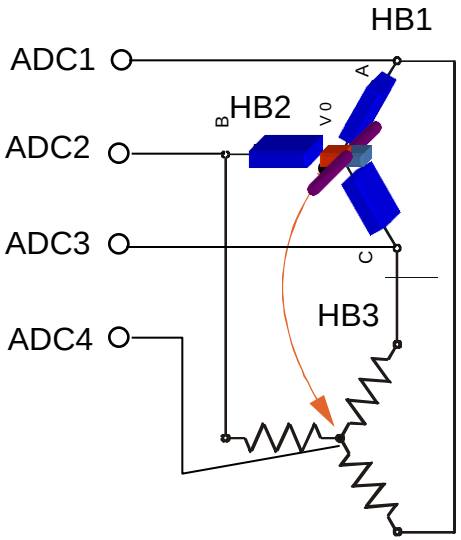
    - Bipolar
      - all cycles
    - Unipolar
      - Both transistors ON
    - Drawbacks:
      - Low sensing period at low speed
  - GND reference

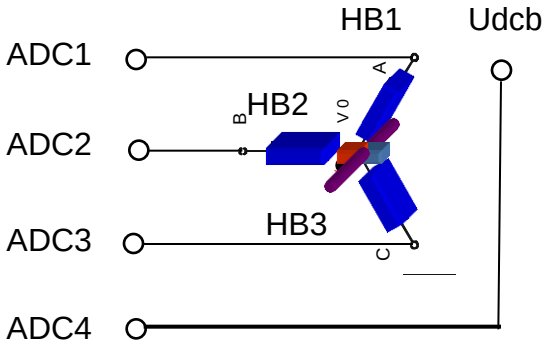


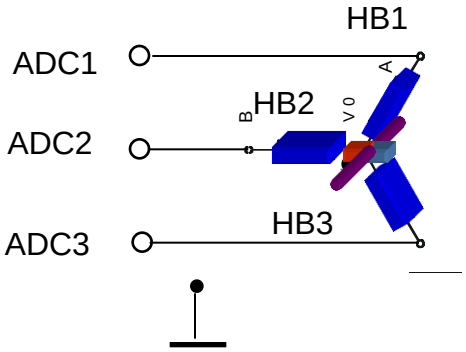
    - Bipolar
      - No sensing
    - Unipolar
      - Bottom transistor ON
    - Drawbacks:
      - offset error (LV)
      - Low sensing period at high speed

Zero Crossing Sensing using ADC

- The principle is same as the HW topology, but more flexible
- Virtual CP reference
- ½ U_{DCB} reference
- GND reference



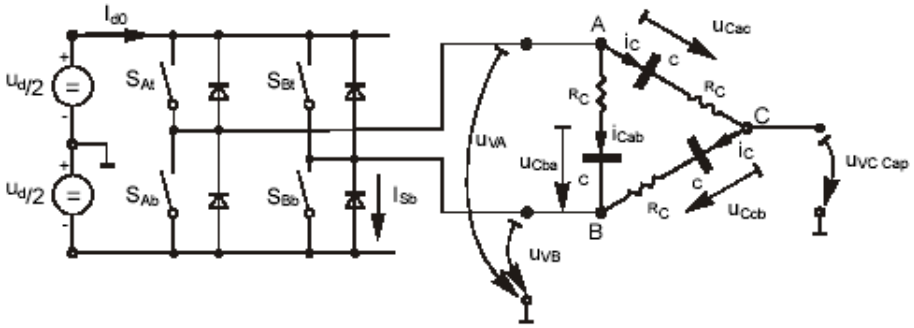


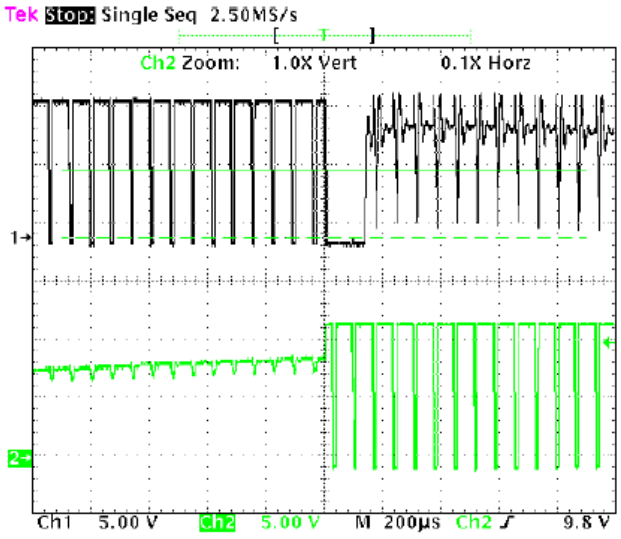


- The features same as:
  - Zero Crossing Sensing Comparators - Topologies

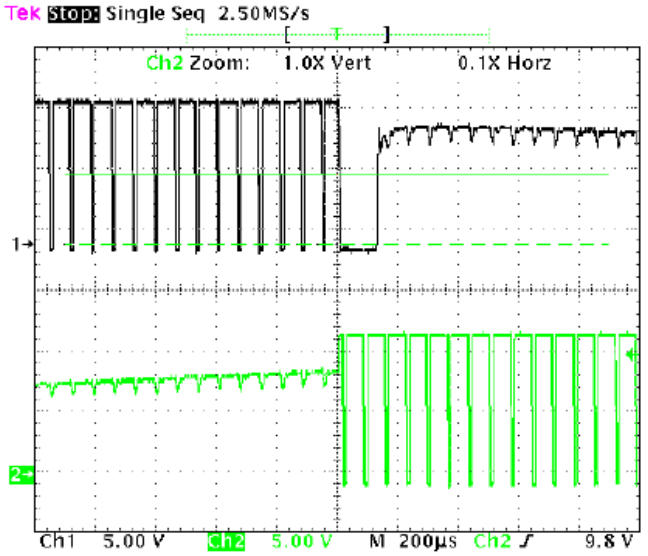
Zero Crossing Detection and PWM Switching Noise

- Phase Mutual Capacity Effect





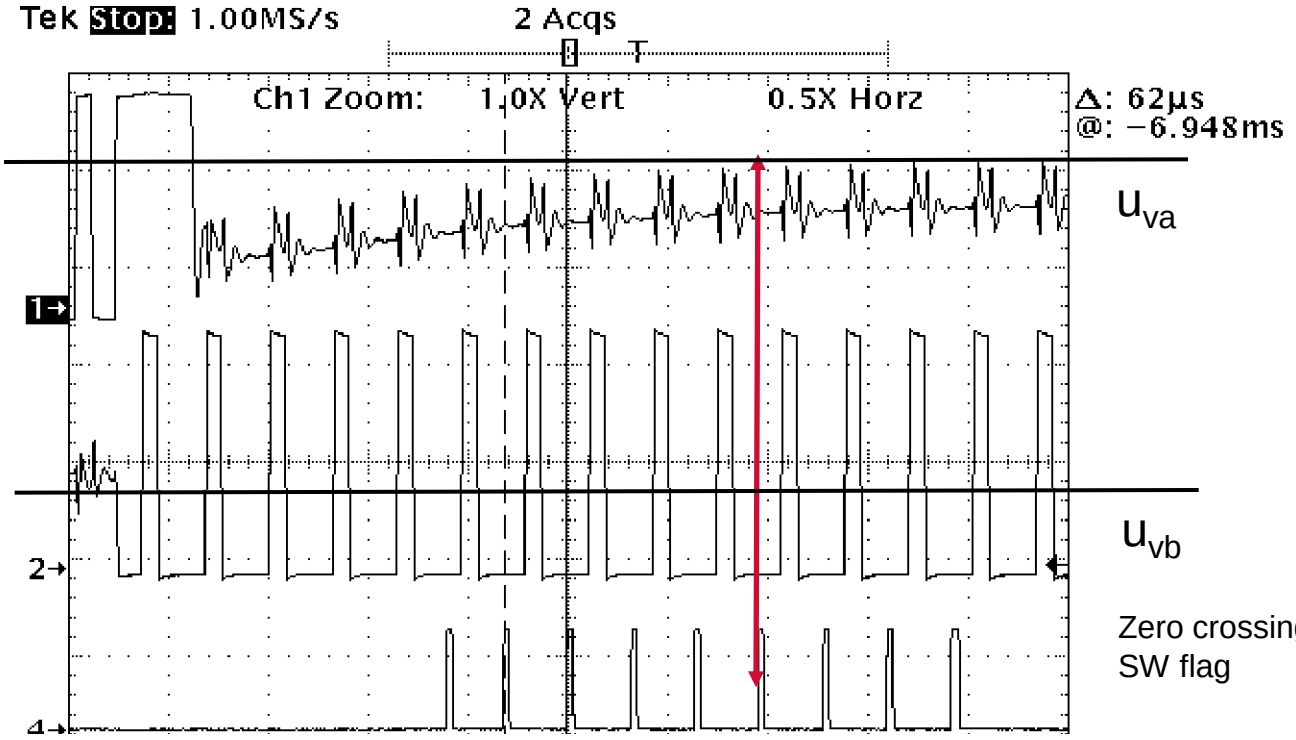
- Unbalanced



- Balanced

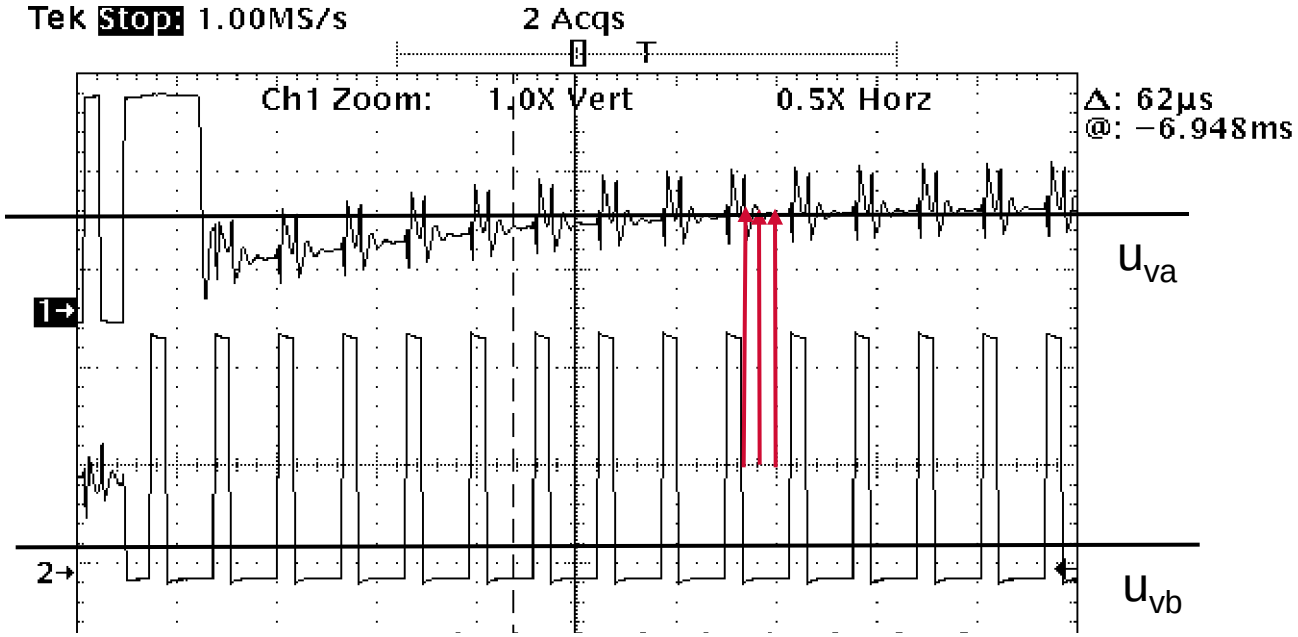
Zero Crossing Synchronization

- BEMF zero crossing synchronization with PWM



Zero Crossing Oversampling

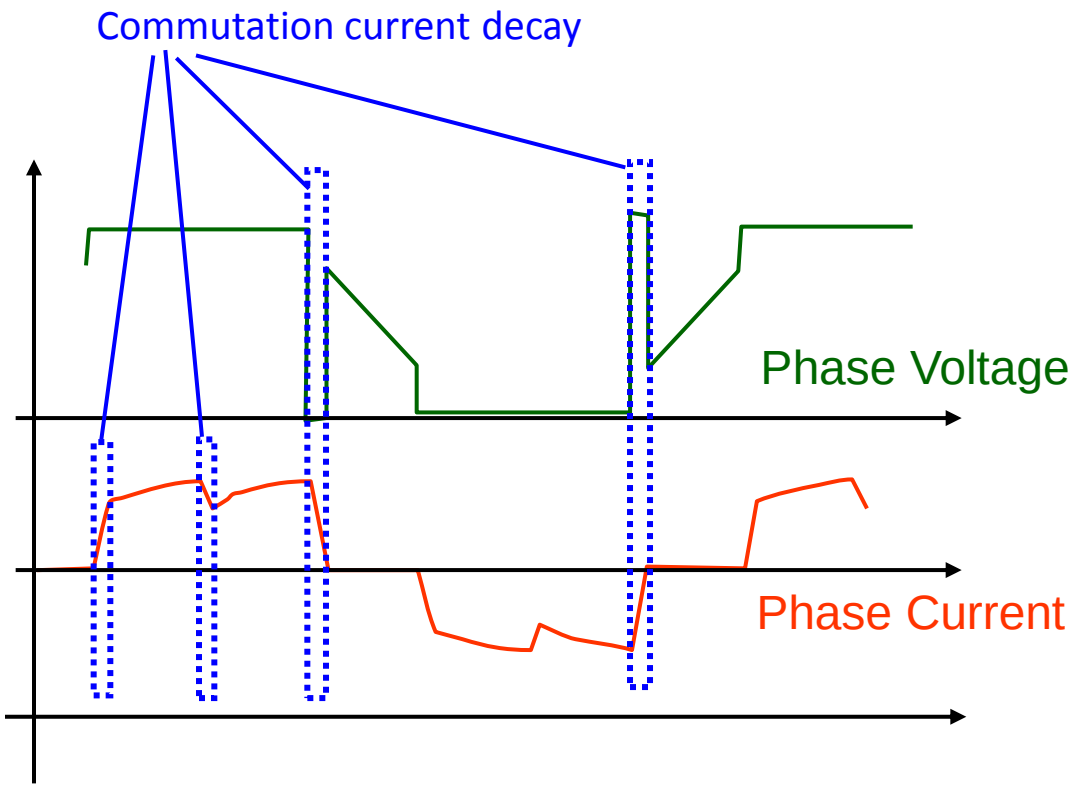
- BEMF zero crossing sampling

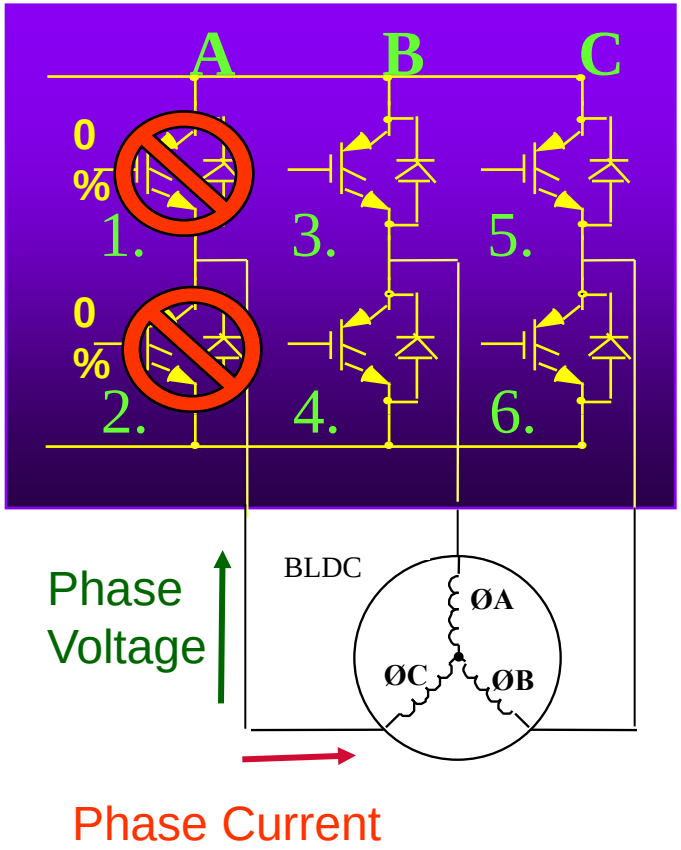


ZC Sampling Cycle1

Zero Crossing Detection - Limitation

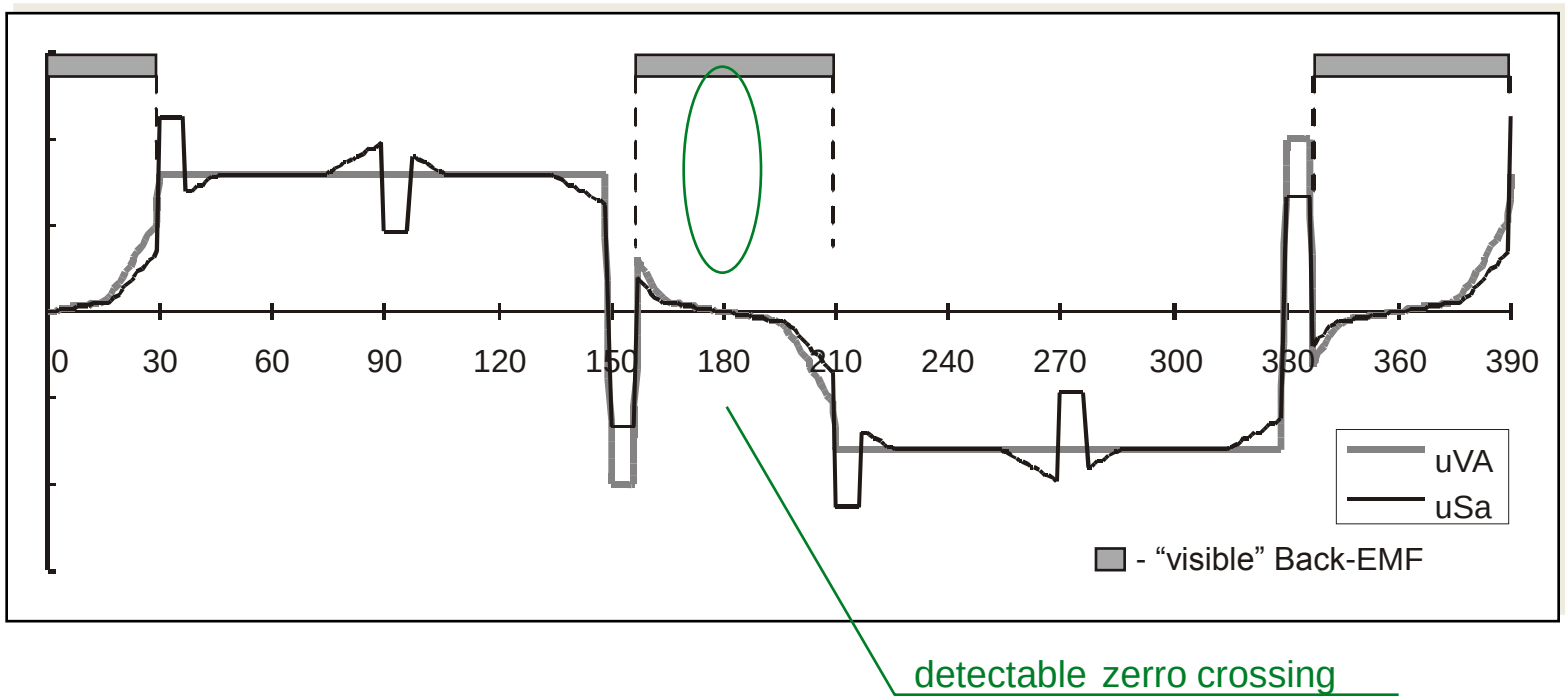
- Commutation Transient



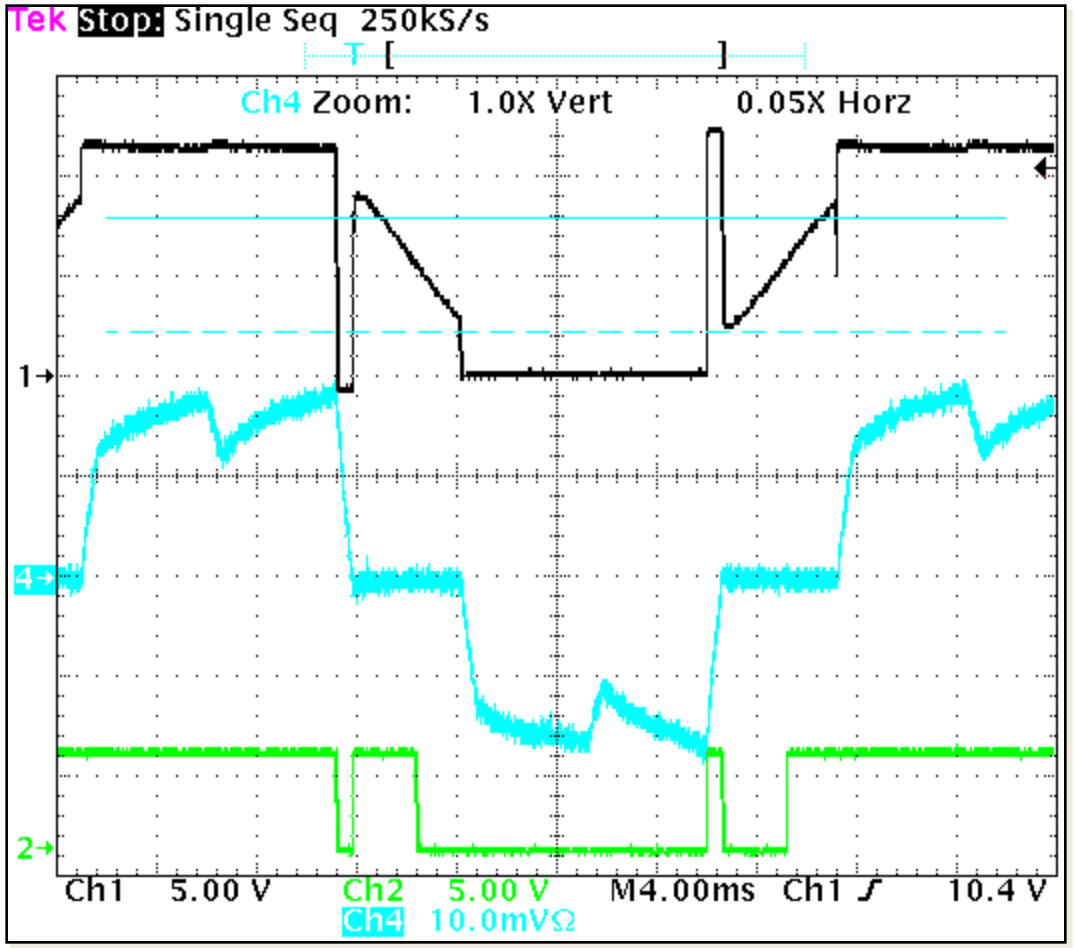


Zero Crossing Detection - Limitation

- Zero crossing window



Steady State at Constant Speed



Channel1 - “branch” phase voltage C

Channel 4 - phase current C (0,5 A/d).

Channel 2 HW zero-crossing detection signal

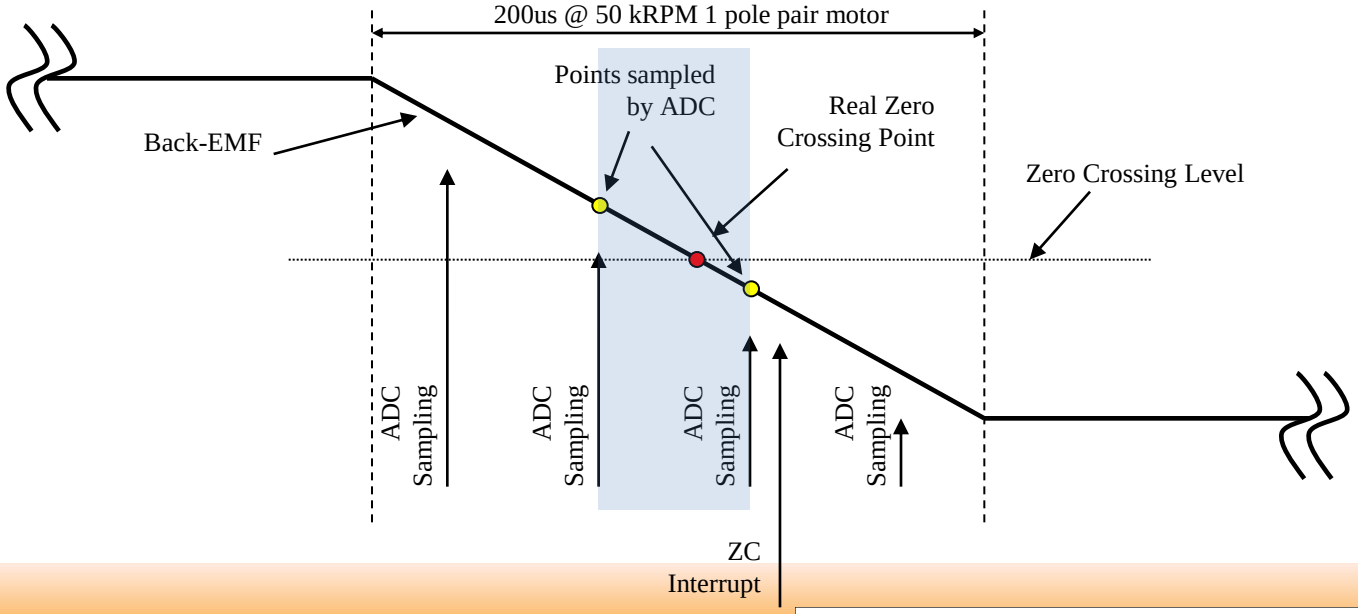
Constant speed 980 ot.min^{-1} at load.

Maximal Speed

- Maximal speed -> shortest commutation period

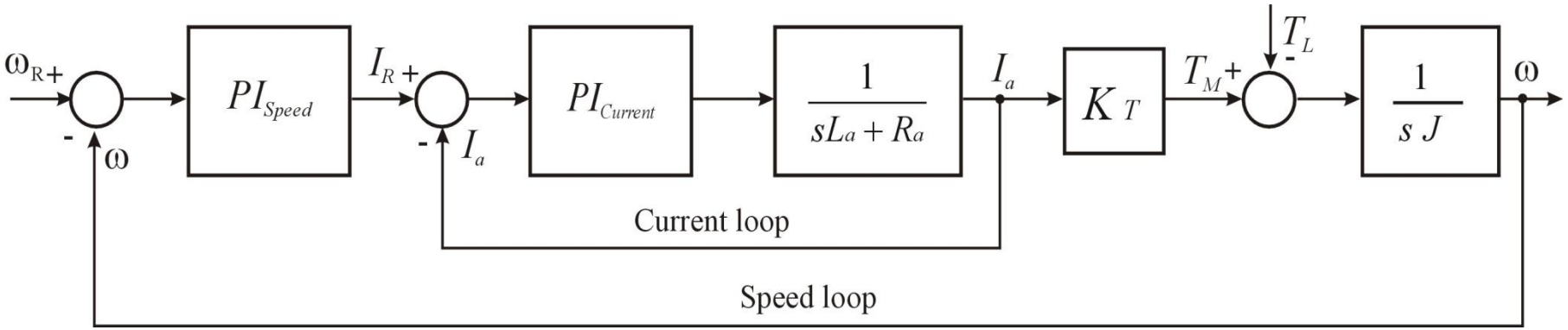
High Speed Implementation

- We detect Zero Crossing instants from Back-EMF signals received by the ADC
- Using ADC, we sample one time in every PWM period. The PWM period is 50us at 20kHz switching frequency.
- On the other hand, at 50 kRPM speed, one commutation period is 200us. (one pole pair motor)
- It means, in one commutation period, we can take only 4 samples.
- In this case, how precise can we know, where the real Zero Cross has happened ?



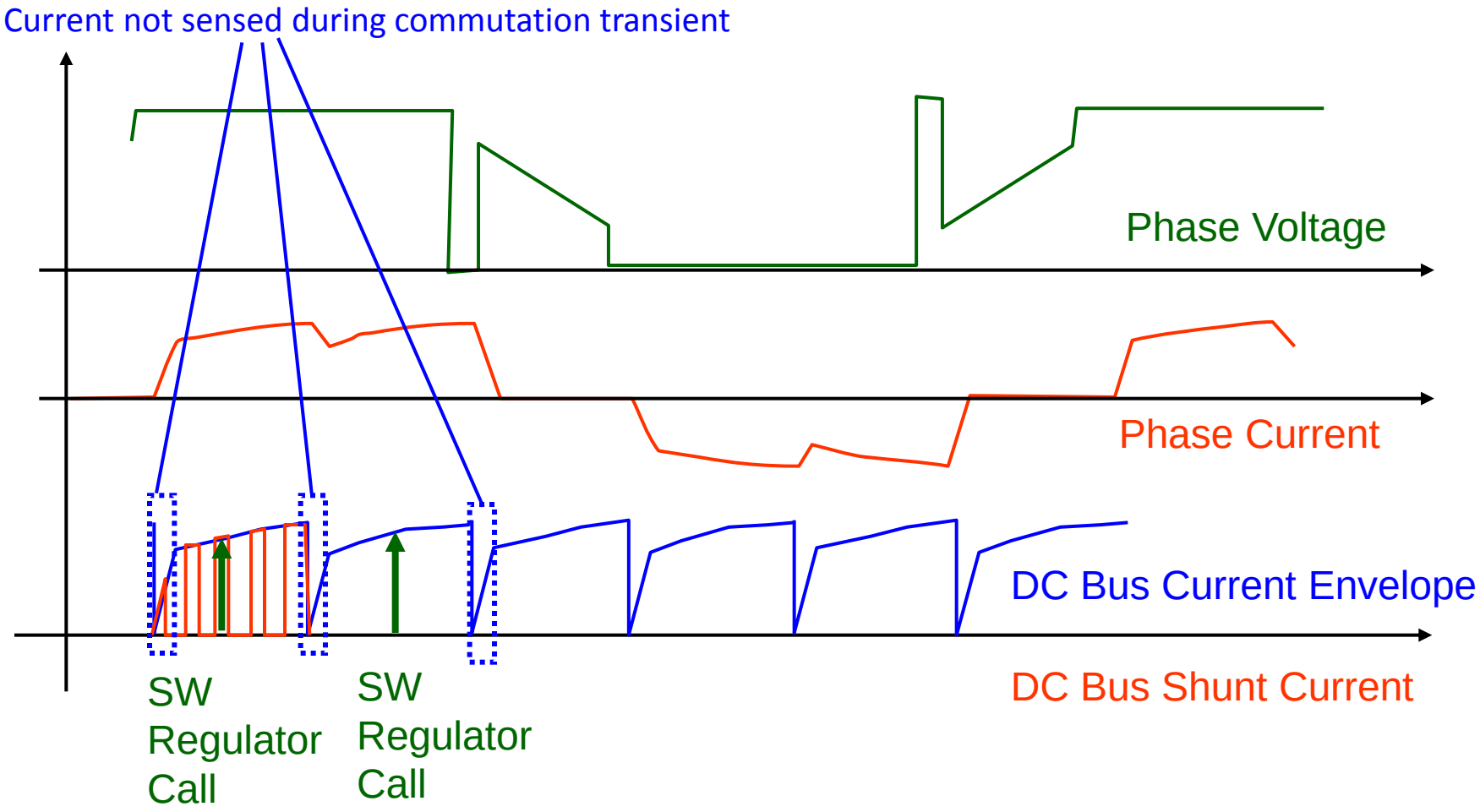
BLDC Motor Speed & Torque/Current Control

- The same structure as Brush DC motor
- Current is not constant over commutation period
- Torque Control is preferred



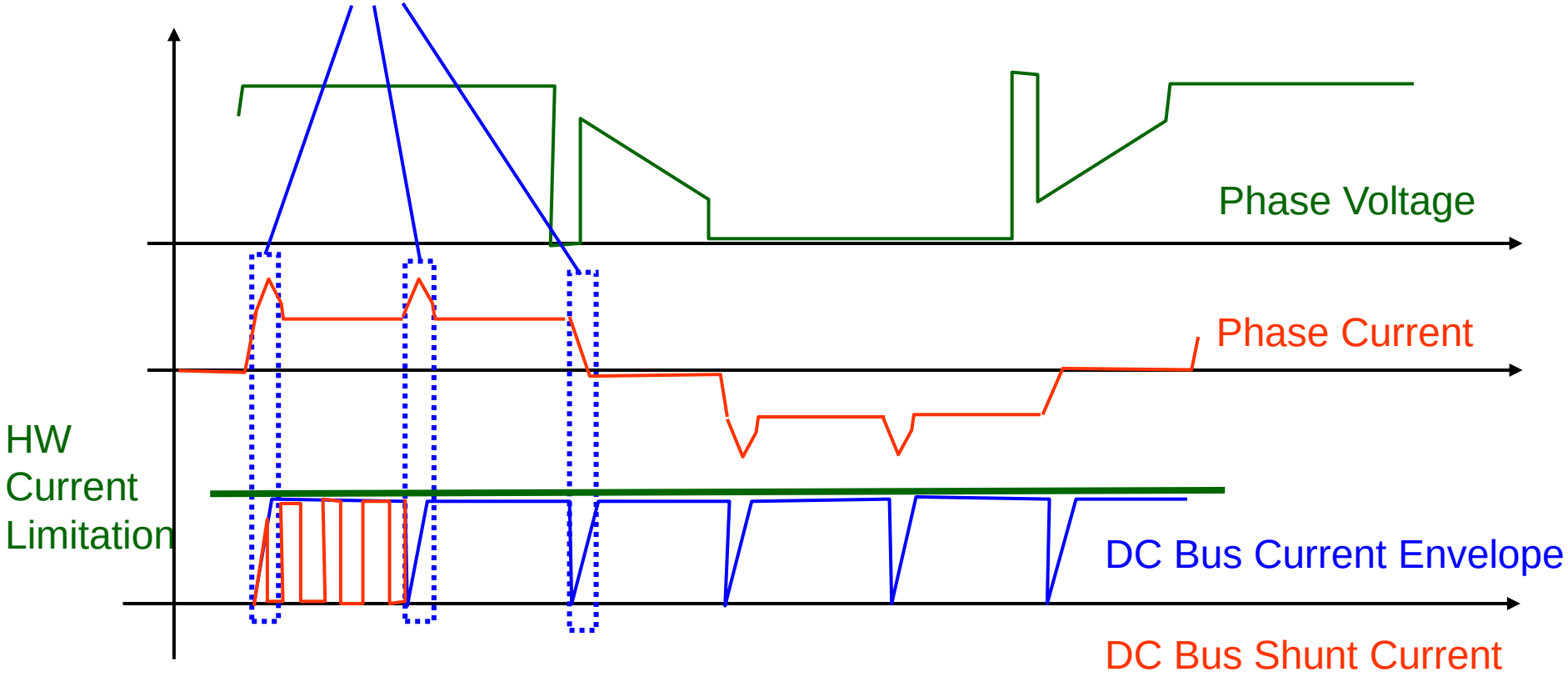
SW Current Control

- Commutation Transient:



HW Current Limitation

- Commutation Transient:
  - Problematic current sensing

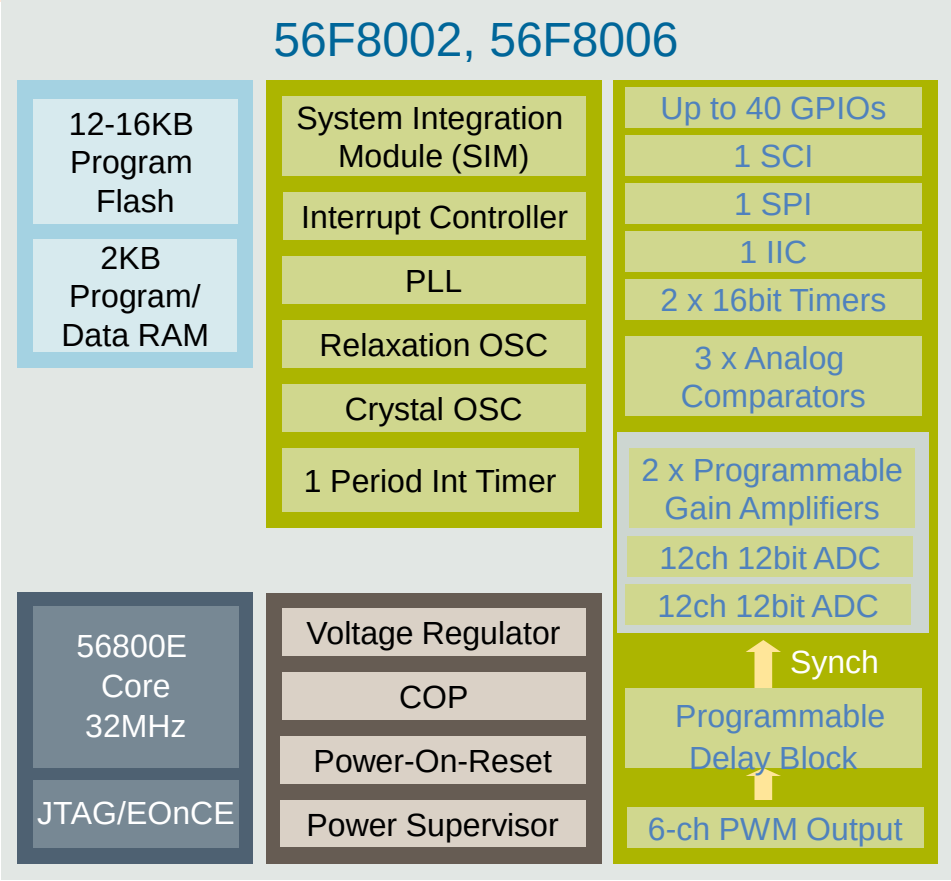


Agenda

- Basic Terms
- BLDC Motor Theory
- Sensorless Technique of BLDC Motors
- Microcontroller MC56F8006/2
  - Microcontroller overview
  - PWM module
  - ADC module
  - ADC to PWM synchronization
- Freescale Software Library

Ultra Low Cost: 56F800x

- 32 MHz/32 MIPS 56800E core
- 1.8-3.6V operation
- 12K - 16K Bytes program FLASH with Flash security
- 2K Bytes program/data RAM
- Tunable internal relaxation oscillator and 32 KHz clock
- Phase locked loop (PLL)
- Up to 96 MHz peripherals – timers, PWM & Hi-SCI
- 6 output PWM module with 4 programmable fault inputs with selectable PWM frequency for each PWM signal complementary pair
- Two programmable gain amplifiers with x2, x4, x8, x16 gains (clocked in order to cancel input offset)
- Two 12-bit ADCs with up to 24 inputs , 2.5us per conversion
- Programmable delay block provides precise control of ADC/PGA sample times relative to PWM reload cycles
- Three high speed analog comparators
- 2 multiple function programmable timers
- Computer operating properly timer
- One periodic interval timer (PIT)
- 1 high speed serial communication interface (Hi-SCI)
- 1 serial peripheral interface (SPI)
- I^2C communications interface
- Up to 40 GPIOs – versatile pin usage
- JTAG/EOnCE™ debug port
- Industrial temperature range: -40C – 105C



Package: 28SOIC, 32SDIP , 32LQFP, 48 LQFP

In Production

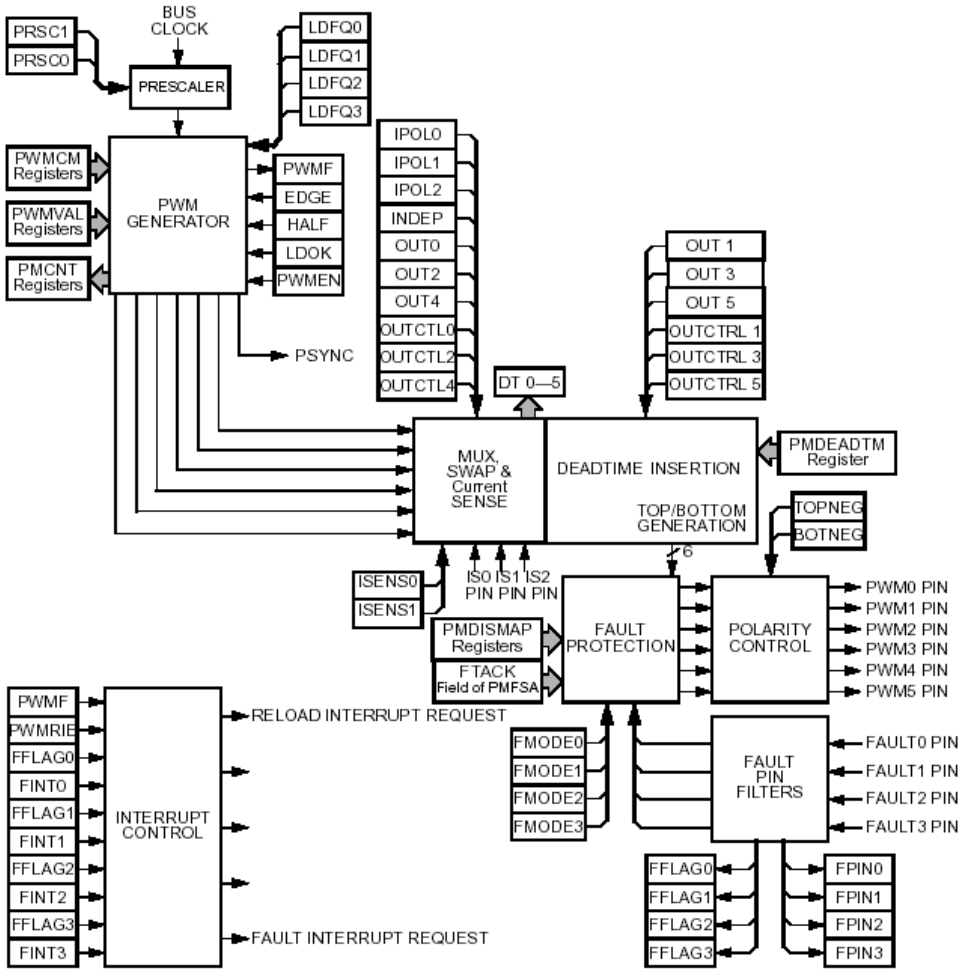
56F8000 Series Feature Summary

	56F8002	56F8006	56F8011	56F8013	56F8014	56F8023	56F8025	56F8036	56F8027/37
Performance	32MHz/MIPs	32MHz/MIPs	32MHz/MIPs	32MHz/MIPs	32MHz/MIPs	32MHz/MIPs	32MHz/MIPs	32MHz/MIPs	32MHz/MIPs
Temperature Range (V)	-40C~105C	-40C~105C	-40C~125C	-40C~125C	-40C~125C	-40C~105C	-40C~105C	-40C~105C	-40C~105C
Voltage Range	1.8V - 3.6V	1.8V - 3.6V	3.0V - 3.6V	3.0V - 3.6V	3.0V - 3.6V	3.0V - 3.6V	3.0V - 3.6V	3.0V - 3.6V	3.0V - 3.6V
Voltage Regulator	On-Chip	On-Chip	On-Chip	On-Chip	On-Chip	On-Chip	On-Chip	On-Chip	On-Chip
Program/Data Flash	12KB	16KB	12KB	16KB	16KB	32KB	32KB	64KB	32KB / 64KB
Program/Data RAM	2KB	2KB	2KB	4KB	4KB	4KB	4KB	8KB	4KB /8KB
Program Security	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
On Chip Relaxation Osc.	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
PLL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
COP (Watchdog)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
PWM (96 Mhz Clock)	1 x 6ch	1 x 6ch	1 x 6ch	1 x 6ch	1 x 5ch	1 x 6ch	1 x 6ch	1 x 6ch	1 x 6ch
PWM Fault Inputs	4	4	4	4	3	4	4	4	4
12-bit ADCs	2 x 8ch	2 x 12ch	2 x 3ch	2 x 3ch	2 x 4ch	2 x 3ch	2 x 4ch	2 x 5ch	2 x 8ch
12-bit DACs	0	0	No	No	No	2	2	2	2 (Pinned out)
Analog Comparator	3	3	No	No	No	2	2	2	2
Prog Gain Amp	2	2	No	No	No	No	No	No	No
16-bit Timers	3	3	4	4	4	4	4	4	8
Prog. Interval Timers	1 (RTC)	1 (RTC)	No	No	No	1	3	3	3
GPIO (max) (+/-8mA)	23	40	26*	26*	26*	26*	35*	39*	53*
IIC	1	1	1	1	1	1 - QIIC	1 - QIIC	1 - QIIC	1 - QIIC
SCI (UART) / LIN Slave	1 - SCI	1 - SCI	1 - SCI	1 - SCI	1 - SCI	1 - QSCI	1 - QSCI	1 - QSCI	1 - QSCI
SPI (Synchronous)	1 - SPI	1 - SPI	1 - SPI	1 - SPI	1 - SPI	1 - QSPI	1 - QSPI	1 - QSPI	1 - QSPI
CAN	No	No	No	No	No	No	No	MSCAN	MSCAN
JTAG/EOnCE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Power Consumption	IDD = 45.6mA;IDDA = 4.5mA		IDD = 42mA; IDDA = 13.5mA			IDD = 48mA; IDDA = 18.8mA		IDD = 48mA; IDDA = 18.8mA	
Package	32LQFP (.8p)	28SOIC 32LQFP 32SDIP 48LQFP	32LQFP	32LQFP	32LQFP	32LQFP	44LQFP	48LQFP	64LQFP

Pulse Width Modulator

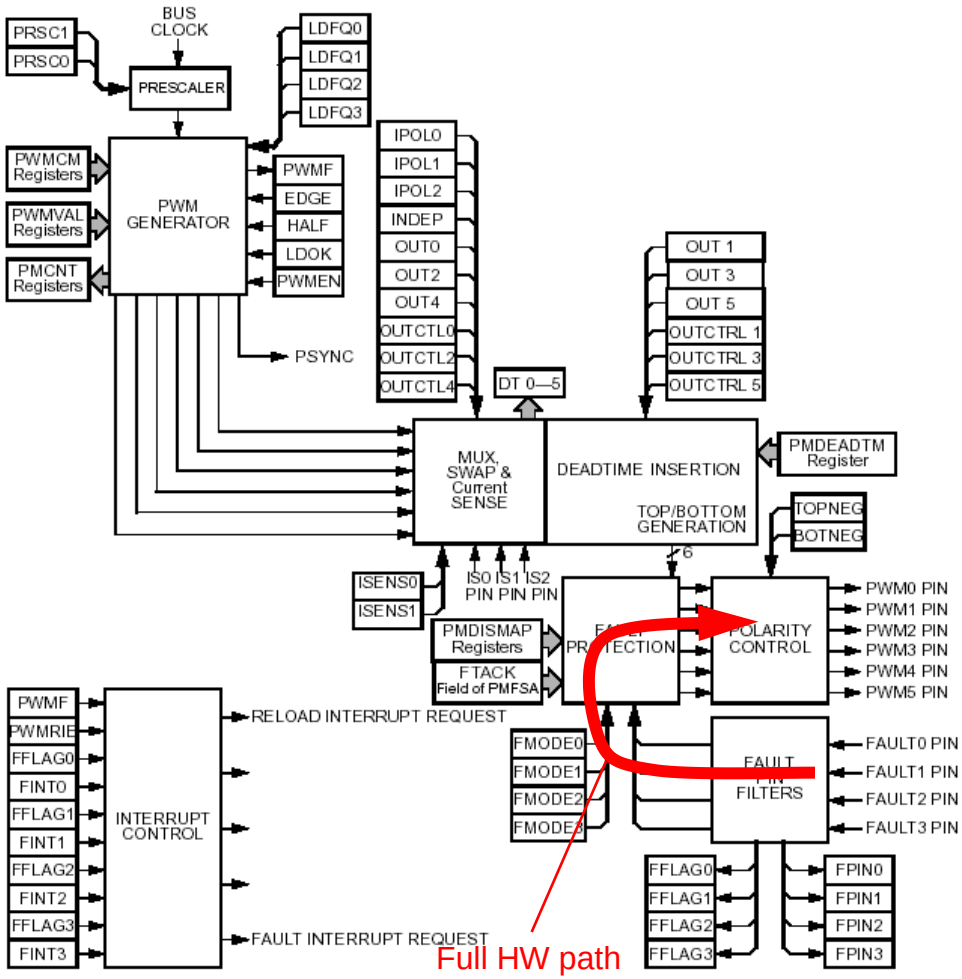
- **Up to 96 MHz operation**
- **Six PWM signals**
  - All independent
  - Complementary pairs
  - Mix independent and complementary
- **Features of complementary channel operation**
  - Independent top and bottom deadtime insertion
  - Separate top and bottom pulse width correction via current status inputs or software
  - Separate top and bottom polarity control
  - Can be controlled from internal PWM generator, software, external digital pins, timers or results of ADC
- **Edge- or Center-Aligned PWM signals**
- **Asymmetric PWM outputs**
- **15-bits of resolution**
- **Half-cycle reload capability**
- **Integral reload rates from 1/2 to 16**
- **Individual software controlled PWM output**
- **Programmable fault protection**
- **Write protected registers**
  - Protection for key parameters

DSC56F80xx Pulse Width Modulator



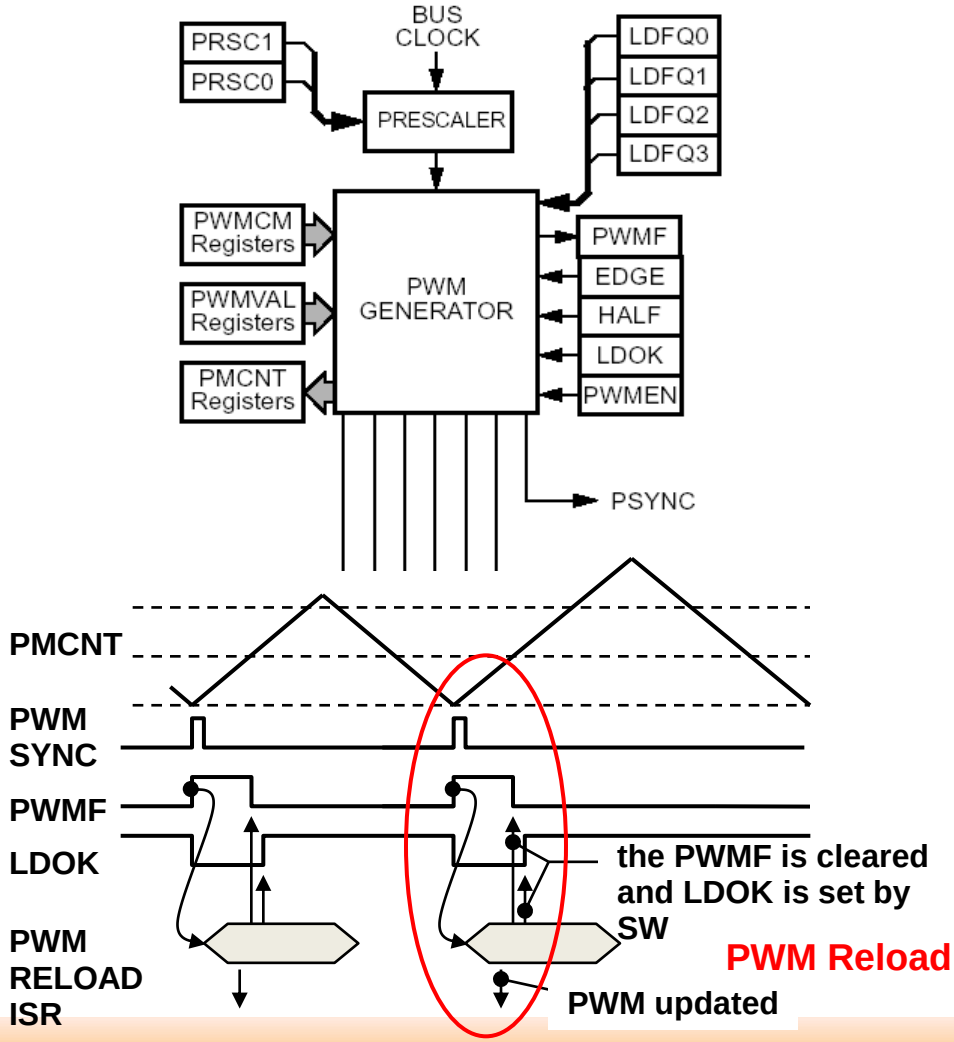
- **Safety** - Write protected registers
- Prescaler
- PWM Generator
- MUX Swap & Current sense
- Deadtime Insertion Top/Bottom Generation
- Software Output Control
- Fault Protection
- Fault Pin Filters
- Polarity Control
- Interrupt Control

PWM - Safety



- **Safety critical setting can be protected** by the write-protect bit (WP).
  Once set **it prevents** any further **writes to write-protected registers or bits**.
  Protection can be cleared only by RESET !
- The list of write-protected registers (their bits) and functions:
  - **PWM polarity**
  - **Complementary PWM pair operation**
  - **Centre aligned PWM channels**
  - **Deadtime value**
  - **PWM fault disable mapping matrix**
  - **HW acceleration features**
  - **PWM generator channels swapping**
  - **Enable PWM in Debug or Wait mode bits**
  - **56F80x Compatibility bit**
- The functions which are still available:
  - PWM Value setting, PWM Frequency setting
  - PWM Clock Prescaller setting
  - PWM Re-load Frequency + Half Cycle reload
  - Deadtime Correction Method setting, odd/even PWMVALx selection for Deadtime Correction
  - Fault Interrupts Enable/Disable, Fault Clearing Mode selection, Fault acknowledging
  - PWM Pins SW Output Control
  - PWM channels masking.
- **At fault the PWM's are forced to INACTIVE state ! It is faster than tri-stating PWM's.**

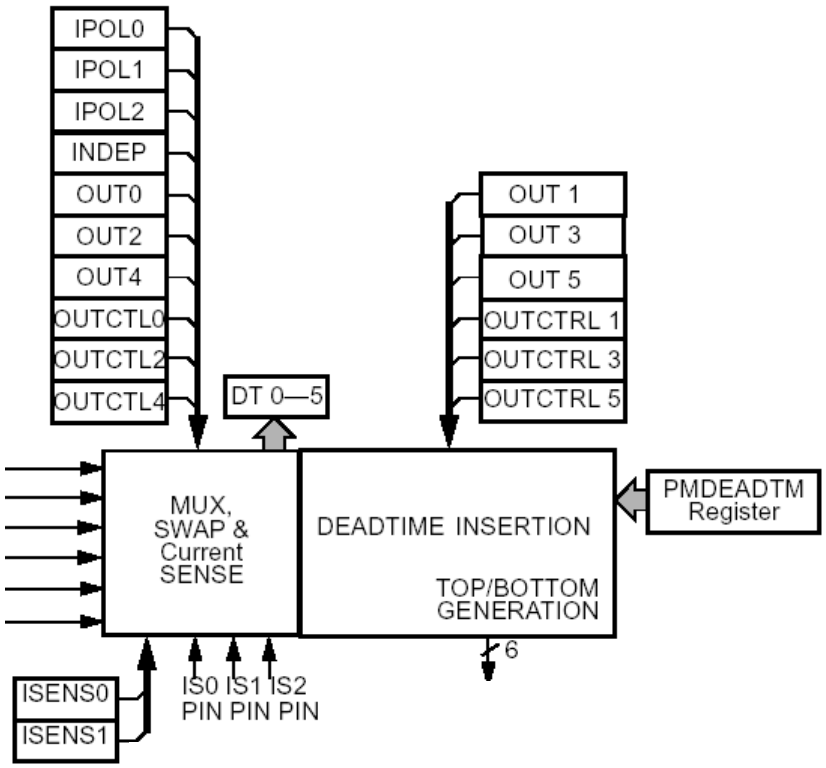
PWM - 6xPWM Generator



- **Prescaler** - 1, 2, 4, 8
- **PWM Generator**
  - **Contains up/down counter counts**
    - 15 bit resolution
    - up to 96 MHz max → 10.417 nsec resolution
  - **Alignment** - edge/centre-aligned
  - **Period** - set by PWM Counter Modulus
  - **Pulse Widths** - defined by PWM Value Registers
    Resolution:
    - 12.2bit @ 20 kHz PWM(edge-aligned)
    - 11.2bit @ 20 kHz PWM(centre-aligned)
  - **Reload Frequency** - half to 16 cycles
  - **Load Enable** interlock bit - prevents reloading of the PWM parameters before software is finished calculating them - **coherent update**
  - **Synchronization output** - high-true pulse occurs for each PWM reload
  - **HW Acceleration** - enables multi-write access of the PWM Value Registers

PWM Reload Interlock Mechanism for Coherent PWM Update

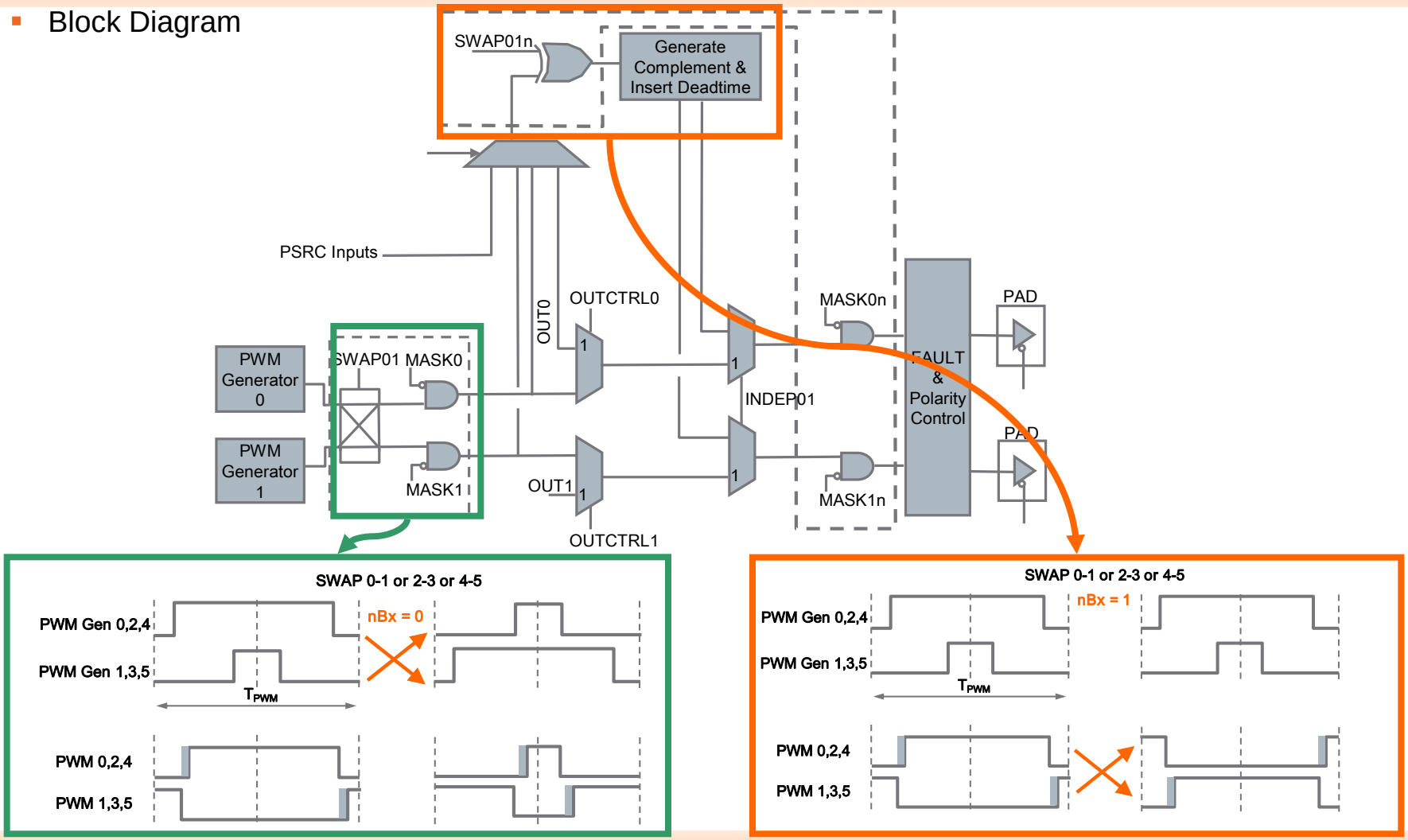
PWM - MUX, SWAP, MASK, Deadtime, SW OUT CTRL



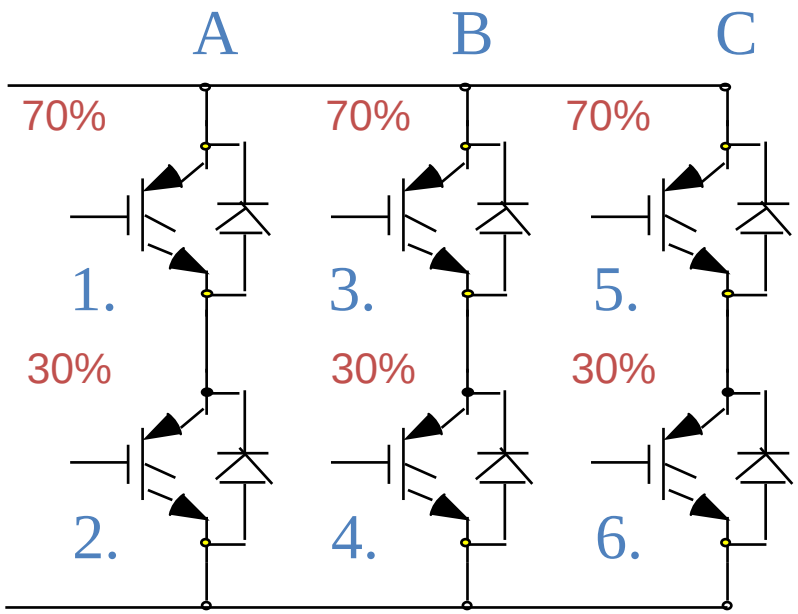
- **MUX Swap & Current sense**
  - **Channel Mask & Swap** - individually swaps and masks PWM generator channels
  - **MUX & Current Sense** - enables SW Output Control and Dead time correction
  - **SW Output Control** - individually controls the PWM outputs with respect to deadtime and complementary operation settings
- **Deadtime Insertion & Top/Bottom Generation**
  - Deadtime generators automatically insert software-selectable “deadtimes” into each pair of PWM outputs
  - The Pulse Module Deadtime register specifies the number of PWM clock cycles to use for deadtime delay
  - Every time the deadtime generator inputs changes state, deadtime is inserted
- **Software Output Control**
  - In an independent mode the output bit OUTx controls the PWMx channel.
  - In a complementary channel operation the OUT0/2/4 bits control the top/bottom pair.

PWM – Mux and Swap Options

- Block Diagram



BLDC Motor Commutation – MC56F800x, MC56F80xx



PWM Channel Control Register

MASK												SWAP		
		0	0	0	0	0	0					0	0	0

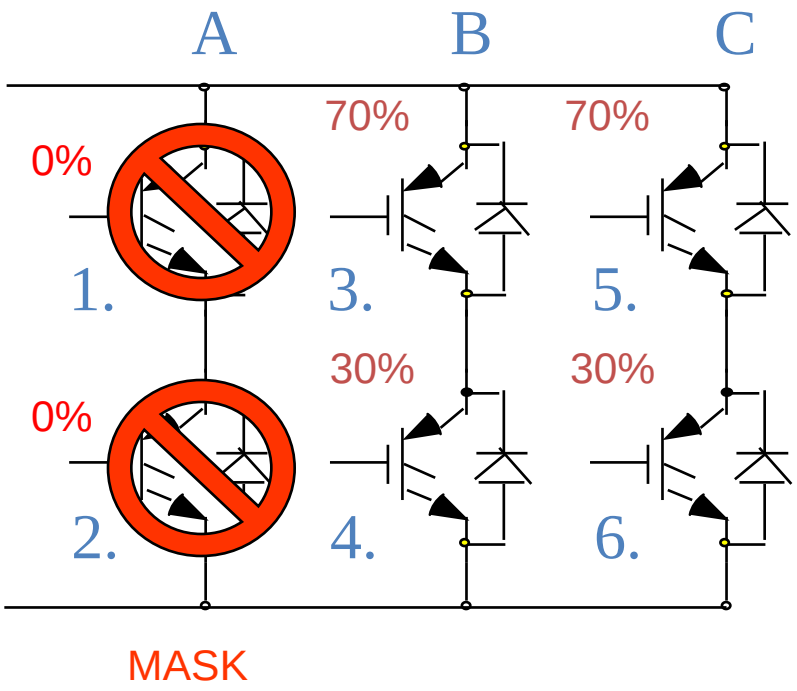
One “shot” write to all six PWM channels!
Value written to this register defines just the speed of motor rotation and

PWM Value Register

0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1

5999h ≈ 70% of max. value of Duty Cycle when 7FFFh is the max. value

BLDC Motor Commutation – MC56F800x, MC56F80xx



PWM Channel Control Register

MASK										SWAP			
		0	0	0	0	1	1					0	0	0

6. 5. 4. 3. 2. 1. transistors

Mask will disable the complementary transistor pair

PWM Value Register

0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1

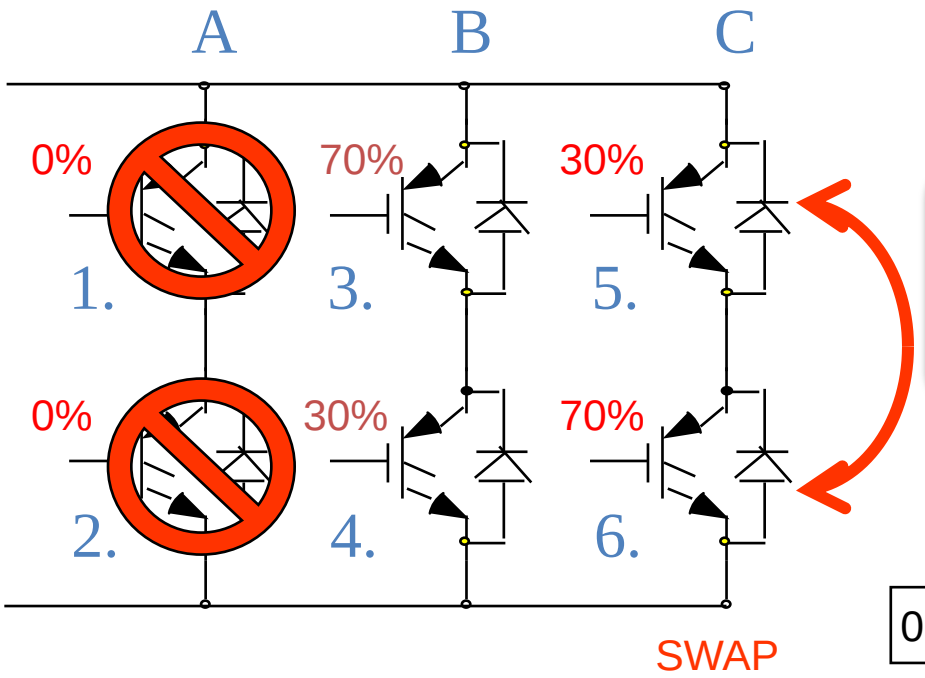
5999h ≈ 70% of max. value of Duty Cycle when 7FFFh is the max. value

BLDC Motor Commutation – MC56F800x, MC56F80xx

Mask and Swap are set simultaneously!
Value written to this register defines the exact commutation stage.

PWM Channel Control Register

MASK										SWAP		
		0	0	0	0	1	1				1	0	0
		6.	5.	4.	3.	2.	1.				C	B	A
		transistors									phases		



Swap reroutes the top and bottom control signals of complementary pair

PWM Value Register

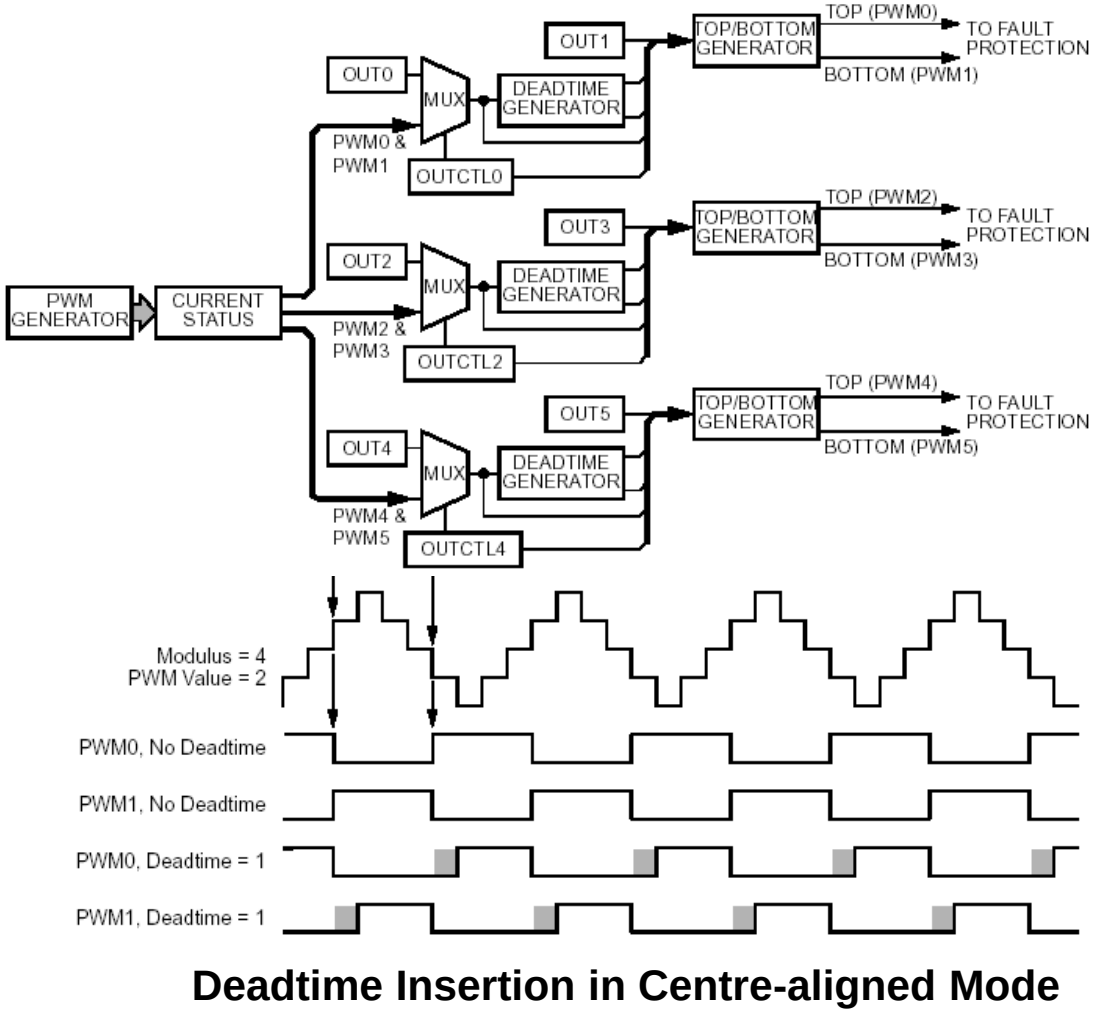
0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1

5999h ≈ 70% of max. value of Duty Cycle when 7FFFh is the max. value

BLDC Motor Commutation – MC56F800x, MC56F80xx

- Advantage:
  - The MASK and SWAP feature is asynchronous to PWM generation
  - Speed control and commutation control are fully independent
- The same approach can be used on 56F82xx using FORCE_OUT logic
- The 56F82xx allows preloading of new state and time for next commutation. The commutation is performed based on timer event

PWM - Deadtime Generators

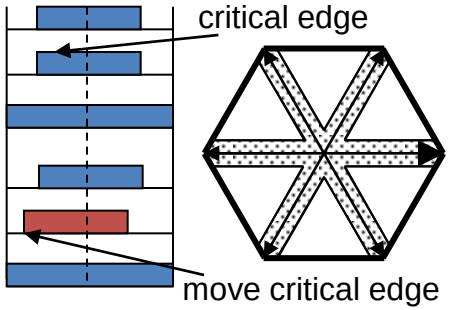


Deadtime Insertion in Centre-aligned Mode

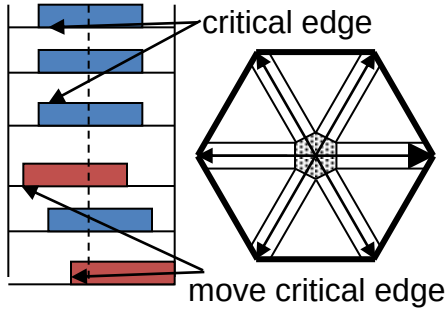
- **Deadtime Generators**
  - Deadtime generators automatically insert software-selectable “deadtimes” into each pair of PWM outputs
  - Every time the deadtime generator inputs changes state, deadtime is inserted
  - In Software Output Control
    - Deadtime generators continue to insert deadtime whenever an OUT0/2/4 bit toggles.
    - Deadtime is not inserted when the OUT1/3/5 bit toggles.
  - The Pulse Module Deadtime register specifies the number of PWM clock cycles to use for deadtime delay

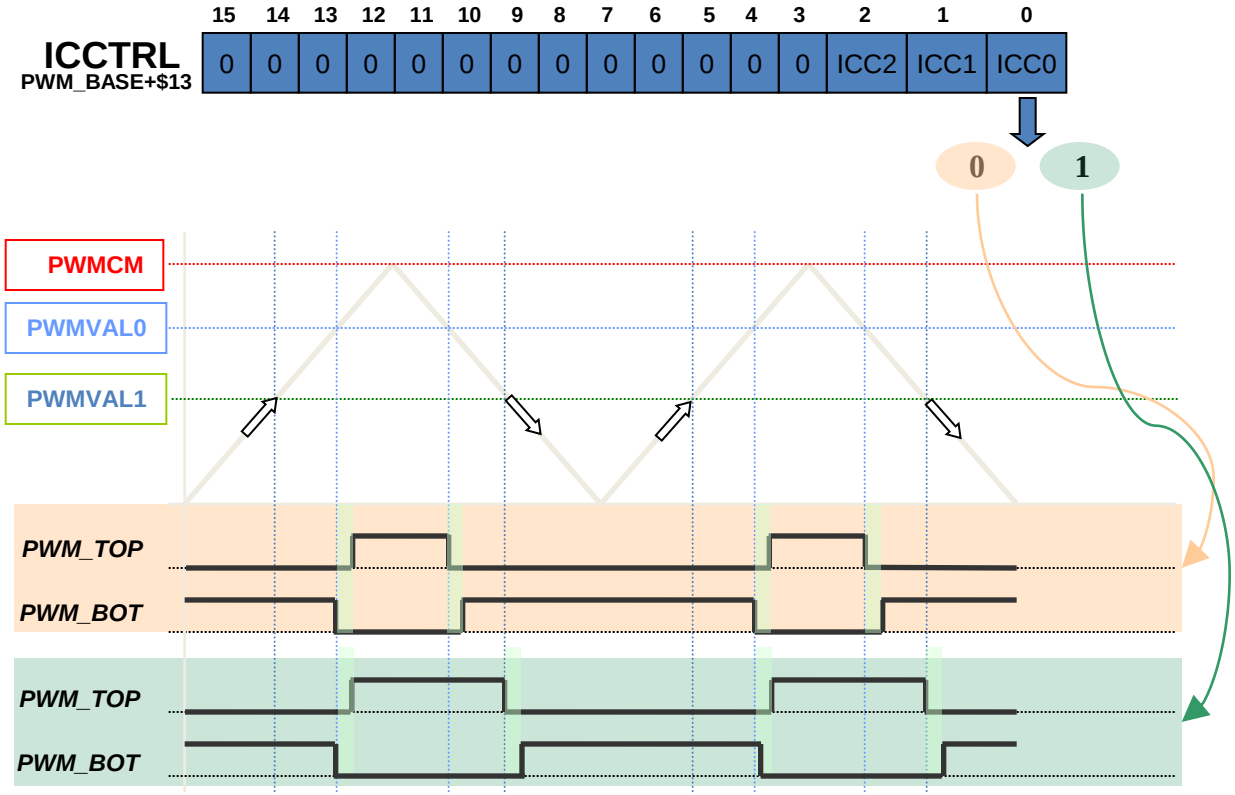
PWM – Asymmetrical Generation

- Passing active vector



- Low modulation index



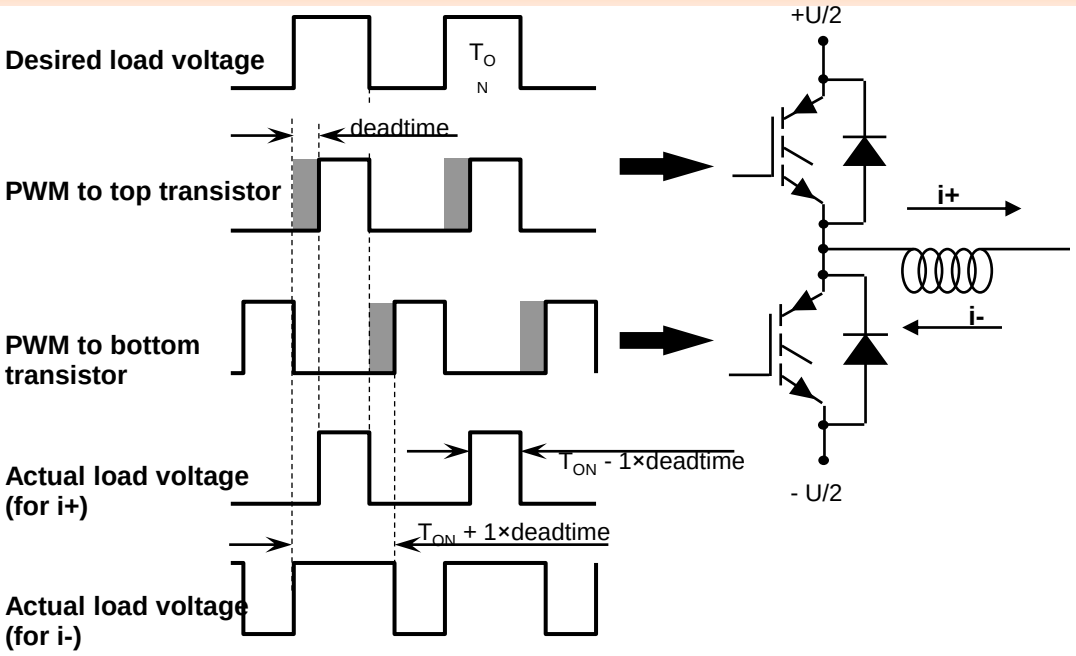


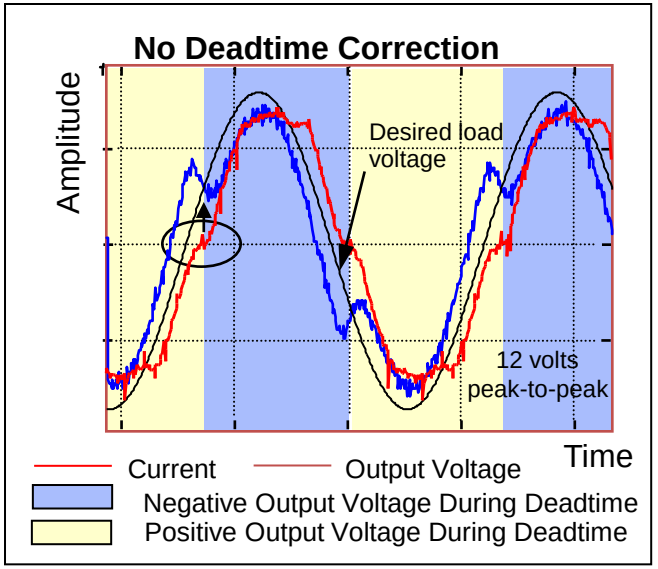
PWM - SW Output Control - Details

OUTx bit	Complementary channel operation	Independent channel operation
OUT0	1—PWM0 is active + deadtime insertion 0—PWM0 is inactive	1—PWM0 is active 0—PWM0 is inactive
OUT1	1—PWM1 is complement of PWM 0 0—PWM1 is inactive	1—PWM1 is active 0—PWM1 is inactive
OUT2	1—PWM2 is active + deadtime insertion 0—PWM2 is inactive	1—PWM2 is active 0—PWM2 is inactive
OUT3	1—PWM3 is complement of PWM 2 0—PWM3 is inactive	1—PWM3 is active 0—PWM3 is inactive
OUT4	1—PWM4 is active + deadtime insertion 0—PWM4 is inactive	1—PWM4 is active 0—PWM4 is inactive
OUT5	1—PWM5is complement of PWM 4 0—PWM5 is inactive	1—PWM5 is active 0—PWM5 is inactive

- ✓ **Complementary channel pairs still cannot be active simultaneously !**
  The OUT0/2/4 replace the PWM generator outputs as inputs to the deadtime generators.
  Deadtime generators continue to insert deadtime whenever an OUT0/2/4 bit toggles.
  Deadtime is not inserted when the OUT1/3/5 bit toggles.
- ✓ **Setting the OUTCTLx bits does not disable the PWM generators and current status sensing circuitry!**
  They continue to run, but no longer control the output pins.
  When the OUTCTLx bits are cleared, the outputs of the PWM generator become the inputs to the deadtime generators at the **beginning of the next PWM cycle**.
- ✓ **Software can drive the PWM outputs even when PWM enable bit (PWMEN) is set to zero.**
- ✓ **During software output control, TOPNEG and BOTNEG still control output polarity !**

PWM - Distortion Caused by Inductive Loads



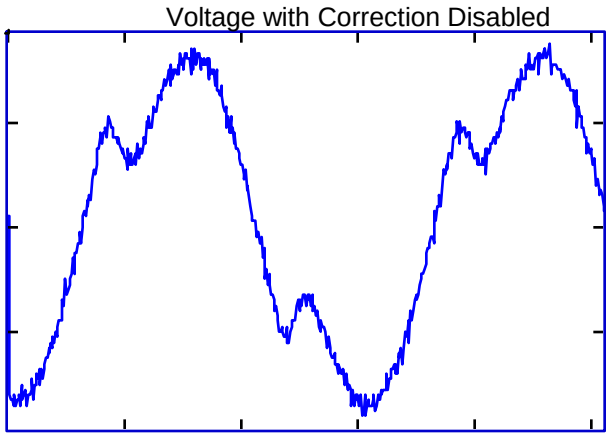


- ✓ Both transistors are "off" during deadtime. **Load inductance** keeps current flowing through the diodes and thus **defines an output voltage**.
- ✓ **Positive current flow** causes negative output voltage during deadtime – **top transistor controls output voltage.**
- ✓ **Negative current flow** causes positive output voltage during deadtime – **bottom transistor controls output voltage.**

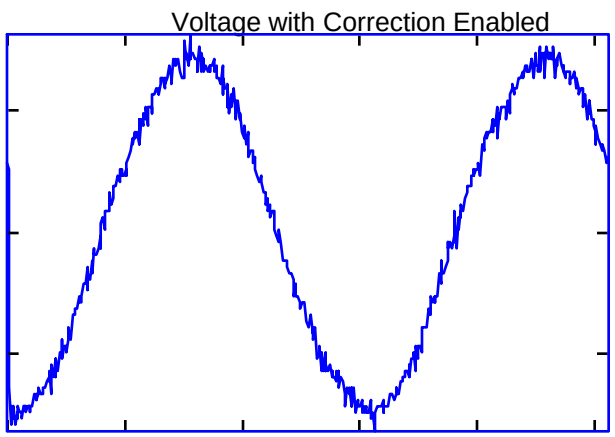
- ✓ **Causes poor low-speed motor performance**
- ✓ **Torque ripple**
- ✓ **Distorts current**
- ✓ **Produces noise in audible region**

PWM - Patented PWM Distortion Correction

Actual waveforms taken on a 1/2 horsepower motor



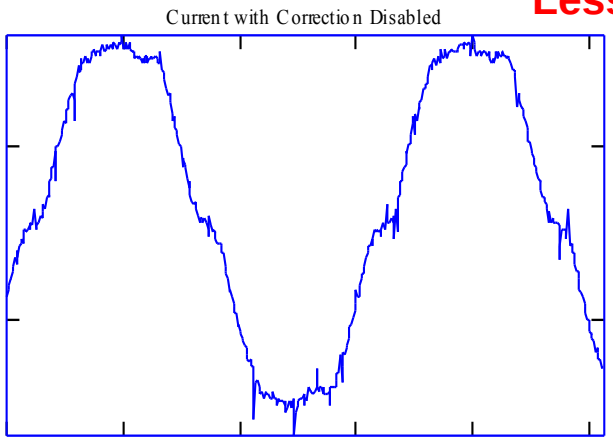
Motor Voltage



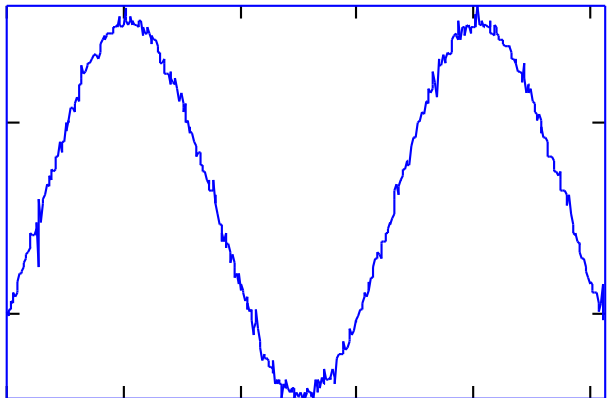
Before

After

Quieter operation
Smoother operation
Less motor harmonic losses



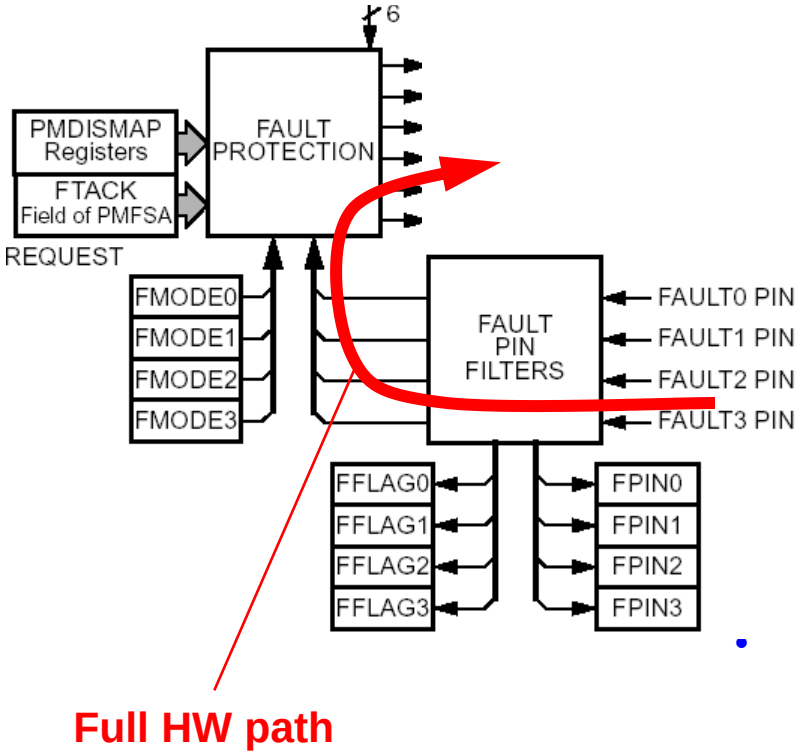
Motor Current



1/2 horse 3 phase motor
PWM Frequency = 7.3 KHz
Dead Time = 3 uS
Output ω = 1.7 Hz.

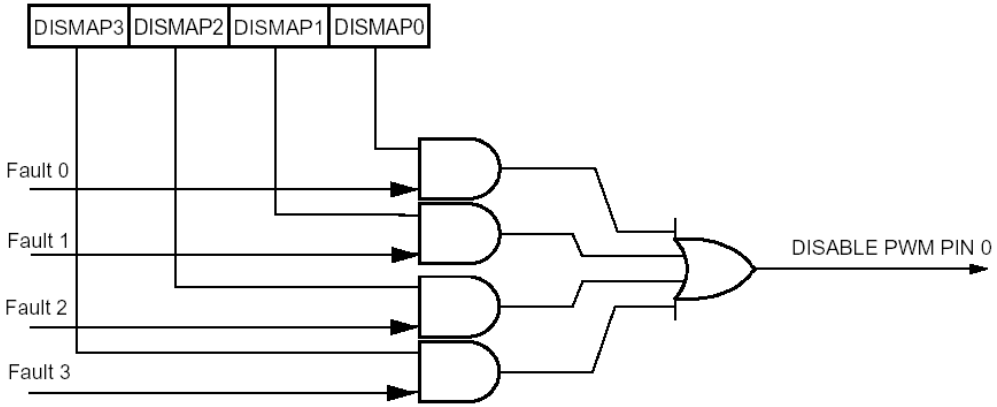
More details in dedicated presentation

PWM - Fault Protection



- **Fault Protection**
  - **Fault protection can automatically disable any combination of PWM pins !**
  - Faults are generated by a logic 1 on any of the FAULT pins.
  - **When fault occurs, only the output pins are deactivated - the PWM generator continues to run !**
  - **The fault protection is enabled even when the PWM is not enabled**. Service faults before PWM enable.
  - **Automatic Fault Clearing -** the disabled PWM pins are enabled when the FAULTx pin returns to logic 0 and a new PWM half cycle begins.
  - **Manual Fault Clearing**
    - FAULT0/2 - the disabled PWM pins are enabled when software clears the FFLAGx flag + next PWM half cycle begins regardless of the logic level detected by the filter at the fault pin.
    - FAULT1/3 - the disabled PWM pins are enabled when software clears the FFLAGx flag + the filter detects a logic zero on the fault pin at the start of the next PWM half cycle.
- **Fault Pin Filters**
  - After every IPBus cycle setting the FAULTx pin at logic 0, the filter synchronously samples the pin once in each of the next two cycles. If both samples are logic 1s, the corresponding fault bits (FAULTx, FPINx, FAULTx, FFLAGx) are set.
  - The FPINx bit remains set until the pin returns to logic 0 and the filter samples a logic 0 synchronously once in the following IPbus cycle.

PWM - Fault Pins Mapping



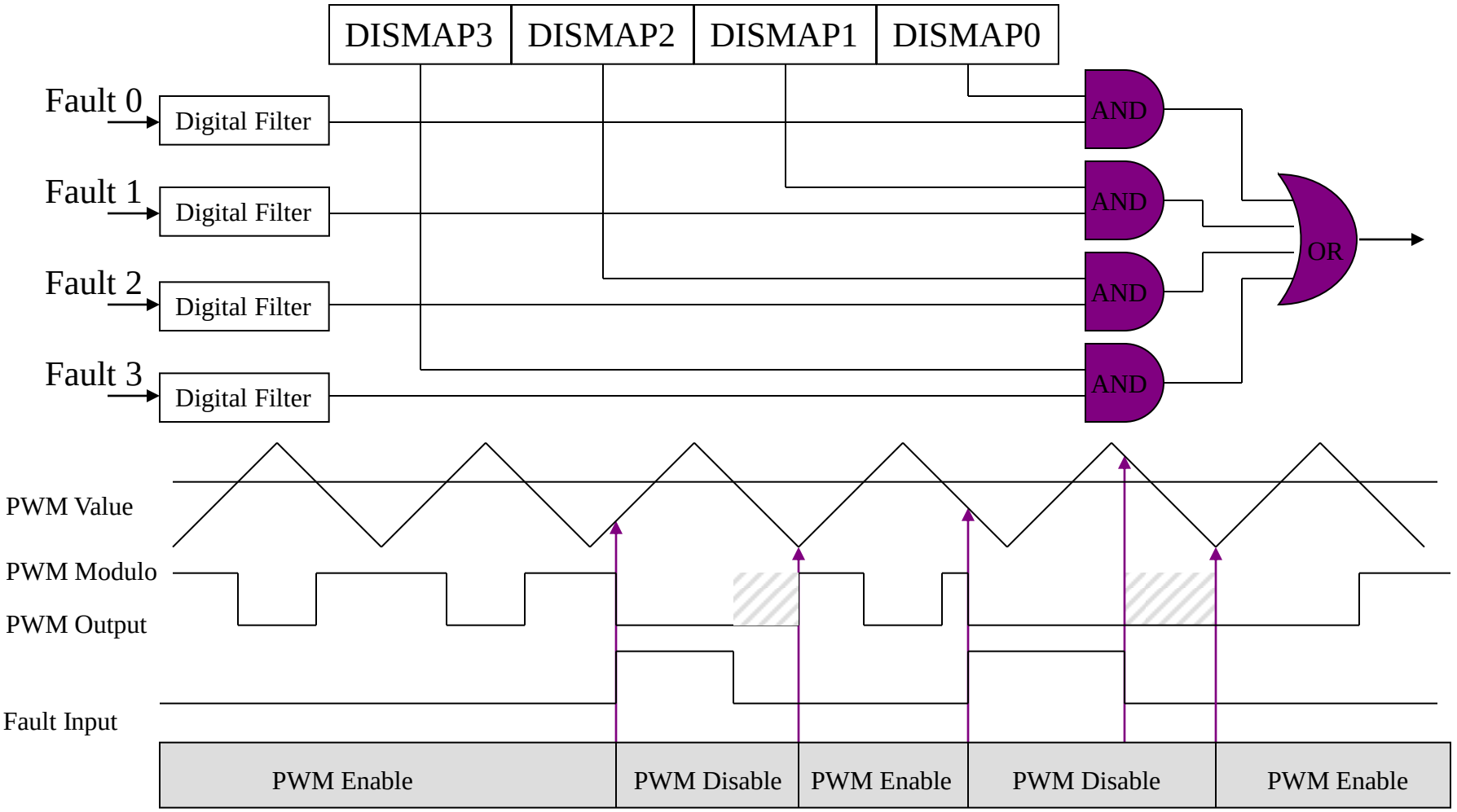
- ✓ **FAULT Pins Mapping**
  - ✓ The fault decoder disables PWM pins pre-selected by the disable mapping register.
    Each FAULT pin can be mapped arbitrarily to any of the PWM pins.
  - ✓ Each bank of four bits in the disable mapping register control the mapping for a single PWM pin.

PWM Pin to be disabled	Fault0	Fault1	Fault2	Fault3	PWM Disable Mapping Register
	to assign Fault and PWM pin set **DISMAP bit #**				
PWM0	0	1	2	3	PMDISMAP1 [0:3]
PWM1	4	5	6	7	PMDISMAP1 [4:7]
PWM2	8	9	10	11	PMDISMAP1 [8:11]
PWM3	12	13	14	15	PMDISMAP1 [12:15]
PWM4	16	17	18	19	PMDISMAP2 [0:3]
PWM5	20	21	22	23	PMDISMAP2 [4:7]

Example

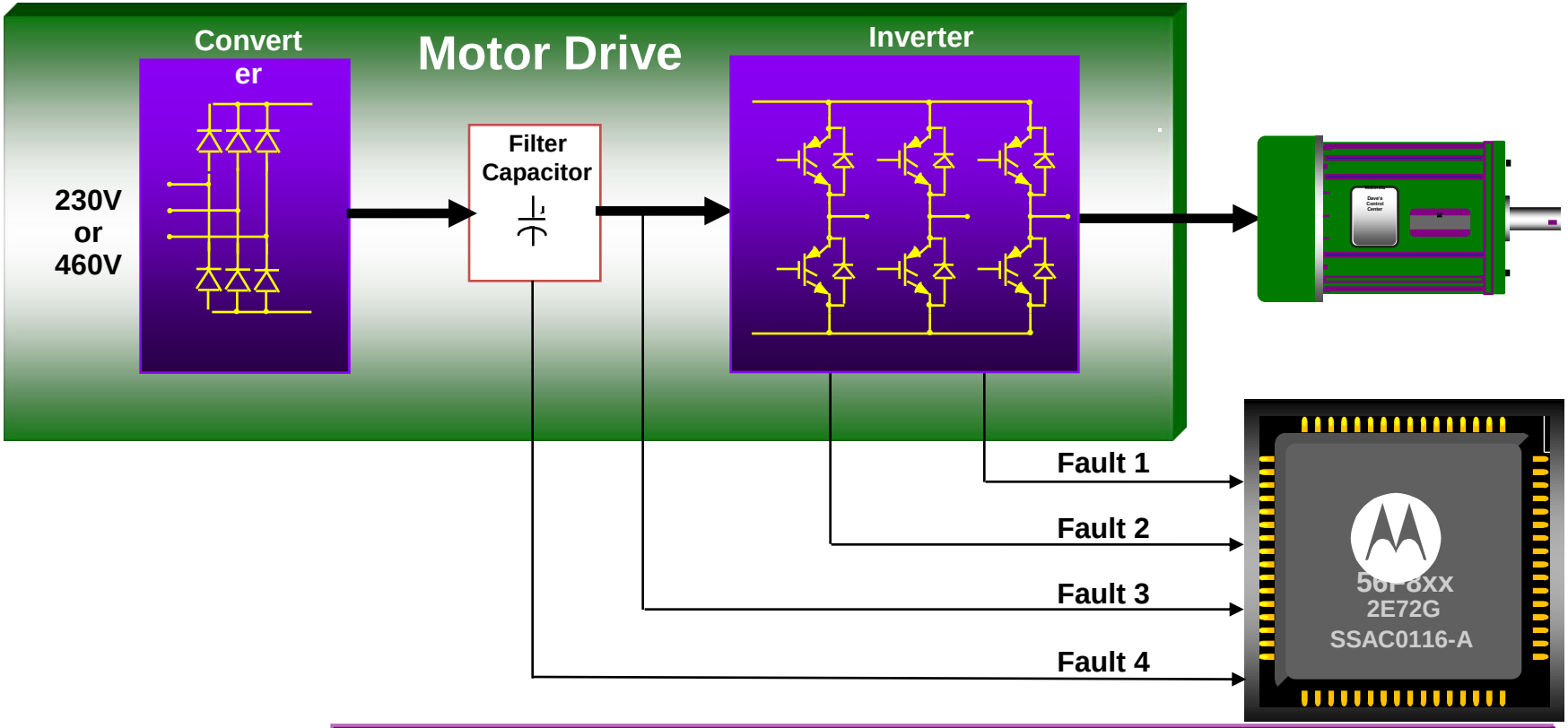
Fault Pins vs. PWM Pins Mapping

PWM Fault Decode And Automatic Clearing



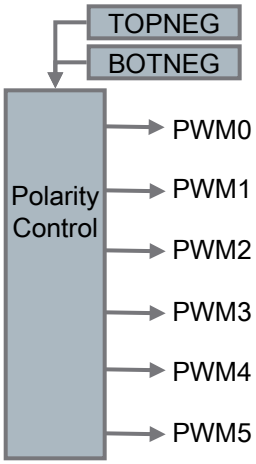
*When Fault logic returns to logic 0, the PWM restart at beginning of the next half cycle.

Multiple Fault Inputs

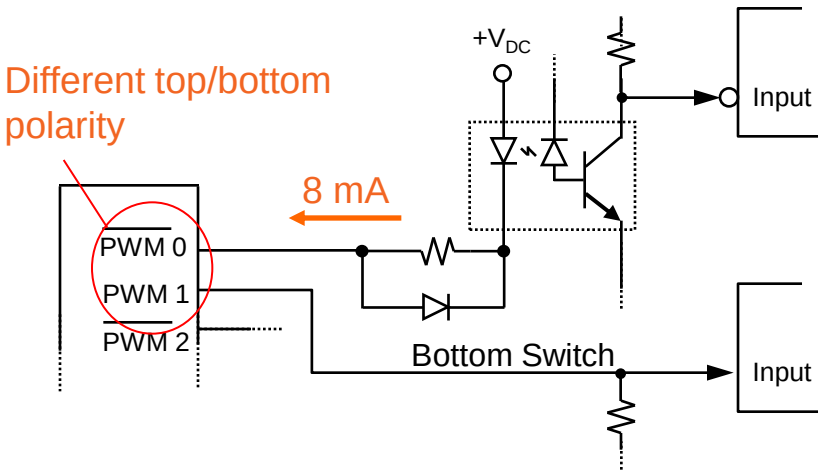


- ✓ Fault inputs can independently monitor critical system parameters, and generate an interrupt when asserted.
- ✓ Each input is mapable to immediately disable any or all PWMs
- ✓ Each input is programmable to allow Automatic or Manual PWM restart.

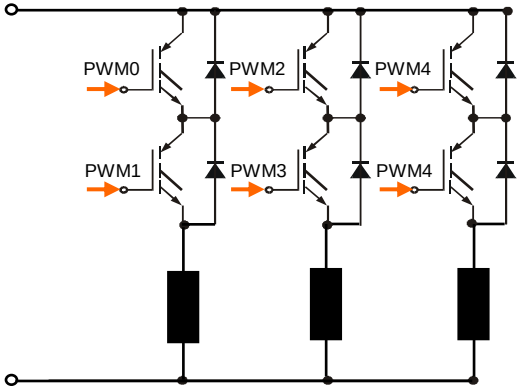
Pulse Width Modulator - Polarity Control



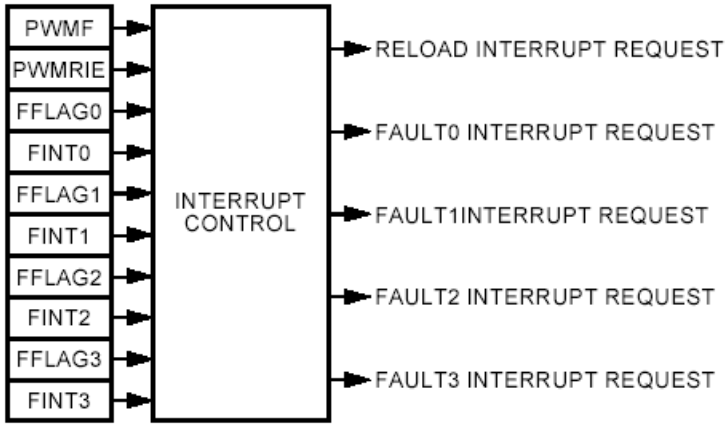
- **Polarity Control**
  - ***Positive polarity** means when the PWM is active - its output is high.*
  - ***Negative polarity** means when the PWM is active - its output is low.*
  - ***Separate control of top and bottom PWM outputs.***
    ***TOPNEG** - controls PWM0/2/4 polarity.*
    ***BOTNEG** - controls PWM1/3/5 polarity.*
- **High Current Capability**
- **8 mA current sink / current source capability**



Direct PWM pin-optocoupler connection



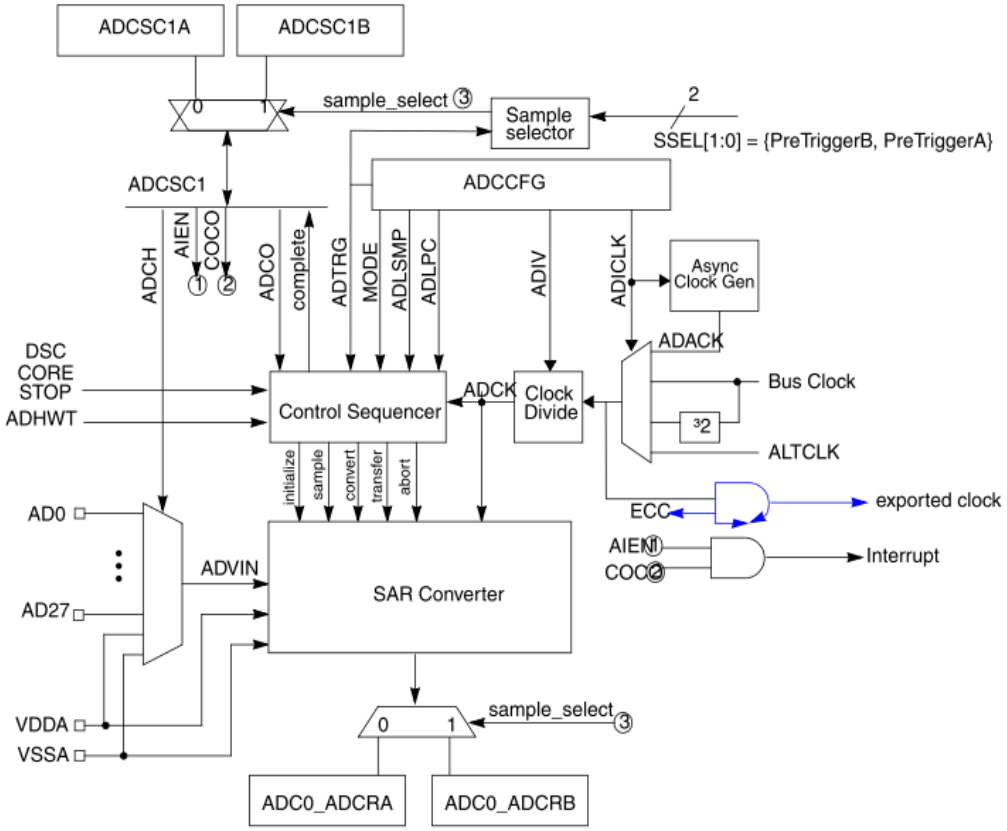
PWM - Interrupt Control



- **The PWM module can generate up to 5 interrupt requests.**
- **Reload flag (PWMF)** - PWMF is set at the beginning of every reload cycle.
  The reload interrupt enable bit (PWMRIE) enables PWMF to generate CPU interrupt requests.
- **Fault flags (FFLAG0–FFLAG3)** - The FFLAGx bit is set when fault pin filters recognises a logic 1 on the FAULTx pin.
  The fault pin interrupt enable bits (FIE0–FIE3) enable the FFLAGx flags to generate CPU interrupt requests.

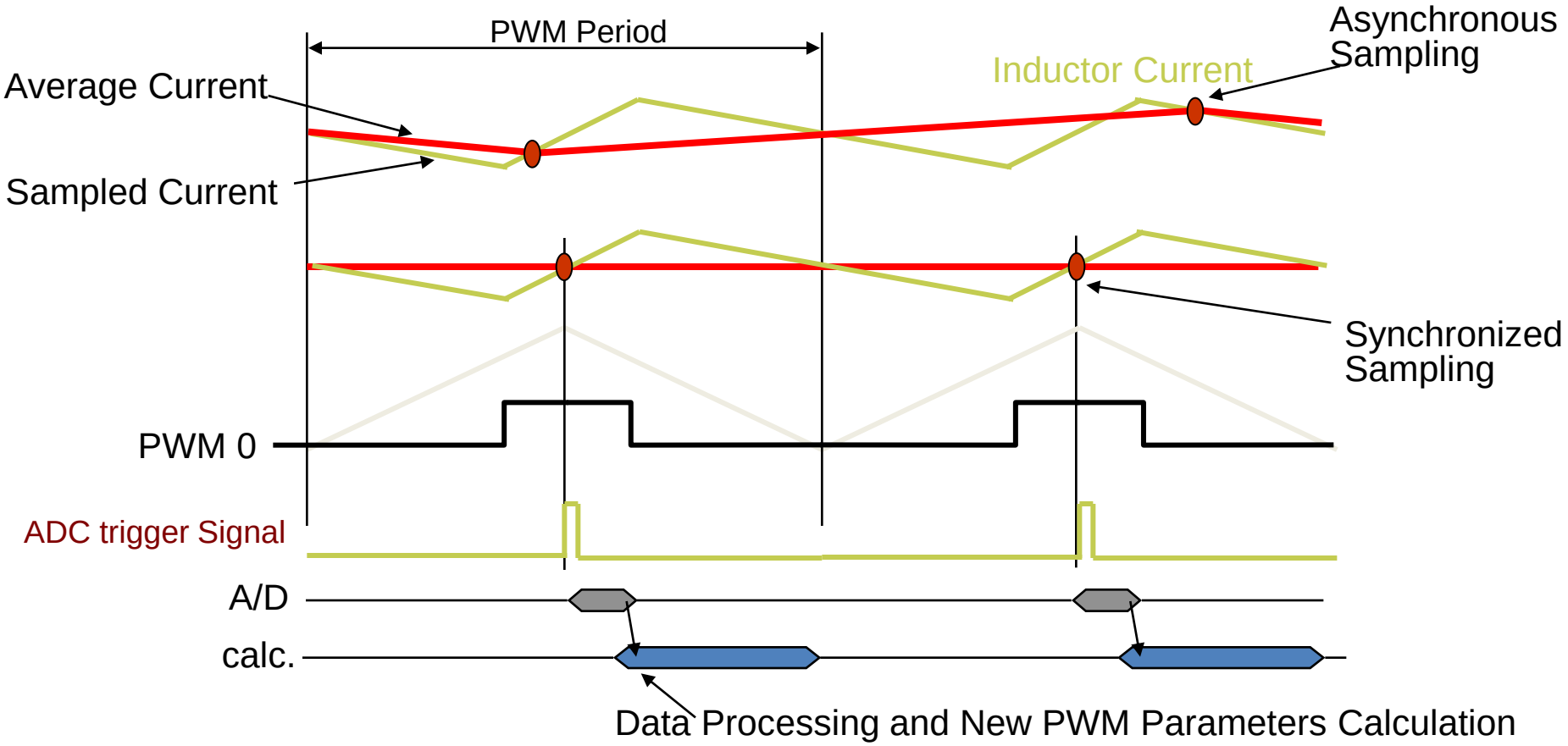
A/D Converter

- Two ADC modules allowing parallel conversion
- Input voltage range from VSSA to VDDA
- Up to 28 analog inputs
- Output in 12-, 10- or 8-bit right-justified format
- Single or continuous conversion (automatic return to idle after single conversion)
- 2.5 μs conversion time
- Configurable sample time and conversion speed/power
- Conversion complete flag and interrupt
- Input clock selectable from up to four sources
- Operation in wait or stop modes for lower noise operation
- Asynchronous clock source for lower noise operation
- Hardware and software triggering
- Temperature sensors that are routed to ANA26 and ANB26
- Support up to four samples per conversion



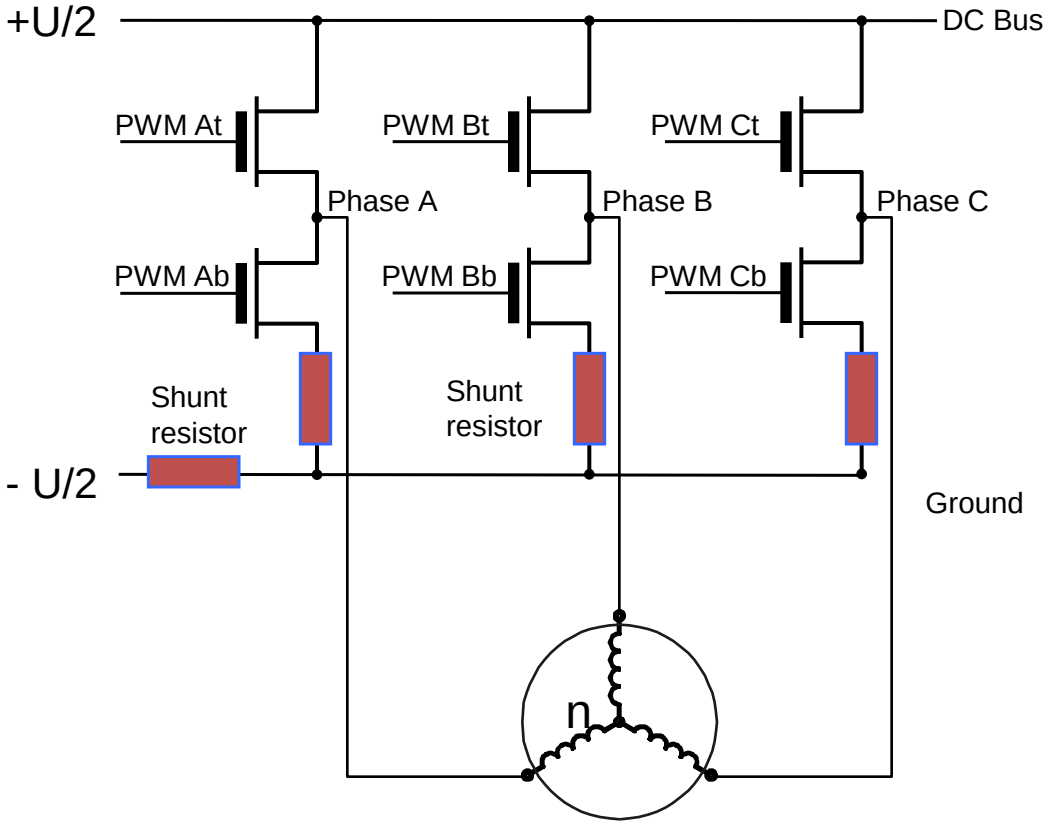
Why Is ADC to PWM Synchronization Needed?

- ADC sampling helps to filter the measured current - antialiasing

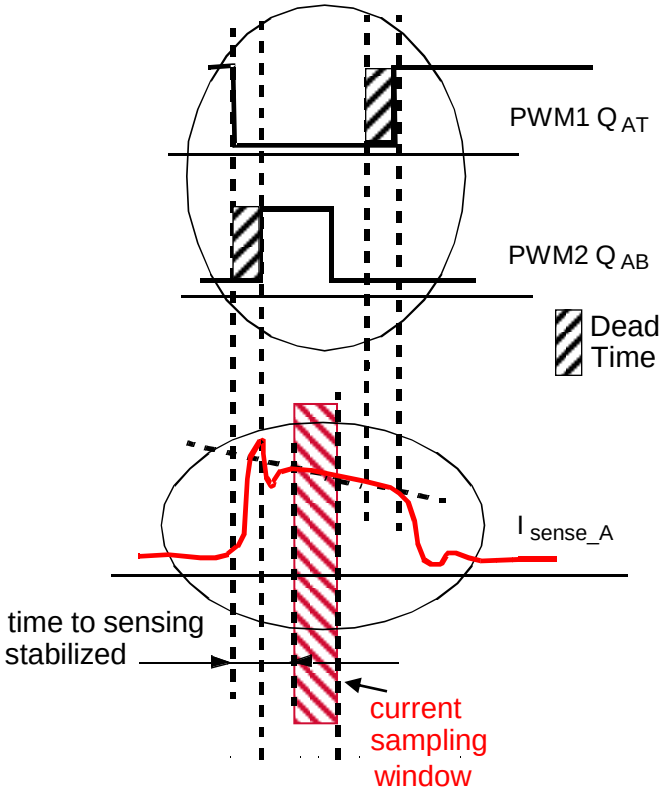


Why Is ADC to PWM Synchronization Needed?

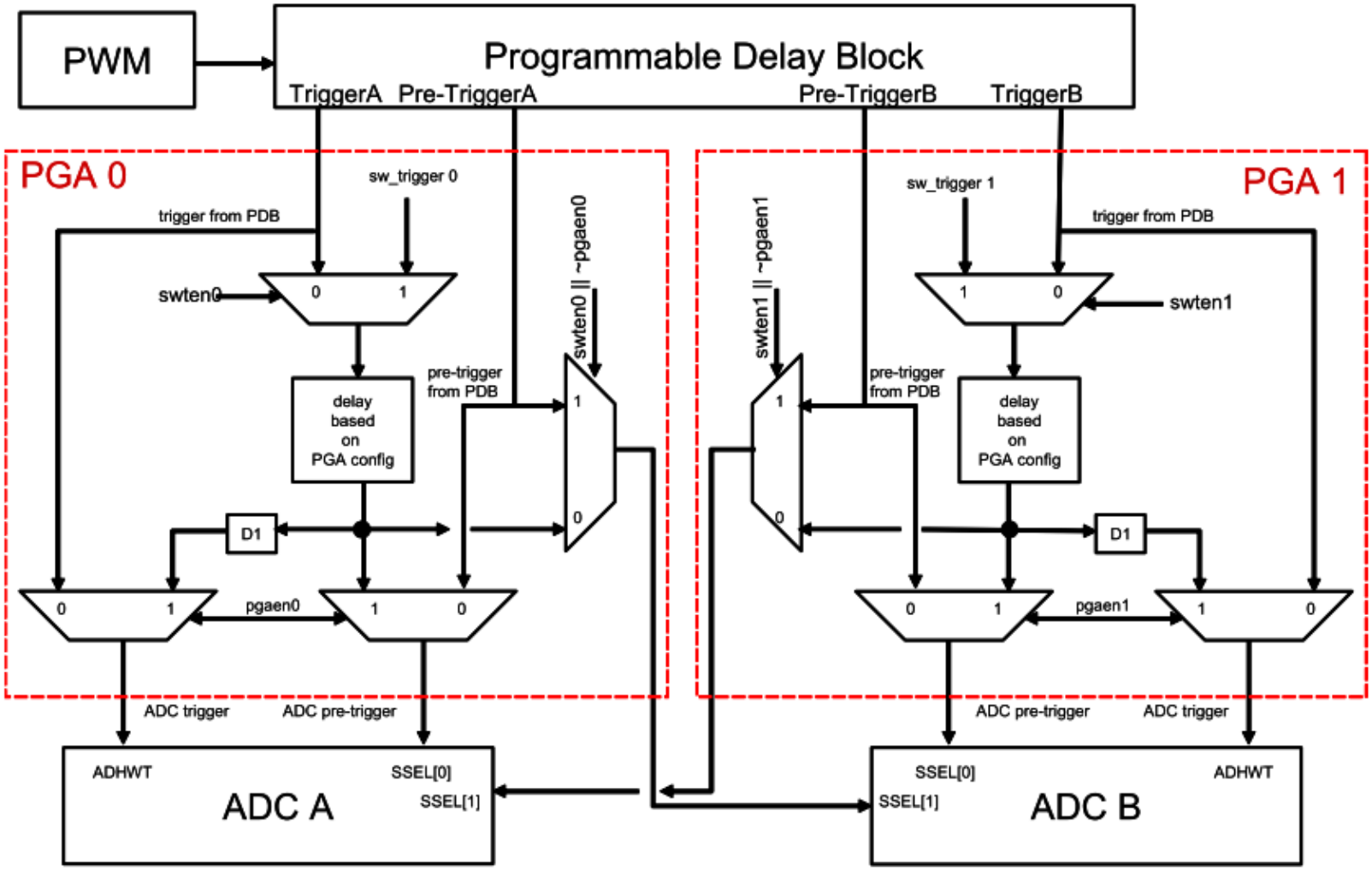
- Phase current can be sensed for certain time only



3-ph AC Induction Motor
3-ph PM Synchronous Motor

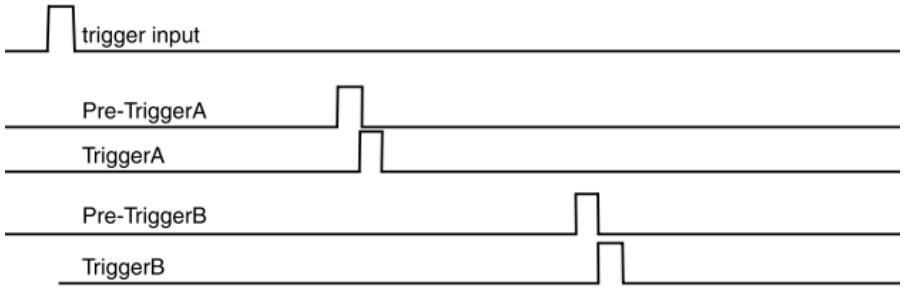


ADC to PWM Synchronization: MC56F800x

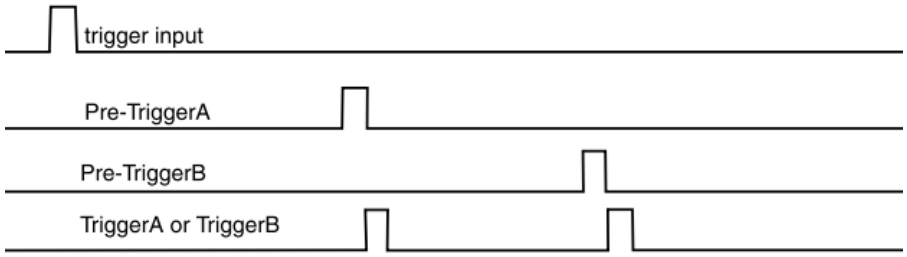


Programmable Delay Block Operation Modes

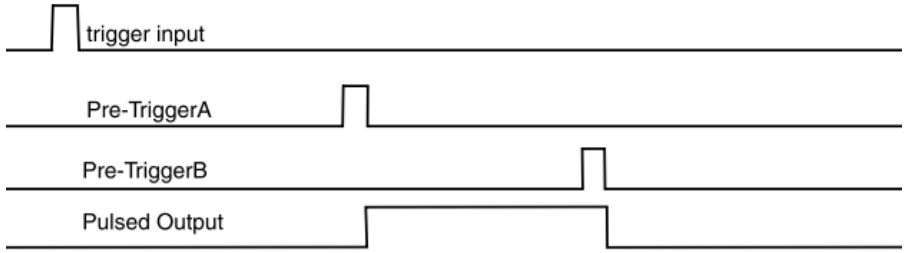
- Individual Operation
  - Each ADC is controlled individually



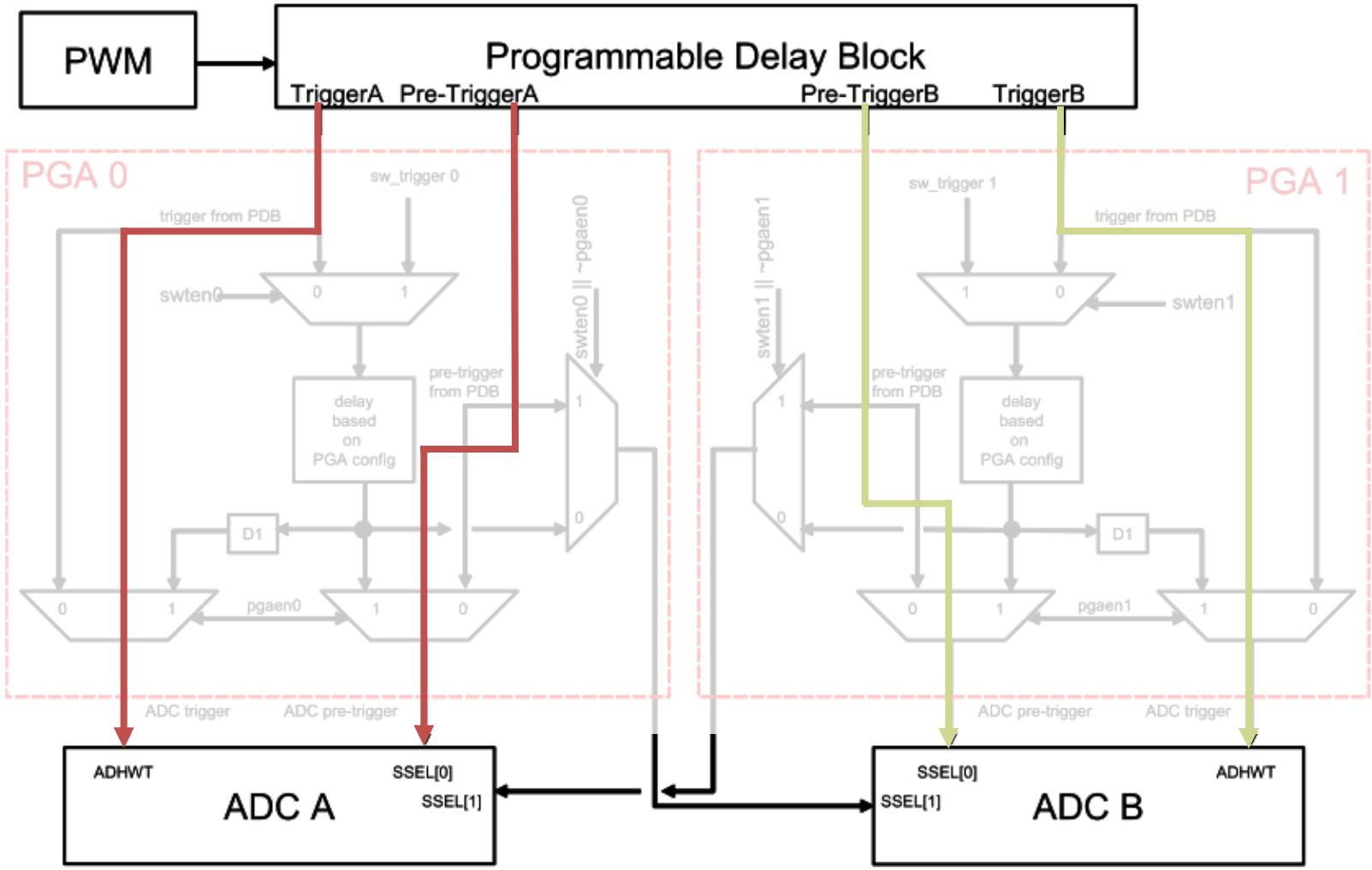
- ORed Operation
  - Both ADC converters operate in ping – pong mode. Up to four samples can be taken per conversion sequence



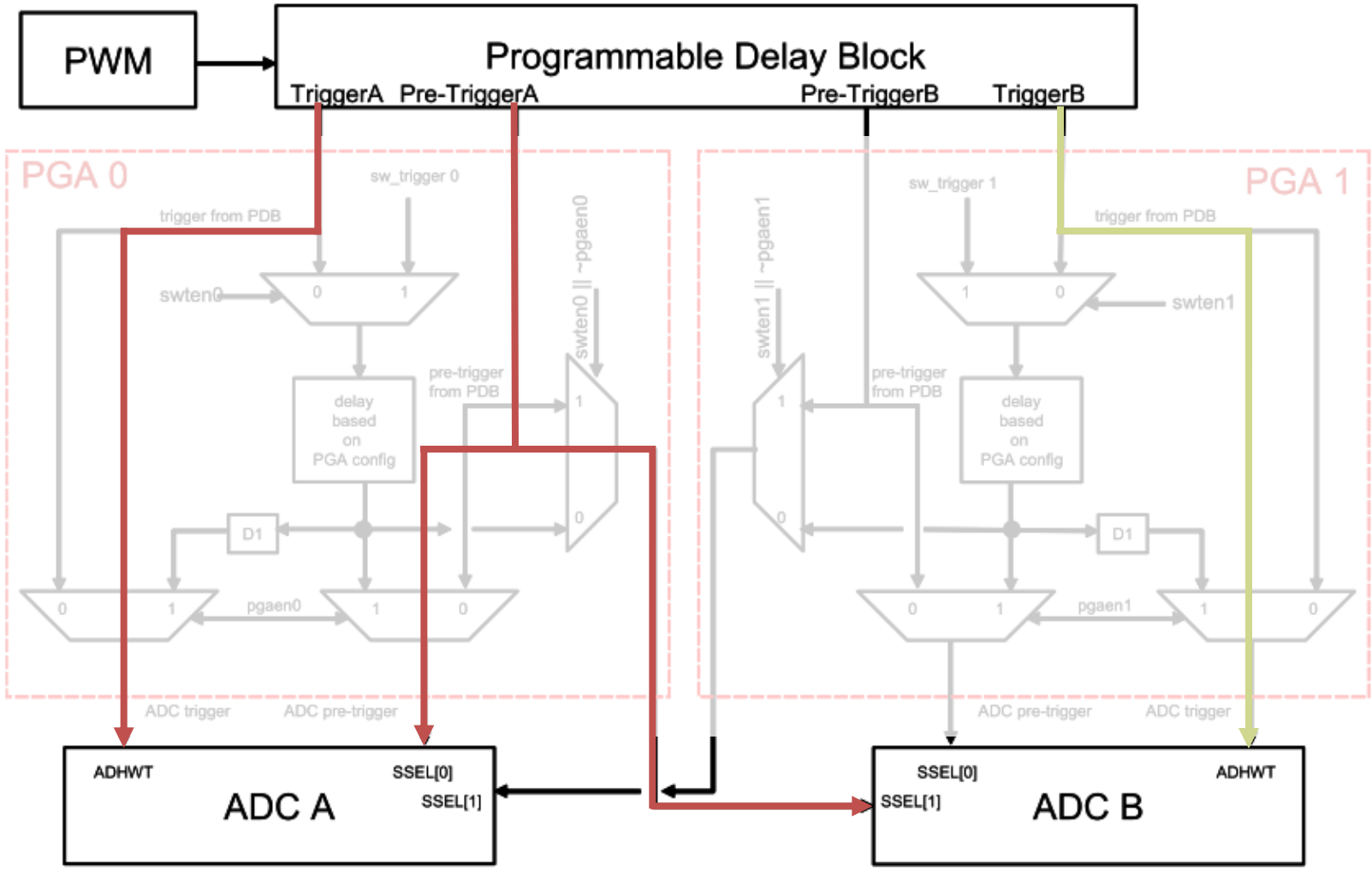
- Pulsed (PWM) Operation
  - This mode can be used for window operation of high speed comparator
  - The output can be connected to pin and generated PWM signal



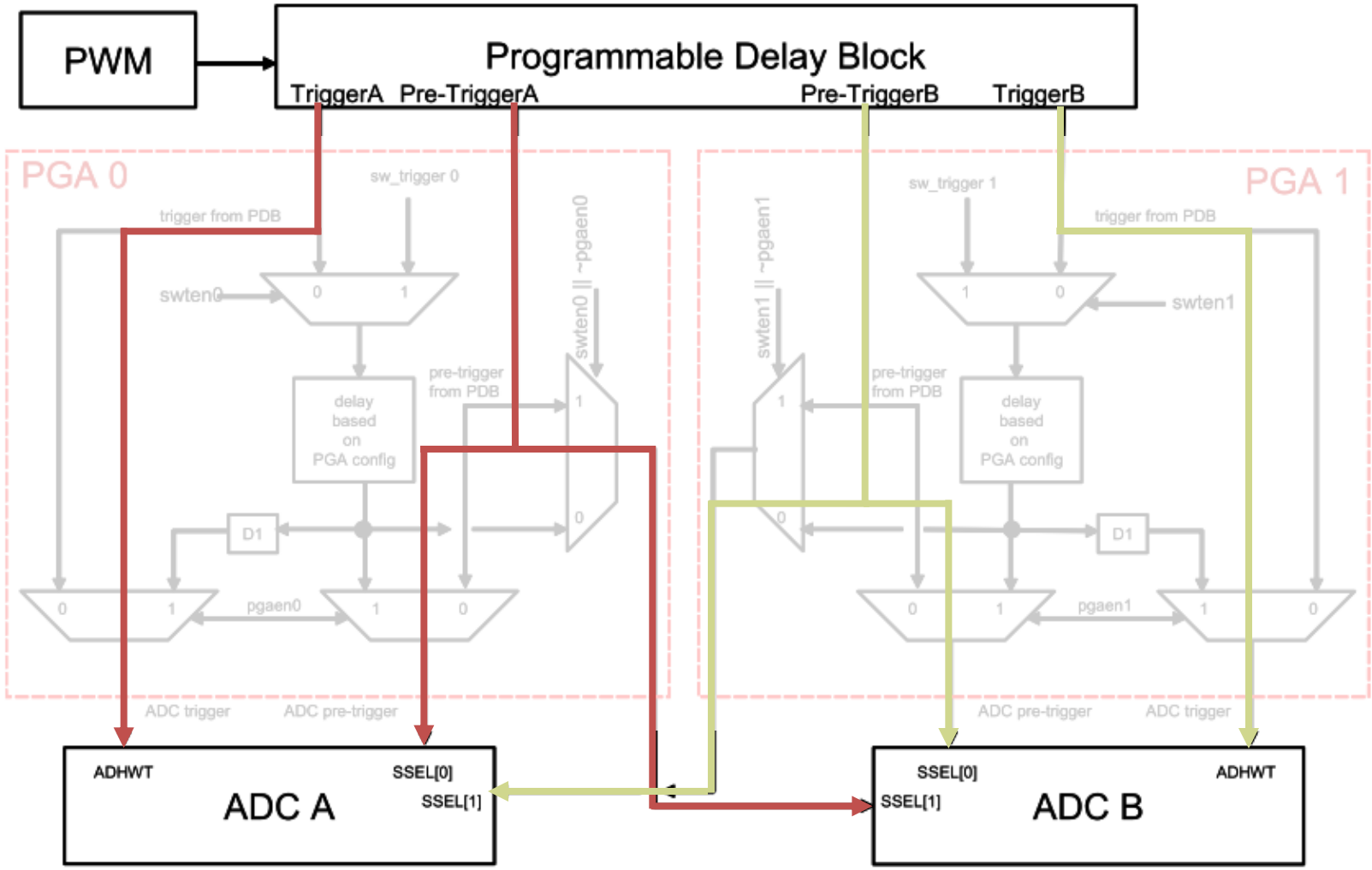
ADC to PWM Synchronization: (Individual Mode)



ADC to PWM Synchronization: (ORed Mode)



ADC to PWM Synchronization: (ORed Mode)



Agenda

- Basic Terms
- BLDC Motor Theory
- Sensorless Technique of BLDC Motors
- Microcontroller MC56F8006/2
- Freescale Software Library
  - GFLIB
  - GDFLIB
  - MCLIB
  - ACLIB

Freescale Software Library

Freescale Software Libraries

- Contains functions for Freescale 56F800E family of DSCs
- The functions are written in assembly language with a C-callable interface
- The functions are highly optimized
- The functions are comprehensively tested
- The library is distributed as an object code (compiled version)
- Detailed documentation
- Simple integration into a user application

Freescale Software Library

Freescale Software Libraries – four modules

- MCLIB – 56800E_MCLIB.lib - Motor Control Library
  - Functions optimized for controlling different types of motors (ACIM, PMSM, BLDC and SR)
- GFLIB - 56800E_GFLIB.lib - General Function Library
  - Contains plenty of useful mathematical functions (e.g. sine, cosine, sqrt, PI controllers, etc.)
- GDFLIB - 56800E_GDFLIB.lib - General Digital Filters Library
  - Contains algorithms of digital filters
- ACLIB - 56800E_ACLIB.lib - Advanced Control Library
  - Contains advanced algorithms for sensorless control of PMS motors

Freescale Software Library – GFLIB
PI Controller

Function name: **GFLIB_ControllerPIp**

Function declaration:

Frac16 **GFLIB_ControllerPIp(Frac16 f16InputErrorK,**

GFLIB_CONTROLLER_PI_P_PARAMS_T *pudtPiParams, const Int16 *pi16SatFlag)

Freescale Software Library – GFLIB PI Controller

Arguments				
Name	**Type In/Out**	**Format**	**Rangeh**	**Description**
f16InputErrorK	In	SF16	0x8000 0x7FFF	Input error at step K processed by P and I terms of the PI algorithm
pudtPiParams	In/Out	N/A	N/A	Pointer to a structure of PI controller parameters; the GFLIB_CONTROLLER_PI_P_PARAMS_T data type is defined in the header file GFLIB_ControllerPIpAsm.h
pi16SatFlag	In	N/A	N/A	Pointer to a 16-bit integer variable; if the integer variable passed into the function as a pointer is set to 0, then the integral part is limited only by the PI controller limits. If the integer variable is not zero, then the integral part is frozen immediately

Freescale Software Library – GFLIB
PI Controller

GFLIB_CONTROLLER_PI_P_PARAMS_T				
Name	**In/Out**	**Format**	**Range**	**Description**
f16PropGain	In	SF16	0x0000... 0x7FFF	Proportional gain
f16IntegGain	In	SF16	0x0000... 0x7FFF	Integral gain
i16PropGainShift	In	SI16	0...13	Proportional gain shift
i16IntegGainShift	In	SI16	0...13	Integral gain shift
f32IntegPartK	In/Out	SF32	0x80000000... 0x7FFFFFFF	State variable; integral part at step k-1; can be modified outside of the function.
f16UpperLimit	In	SF16	0x0000... 0x7FFF	Upper limit of the controller; f16UpperLimit > f16LowerLimit
f16LowerLimit	In	SF16	0x0000... 0x7FFF	Lower limit of the controller; f16UpperLimit > f16LowerLimit
i16LimitFlag	Out	SI16	0 nebo 1	Limitation flag; if set to 1, the controller output reached f16UpperLimit or f16LowerLimit

Freescale Software Library – GFLIB
PI Controller

Algorithm description

- Function calculates the PI (Proportional-Integral) algorithm according to equations below
- The PI algorithm is implemented in the parallel (non-interacting) form allowing user to define the P and I parameters independently without interaction
- The controller output is limited and the limit values (f16UpperLimit and f16LowerLimit) are defined by the user. Limits can be dynamically modified
- The algorithm returns a limitation flag (calculated internally by the algorithm)
  - Flag name: **i16LimitFlag**
  - Flag is member of the structure of PI controller parameters: **GFLIB_CONTROLLER_PI_P_PARAMS_T**
  - If the PI algorithm reaches upper or lower limit the **i16LimitFlag = 1** otherwise **i16LimitFlag = 0**

Freescale Software Library – GFLIB
PI Controller

Algorithm description cont'd

- Anti-windup strategy – implemented by limiting the integral term. Two ways of limiting the integral term
  - The integral state is limited by the controller limits, in the same way as the controller output.
  - When the variable i16SatFlag set by the user software outside the PI controller function and passed into the function and the pointer pi16SatFlag is not zero, then the integral portion is frozen.

PI algorithm in continues time domain:

$$u(t) = K\left[e(t) + \frac{1}{T_I}\int_0^{\tau} e(t)dt\right]$$

- where
  - e(t) - input error in the continuous time domain; processed by the P and I terms of the PI algorithm
  - u(t) - controller output in the continuous time domain
  - T_I - integral time constant - [s]

Freescale Software Library – GFLIB
PI Controller

PI algorithm in discrete time domain (Backward Euler method also know as backward rectangular or right-hand approximation):

$$u(k) = K \cdot e(k) + u_I(k-1) + K_I \cdot e(k)$$

$$u_I(k) = u_I(k-1) + T \cdot e(k)$$

- where
  - e(k) - input error at step k; processed by the P and I terms
  - u(k) - controller output at step k
  - K - proportional gain
  - K_I - integral gain
  - T - sampling time/period - [s]

$$K_I = K \cdot \frac{T}{T_I}$$

Freescale Software Library – GFLIB
PI Controller

PI algorithm in discrete time domain scaled into the fractional range:

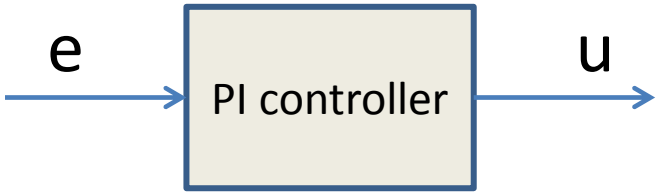
$$u_f(k) = K_{sc} \cdot e_f(k) + u_{If}(k-1) + K_{Isc} \cdot e_f(k)$$

- Where
  - e_{max} - input max physical range
  - u_{max} - output max physical range

$$u_f(k) = u(k)/u_{max}$$

$$e_f(k) = e(k)/e_{max}$$

$$K_{sc} = K \cdot \frac{e_{max}}{u_{max}}$$



$$K_{Isc} = K \cdot \frac{T}{T_I} \cdot \frac{e_{max}}{u_{max}} = K_I \cdot \frac{e_{max}}{u_{max}}$$

Freescale Software Library – GFLIB
PI Controller

Processor implementation:

- Each physical parameter is represented by two parameters in the algorithm representation
- K_{Isc} – f16IntegGain and i16IntegGainShift

$$f16IntegGain = K_{Isc} \cdot 2^{-i16IntegGainShift}$$

- K_{sc} – f16PropGain and i16PropGainShift

$$f16PropGain = K_{sc} \cdot 2^{-i16PropGainShift}$$

- where

$$0 \leq f16PropGain < 1$$

$$0 \leq f16IntegGain < 1$$

$$0 \leq i16PropGainShift < 14$$

$$0 \leq i16IntegGainShift < 14$$

Freescale Software Library – GFLIB
PI Controller

Example:

- Assumption: K_{Isc} = 2.4
  - The parameter K_{Isc} cannot be directly interpreted as a fractional value because the range of fractional values is <-1;1) and the range of the parameter f16IntegGain is <0;1).
  - It is necessary to scale the parameter K_{Isc} parameter using parameter **i16IntegGainShift** to fit the parameter **f16IntegGain** into the range <0;1)
- Solution:
  - The most precise scaling approach is to scale down the parameter K_{Isc} to have **f16IntegGain** in the following range

    0.5 <= f16IntegGain < 1

  - And to calculate the corresponding **i16IntegGainShift** parameter

Freescale Software Library – GFLIB
PI Controller

Solution cont'd:

$$\frac{\log(K_{Isc}) - \log(0.5)}{log2} \geq i16IntegGainShift$$

$$\frac{\log(2.4) - \log(0.5)}{log2} \geq i16IntegGainShift$$

$$2.26 \geq i16IntegGainShift$$

$$\frac{\log(K_{Isc}) - \log(1)}{log2} < i16IntegGainShift$$

$$\frac{\log(K_{Isc})}{log2} < i16IntegGainShift$$

$$\frac{\log(2.4)}{log2} < i16IntegGainShift$$

$$1.26 < i16IntegGainShift$$

Freescale Software Library – GFLIB
PI Controller

Solution cont'd:

- The parameter i16IntegGainShift is in the following range

$$1.26 < i16IntegGainShift < 2.26$$

- Because this parameter is an integer value, the result is

$$i16IntegGainShift = 2$$

- Then

$$f16IntegGain = K_{Isc} \cdot 2^{-i16IntegGainShift}$$

$$f16IntegGain = 2.4 \cdot 2^{(-2)} = 0.6$$

- Result:

$$\boldsymbol{f16IntegGain = 0.6}$$

$$\boldsymbol{i16IntegGainShift = 2}$$

Freescale Software Library – GFLIB
PI Controller

PI algorithm returns:

- Fractional value in the following range

$$f16LowerLimit \leq PIresult \leq f16UpperLimit$$

- PI controller parameters are in the following range

$$0 \leq f16PropGain < 1$$

$$0 \leq f16IntegGain < 1$$

$$0 \leq i16PropGainShift < 14$$

$$0 \leq i16IntegGainShift < 14$$

Freescale Software Library – GFLIB Ramp algorithms

GFLIB contains 4 implementations of the ramp algorithm

- GFLIB_Ramp16
- GFLIB_Ramp32
- GFLIB_DynRamp16
- GFLIB_DynRamp32

Let's choose one ramp implementation

- Function name: **GFLIB_Ramp16**
- Function declaration:
  Frac16 **GFLIB_Ramp16(Frac16 f16Desired, Frac16 f16Actual, const** GFLIB_RAMP16_T *pudtParam)

Freescale Software Library – GFLIB
GFLIB_Ramp16

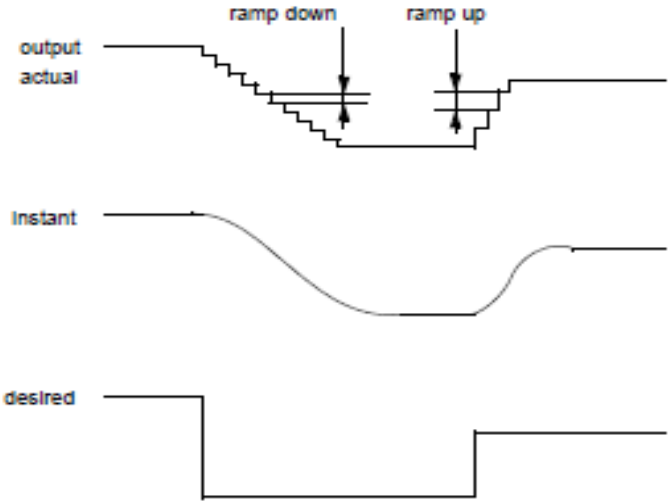
Arguments				
Name	**In/Out**	**Format**	**Range**	**Description**
f32Desired	In	SF32	0x80000000... 0x7FFFFFFF	Ramp set point. It is desired value that the ramp finally reaches
f32Actual	In	SF32	0x80000000... 0x7FFFFFFF	Actual value. Current state of the ramp output
pudtParam	In	N/A	N/A	Pointer to structure containing the ramp-up and ramp-down increments

GFLIB_RAMP32_T				
Name	**In/Out**	**Format**	**Range**	**Description**
f32RampUp	In	SF32	0x80000000... 0x7FFFFFFF	Ramp up increment
f32RampDown	In	SF32	0x80000000... 0x7FFFFFFF	Ramp down increment

Freescale Software Library – GFLIB
GFLIB_Ramp32

Algorithm description

- The ramp function calculates the 32-bit ramp of the actual value by the up or down increments contained in the **pudtParam** structure
- If the desired value is greater than the actual value, the function adds the ramp-up value to the actual value
- If the desired value is lower than the actual value, the function subtracts the ramp-down value from the actual value



Thank you