



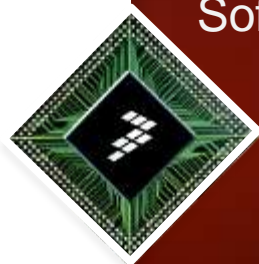
**FTF** | FREESCALE TECHNOLOGY FORUM  
POWERING INNOVATION

# How to Use and Program the New MC33816 High-Performance Fuel Injector Driver Circuit FTF-AUT-F0098

Ralph C.J. Ferrara  
Senior Systems Engineer

Tristan Bosvieux  
Automotive Application Engineer

Terry Peterson  
Software Engineer



June 2012

Freescale, the Freescale logo, AlliVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, mobileGT, PowerQUICC, Processor Expert, QorIQ, Qorivva, StarCore, Symphony and VortiQa are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, CoreNet, Flexis, MagniV, MXC, Platform in a Package, QorIQ Qonverge, QUICC Engine, Ready Play, SafeAssure, the SafeAssure logo, SMARTMOS, TurboLink, Vybrid and Xtrinsic are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © 2012 Freescale Semiconductor, Inc.



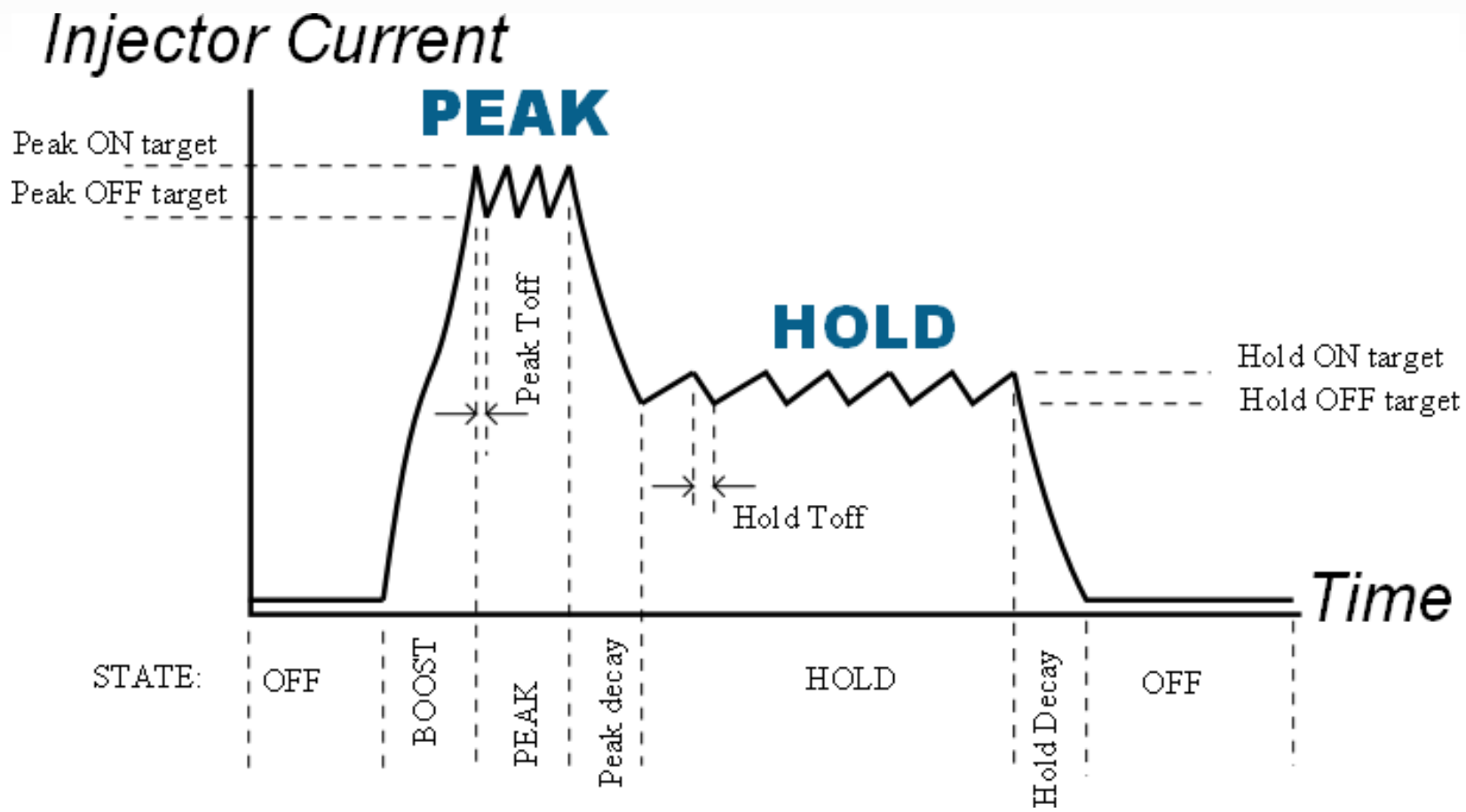
# Agenda

- Purpose for introducing the MC33816
- Overview of MC33816
  - Hardware – Circuit functions and features
  - Software – Instruction set and tools
- MC33816 Applications and Examples
  - 3, 4 and 6 cylinder examples
  - Peak and Hold waveform generation
  - Boost voltage regulator

# Why is the MC33816 “Smart Injector Driver” Important?

- Many engine manufacturers are turning to Direct Diesel Injection (DDI) and Gasoline Direct Injection (GDI) to increase fuel efficiency and reduce engine emissions, hence the need for a “smart” injector driver.
- A “smart” injector driver can off-load tasks from the Engine Control Unit’s (ECU’s) microcomputer (MCU) by performing the high-speed control algorithms and generating the high voltage necessary to produce the complex peak and hold injector current waveform.

# Typical Direct Injector Current Waveform



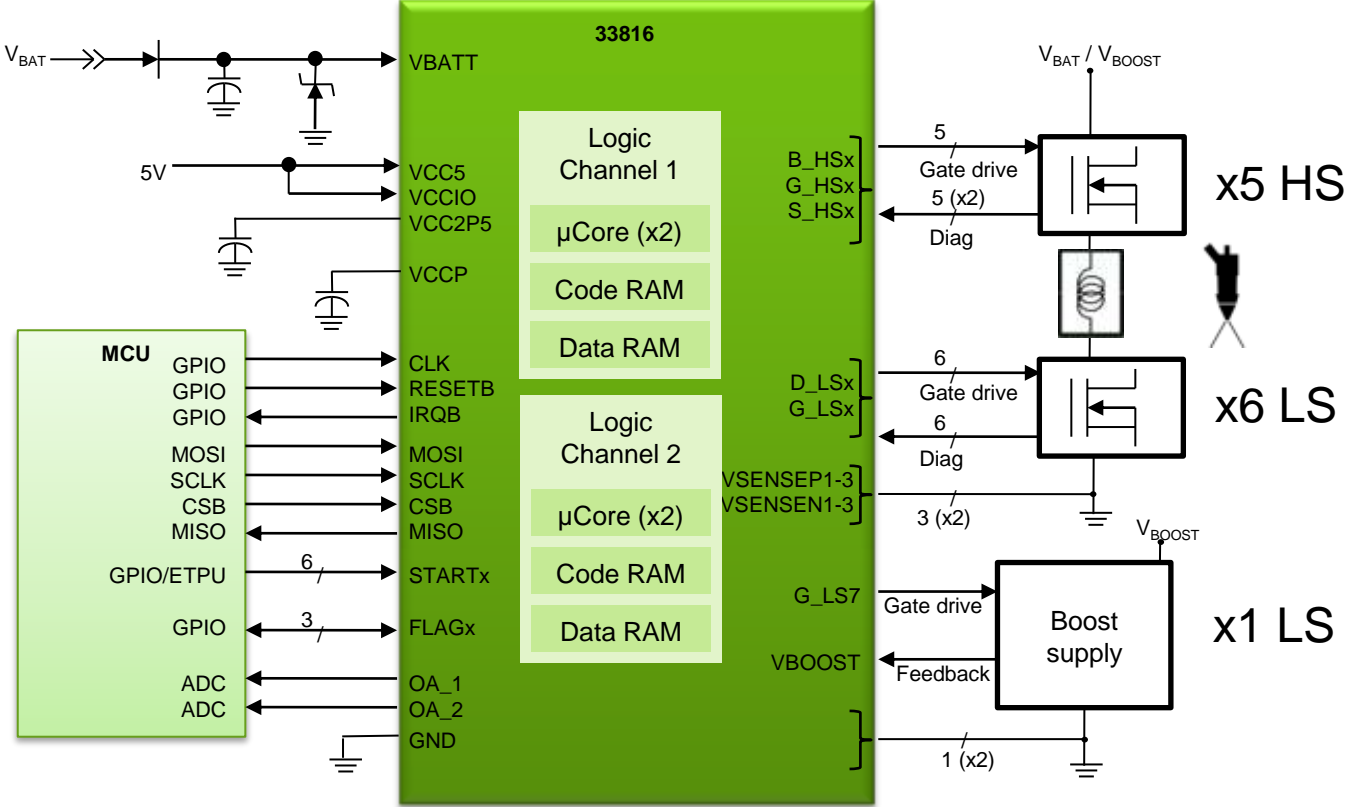


# Agenda

- Purpose for introducing the MC33816
- Overview of MC33816
  - Hardware – Circuit functions and features
  - Software – Instruction set and tools
- MC33816 Applications and Examples
  - 3, 4 and 6 cylinder examples
  - Peak and Hold waveform generation
  - Boost voltage regulator

# MC33816 Simplified Application Drawing

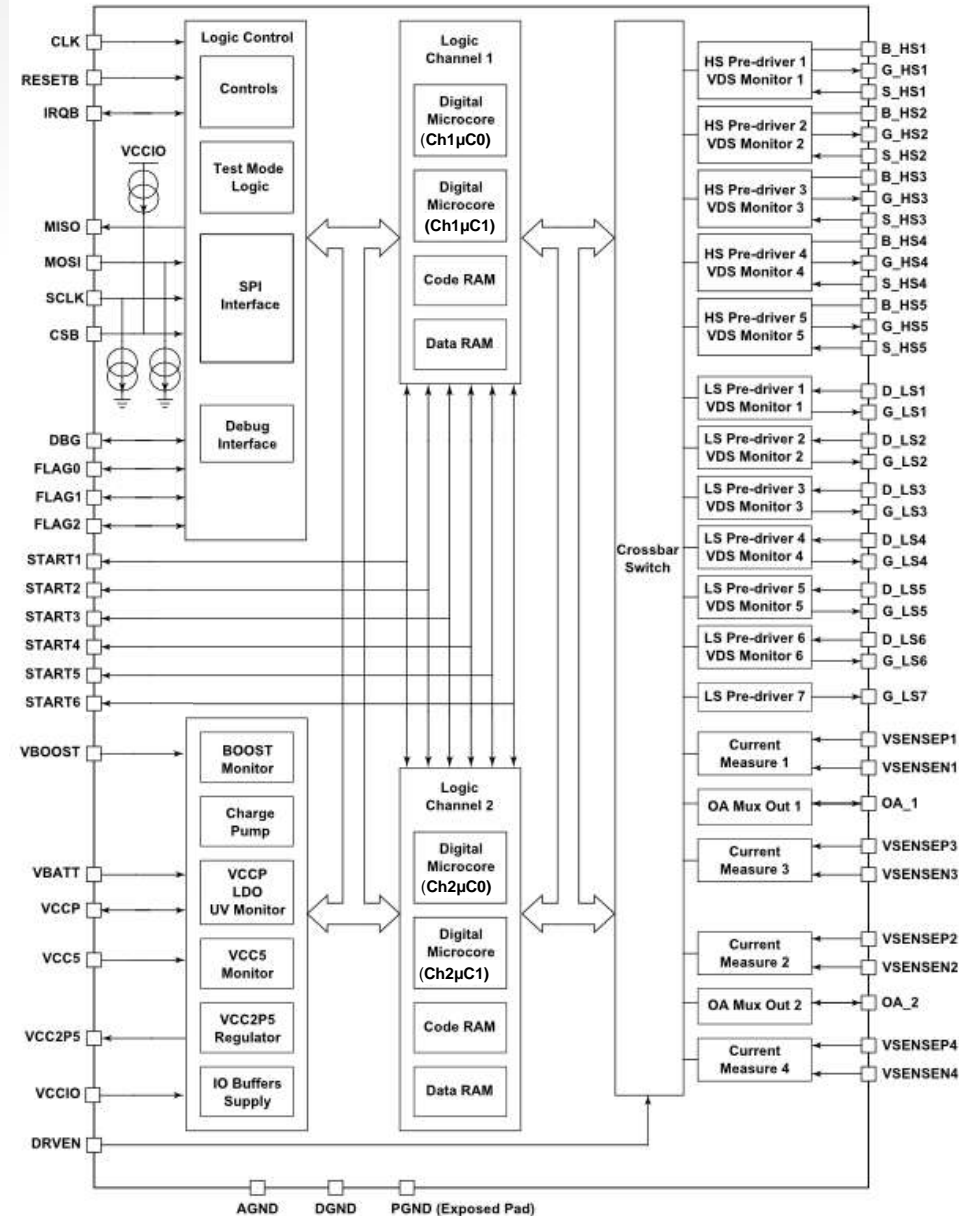
- **Automotive (12V), Truck and Industrial (24V) Applications - Engine Management Systems**
  - GDI / DDI programmable pre-drivers
  - Positioning and dithering of transmission valves
  - Solenoid and valve actuation



# NXP C33816 Internal Block Diagram

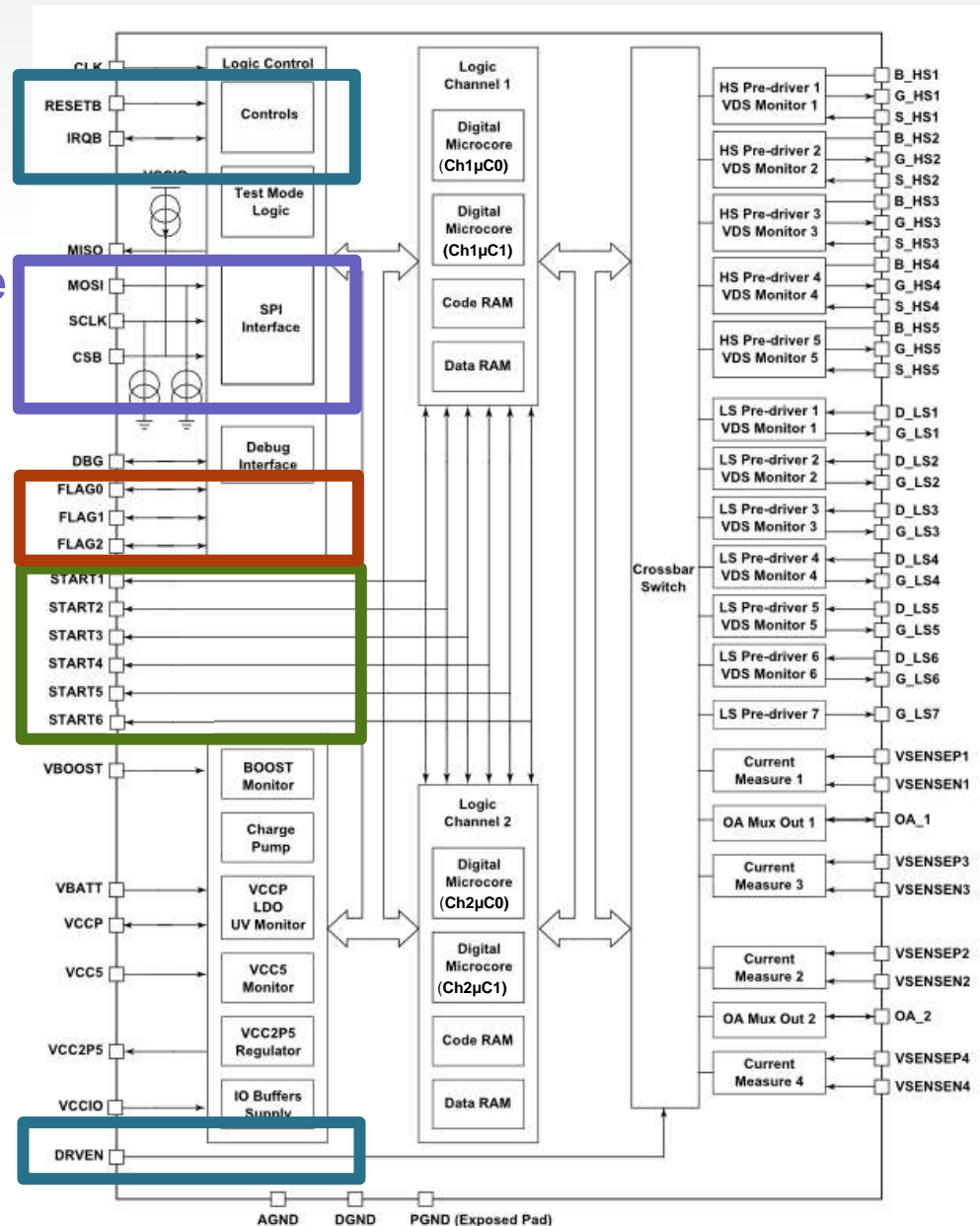
## Key Features:

- 5 High Side (HS) and 7 Low Side (LS) MOSFET pre-drivers
- 2 Logic Channels, consisting of 2  $\mu$ Cores each, for up to 4 simultaneous threads of code execution
- Separate Code and Data RAM blocks for each Logic Channel
- 4 current measurement blocks for closed loop current control
- Built-in fault diagnosis and protection
- Power Supply monitoring
- HV DC-DC converter circuitry



# NXP Communication Interfaces

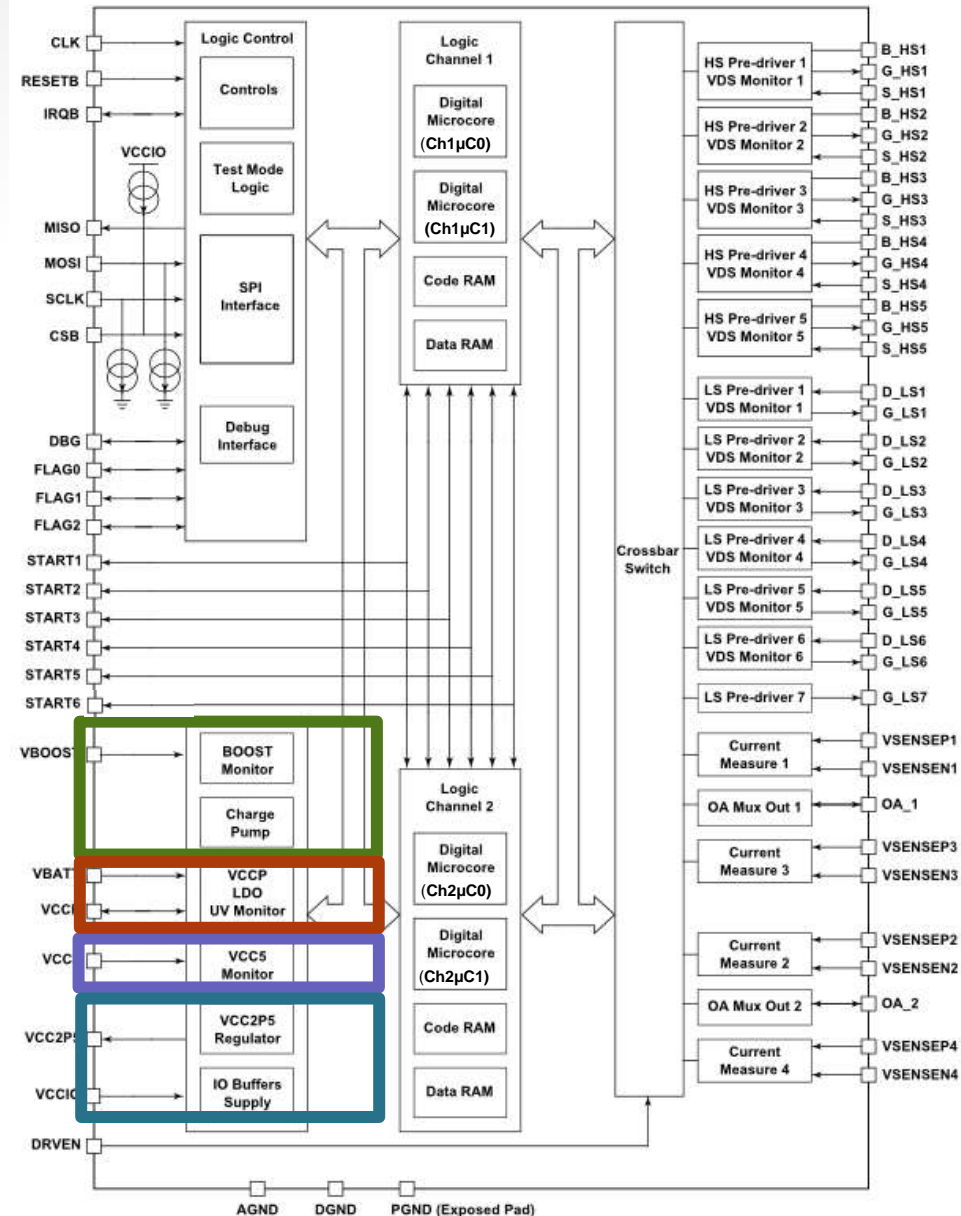
- Communication with MCU via  $\leq 10$  mbps SPI interface
- 6 start inputs (STARTx)
- 3 general purpose I/Os (FLAGx)
- Reset and IRQ pins
- Pre-driver enable input (DRVEN)





# NXP Power Supplies and Monitoring

- Boost voltage monitor input with Integrated Charge Pump
- Integrated 7.0V linear regulator (VCCP) for pre-driver power supply (can be externally supplied for 24V battery systems), with under-voltage monitoring
- External VCC5 (5.0V) supply input with under/over-voltage monitoring
- Integrated 2.5V linear regulator for digital core supply, sourced by VCC5 input supply, with under-voltage monitoring
- VCCIO external supply (5.0V or 3.3V) to select reference for digital I/O
- Built-in thermal monitoring for additional protection



# Logic Channels (2x) and PLL Overview

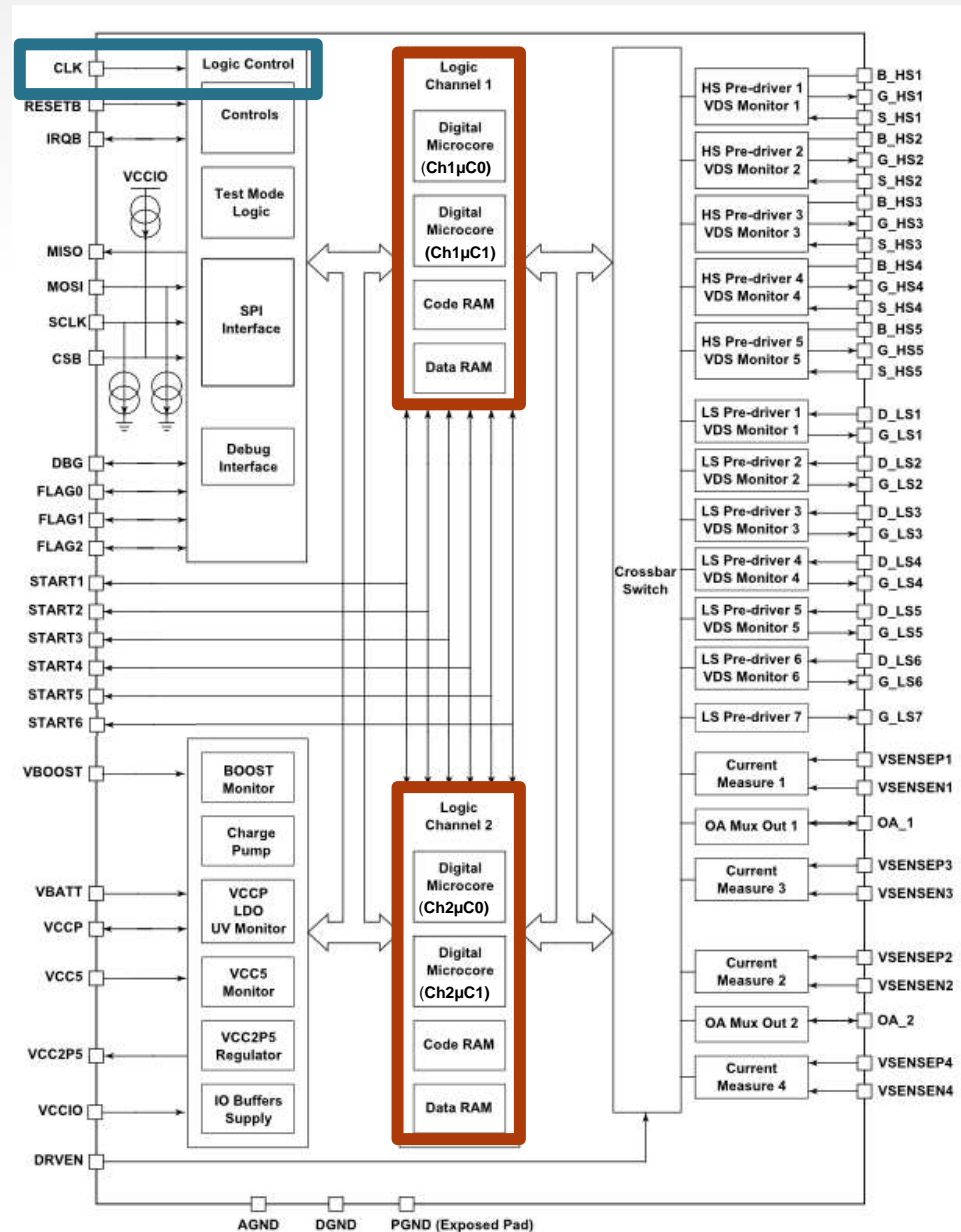
## 2 Logic Channels

Each logic channel consists of:

- 2  $\mu$ Cores
- Code RAM: 1024 x 16 bits
- Data RAM: 64 x 16 bits
- 2  $\mu$ Cores share 1 Code RAM and 1 Data RAM

## 24 MHz, PLL-generated, system clock sync'd to 1.0 MHz clock

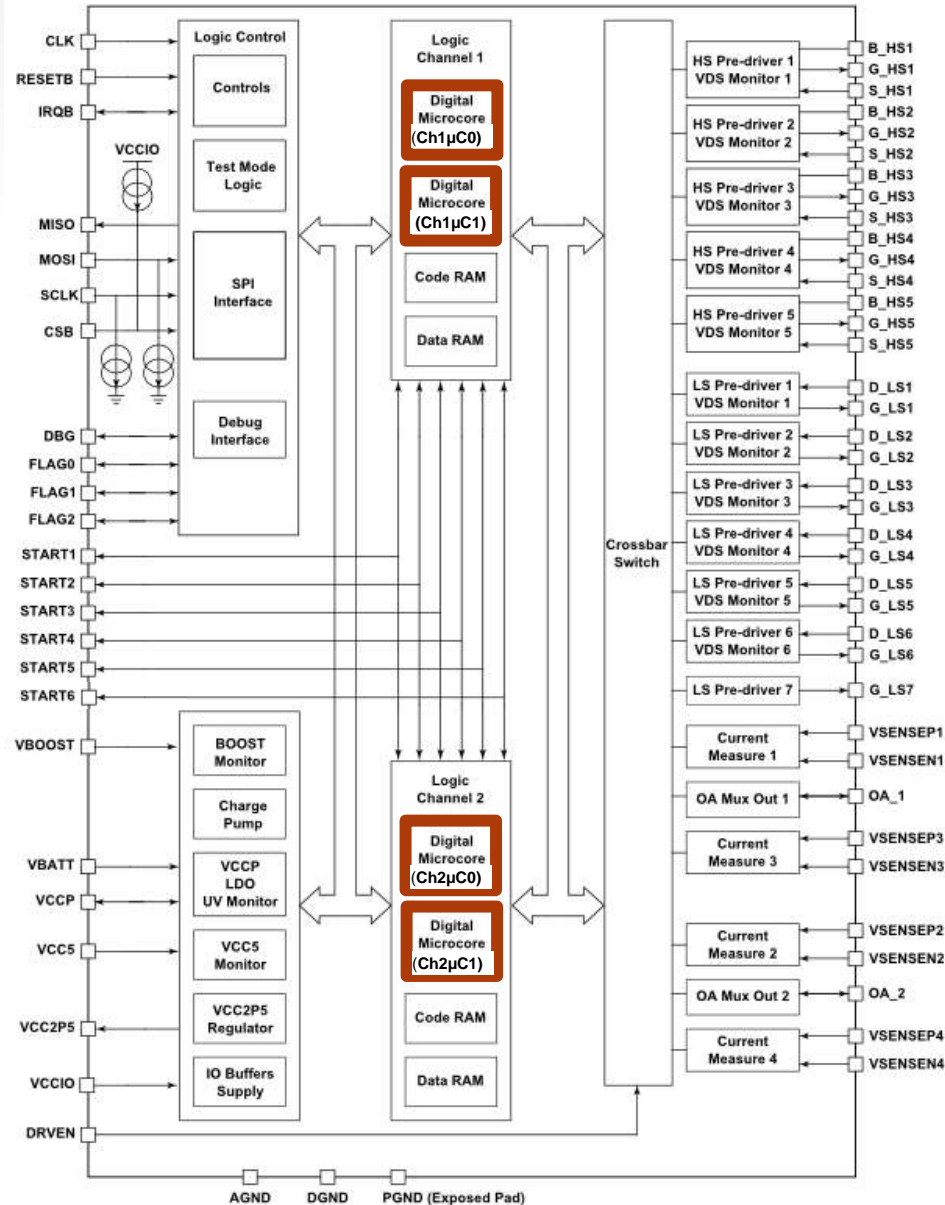
- 1 MHz clock can be provided externally or generated internally.
- Switches automatically to internal clock

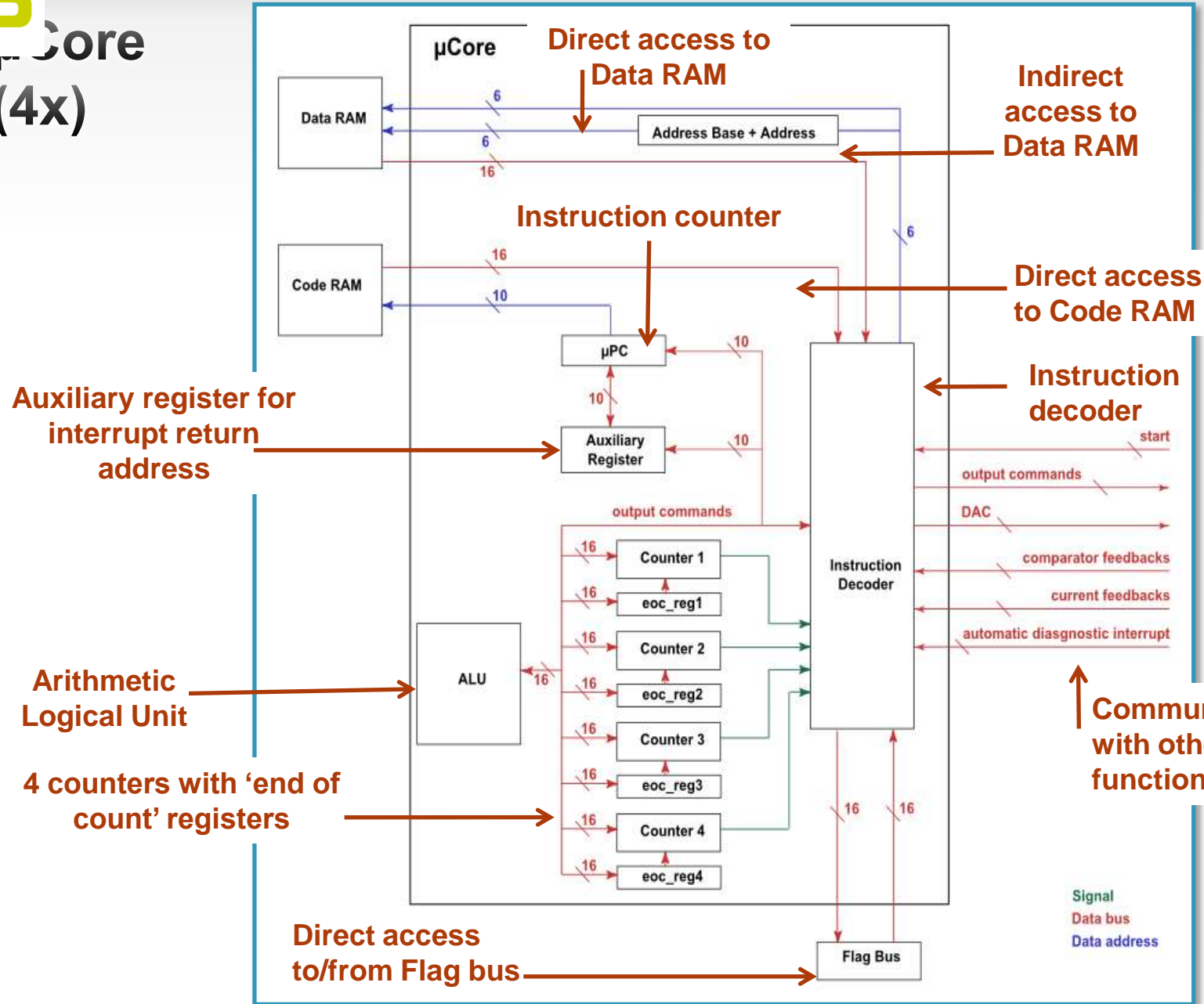


# μCores (4x)

## Features:

- **Single cycle operations:**
  - Addition
  - Subtraction
  - Logic operations (not, or, and, xor)
  - Shift 8 positions
- **Multiple cycle operations:**
  - Multiplication (32 steps)
  - Shift “n” positions (“n” steps)
- **8 General Purpose Registers (16 bits each)**
- **Arithmetic Condition Register**





Auxiliary register for interrupt return address

Arithmetic Logical Unit

4 counters with 'end of count' registers

Direct access to/from Flag bus

Indirect access to Data RAM

Direct access to Code RAM

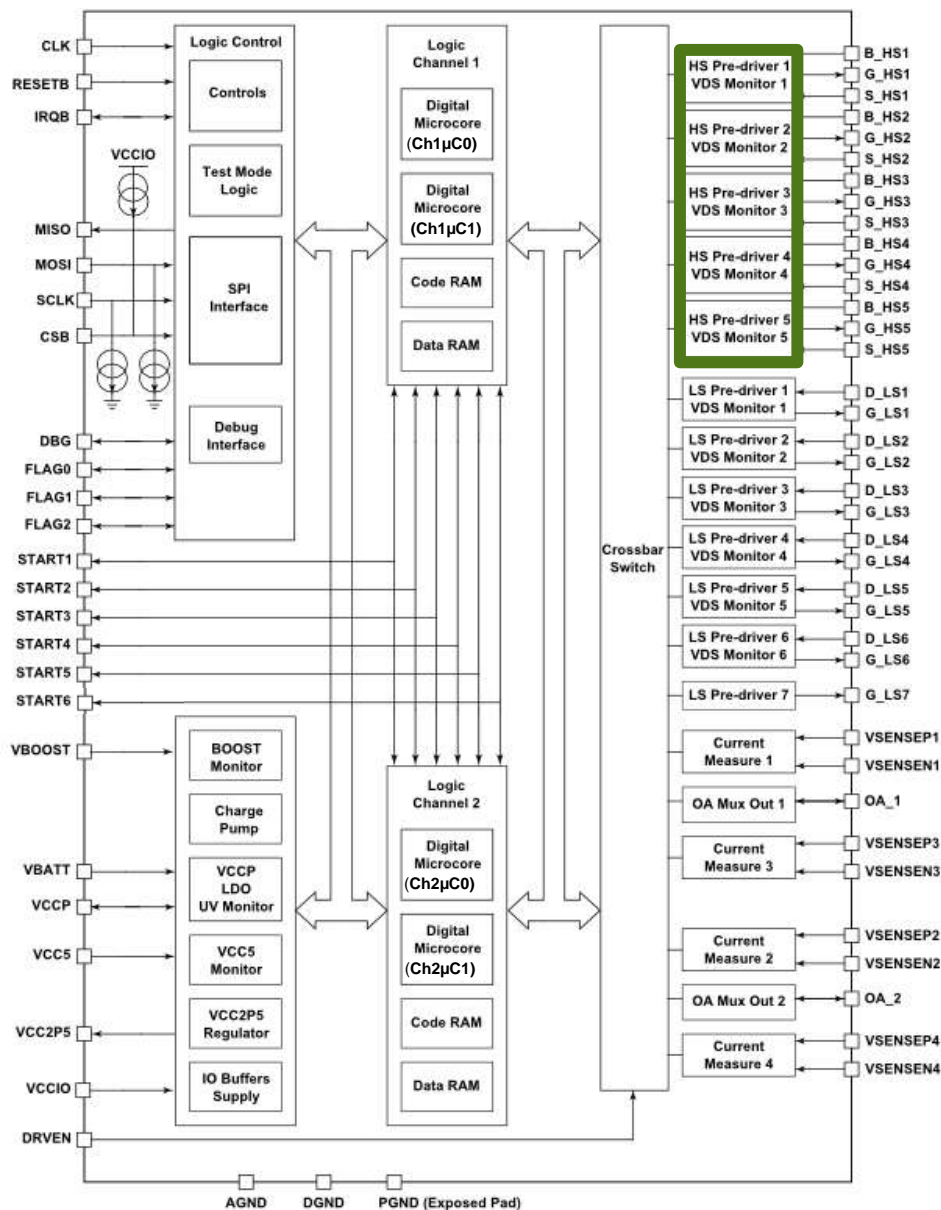
Instruction decoder

Communication with other device functional blocks

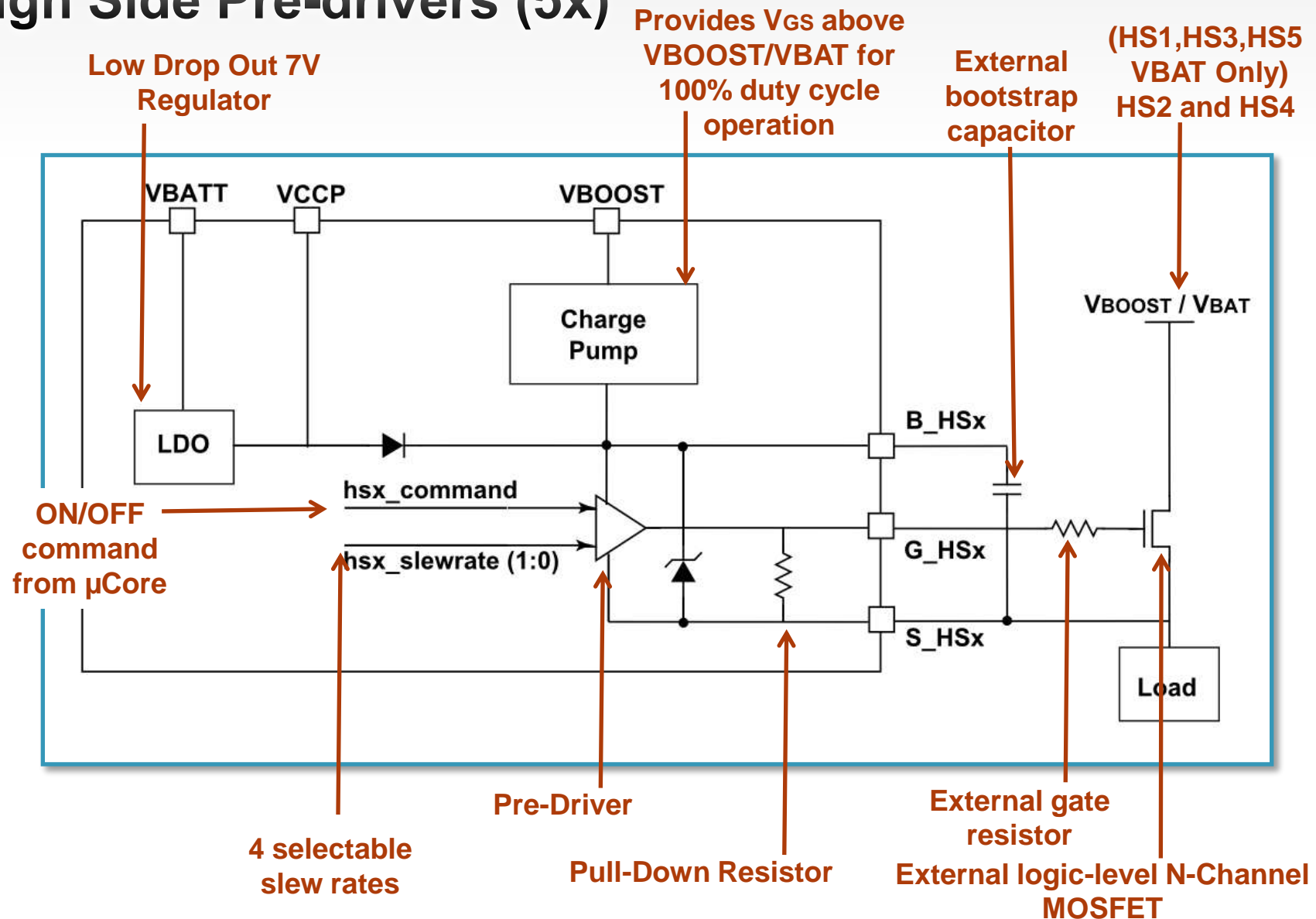
# NXP High Side Pre-drivers (5x)

## 5 high-side pre-drivers

- Require logic-level N-channel MOSFETs
- 4 programmable slew rates
- $V_{S\_HSx} + 4V < V_{B\_HSx} < V_{S\_HSx} + 8V$
- $V_{S\_HSx}$  maximum is 72V
- HS2, HS4 can drive MOSFETs referenced to  $V_{BAT}$  or  $V_{BOOST}$
- HS1, HS3, HS5 can only drive MOSFETs referenced to  $V_{BAT}$
- Built-in bootstrap circuitry
  - ✓ Requires external capacitor
- Built-in charge pump with 100% duty cycle capability
  - ✓ No external capacitor required
- All high-side drivers can also be used as low-side drivers

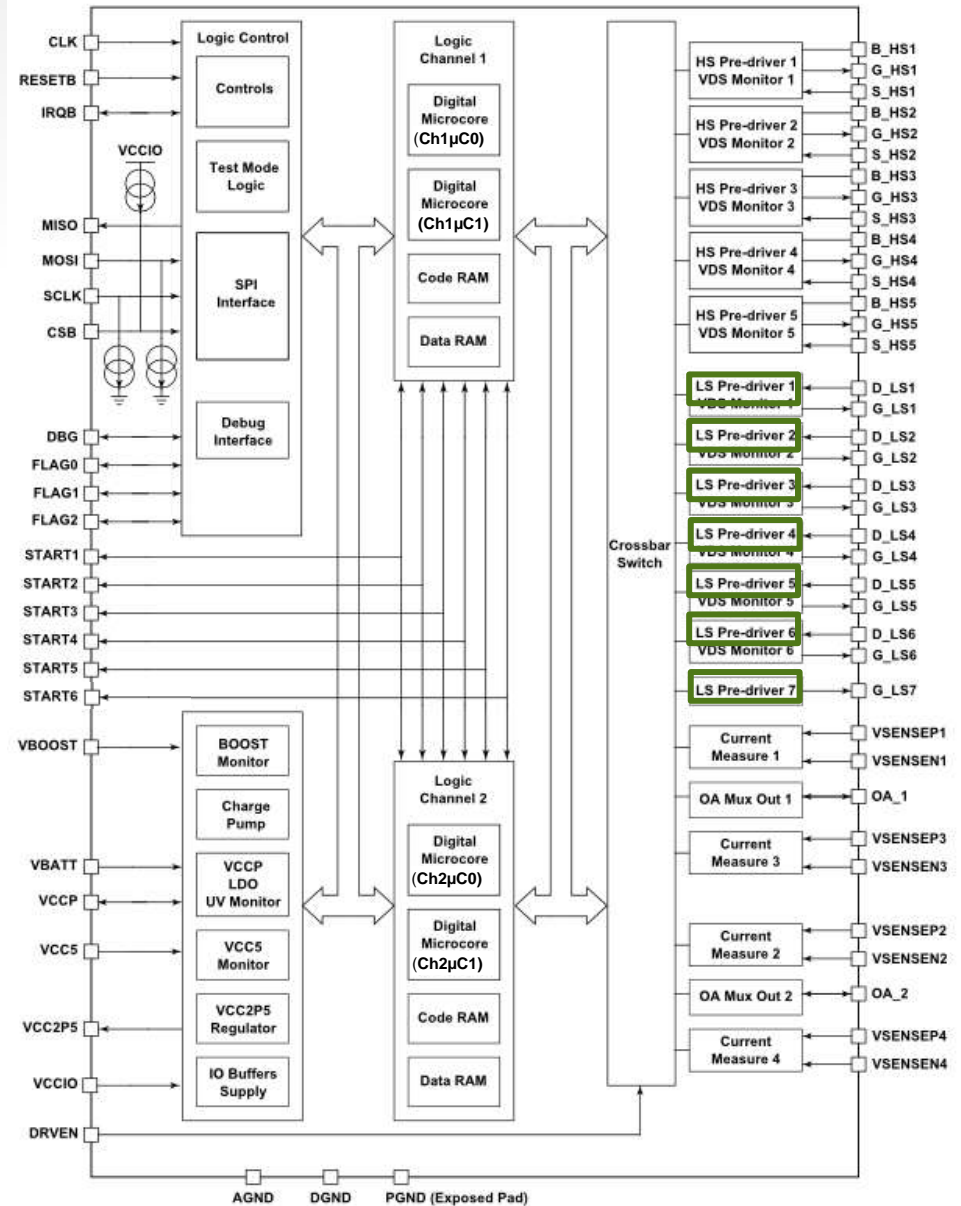


# High Side Pre-drivers (5x)

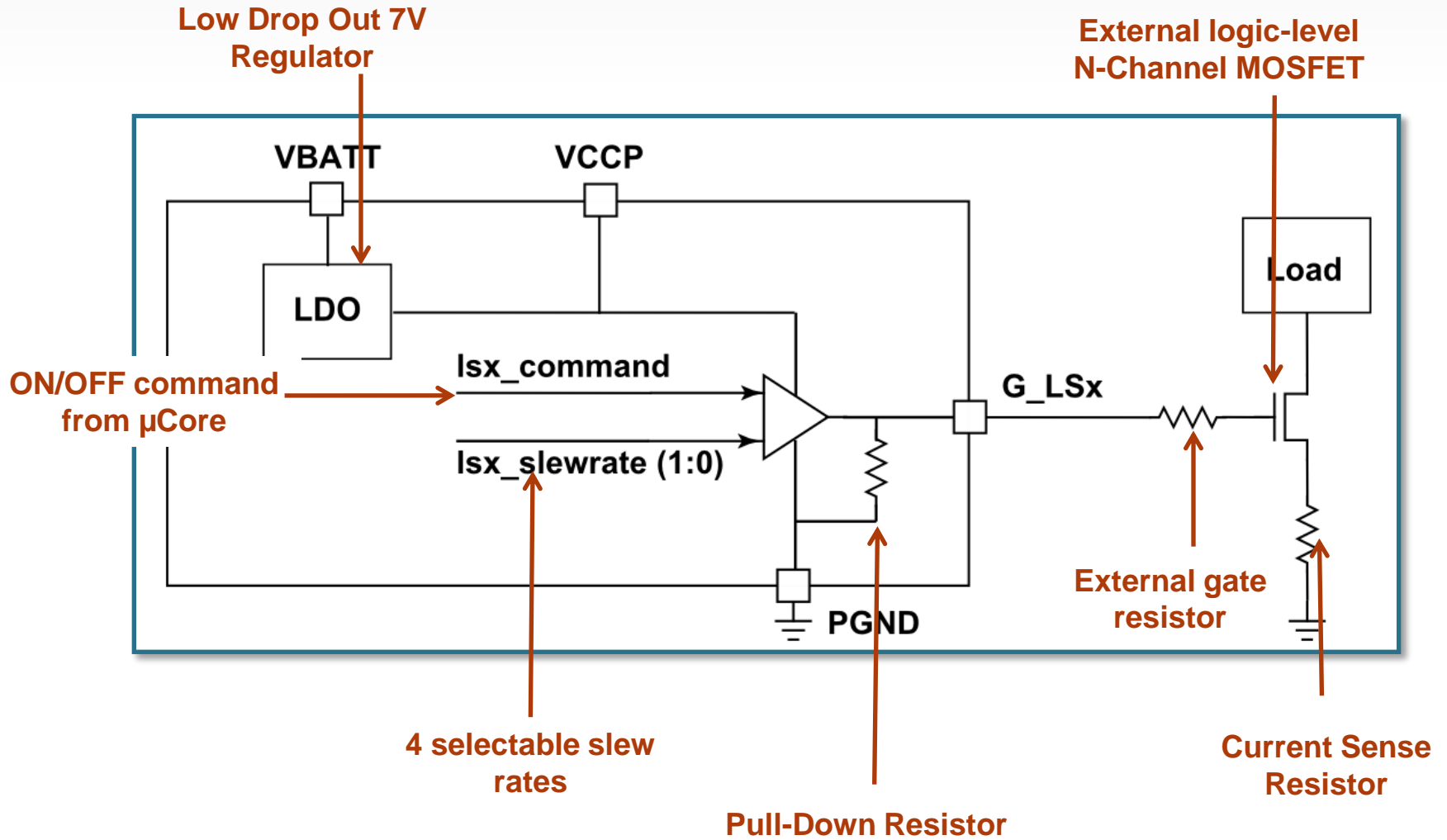


# Low Side Pre-drivers (7x)

- 7 low-side pre-drivers for driving logic level N-channel MOSFETs
- 4 programmable slew rates
- LS7 optionally dedicated to DC-DC converter



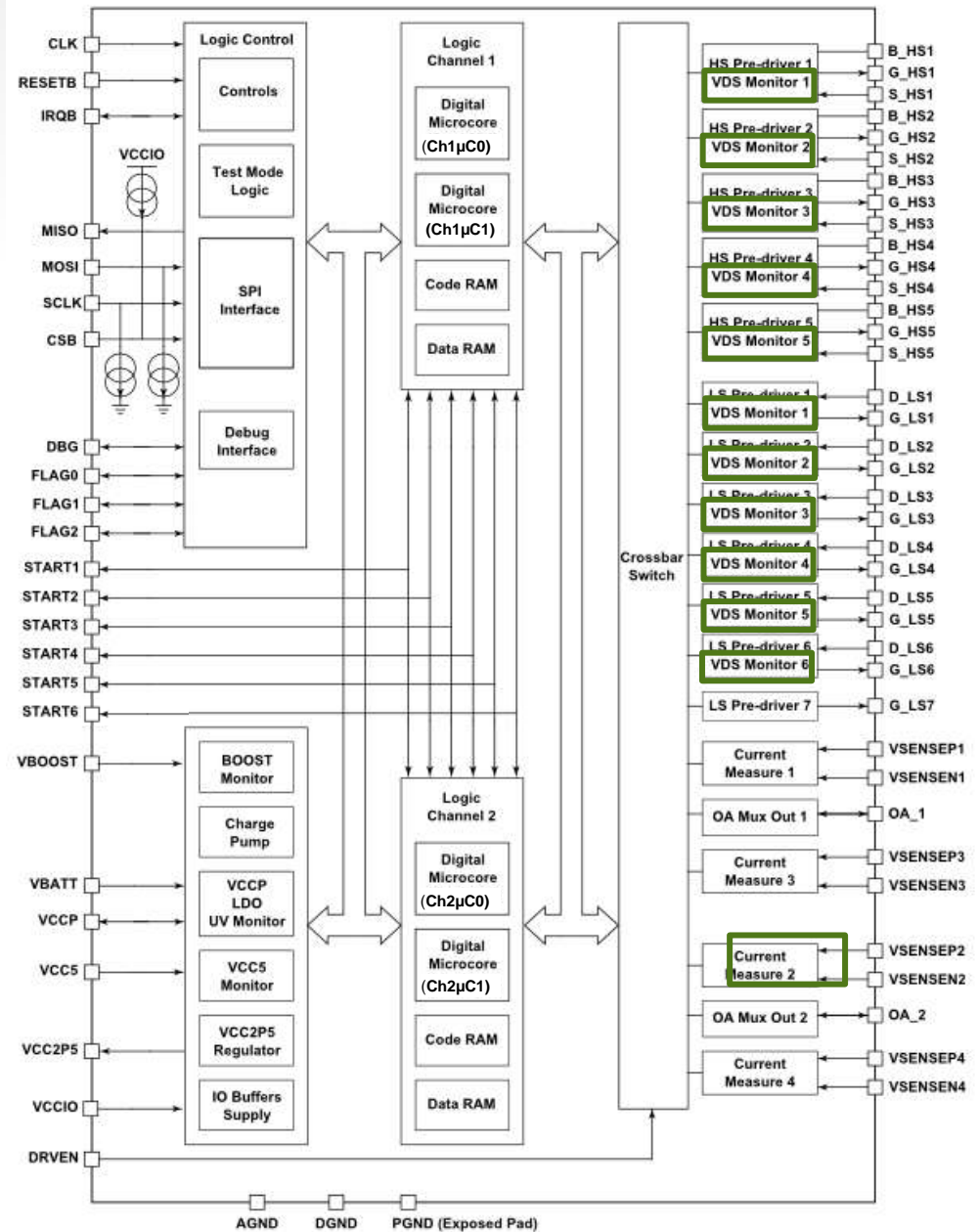
# Low Side Pre-Drivers (7x)





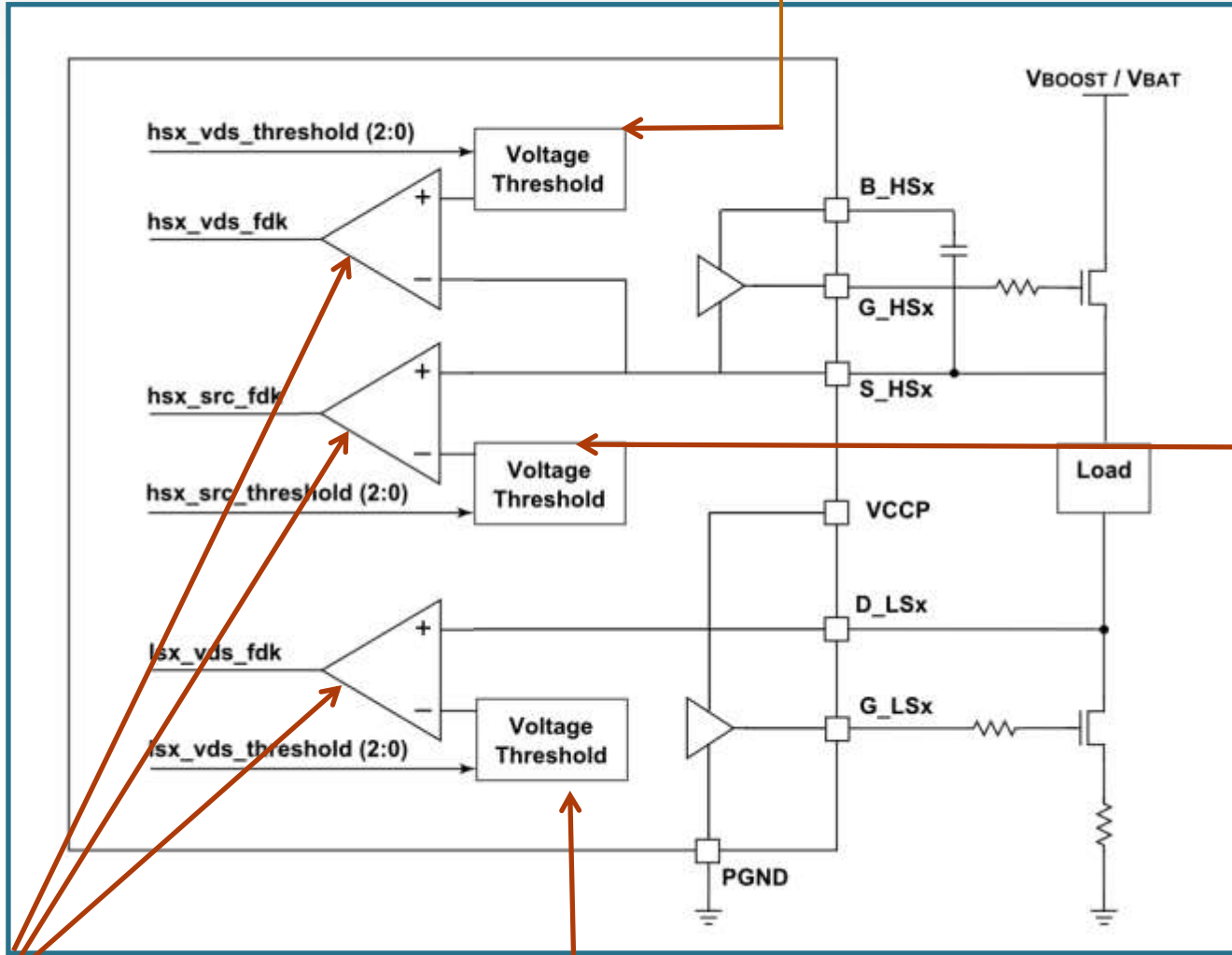
# NXP QoS and VSRC Monitors

- 5 independent high-side VDS monitors:
  - 2 dedicated to VBOOST or VBAT voltage
  - 3 dedicated to VBAT voltage only
- 5 independent high-side VSRC monitors
- 6 independent low-side VDS monitors
- Threshold values changed in data registers via microcode or SPI



# VDS and VSRC Monitors

8 selectable VDS thresholds for high sides



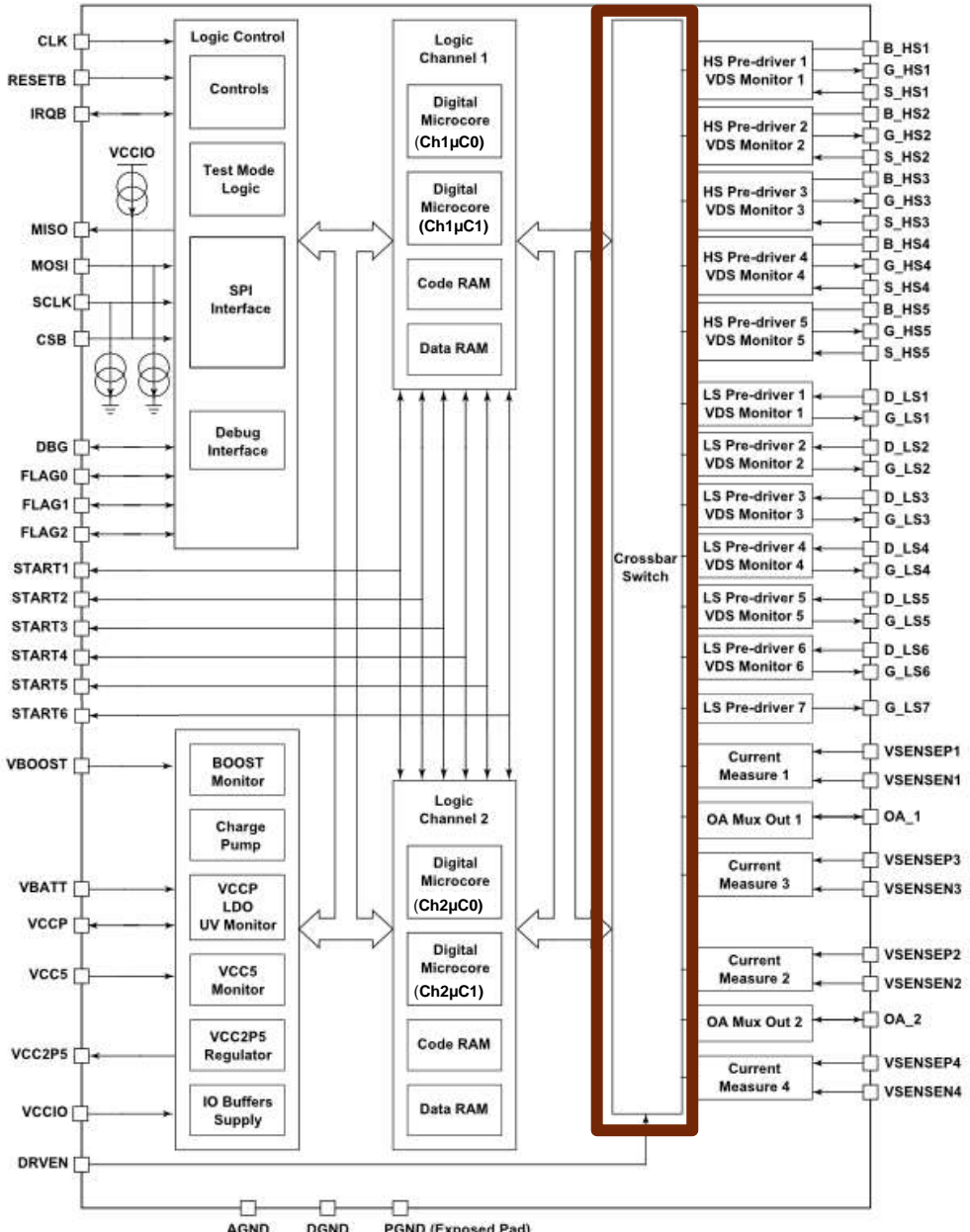
8 selectable VRSC thresholds for high sides

Comparators provide digital fault indications

8 selectable VDS thresholds for low sides

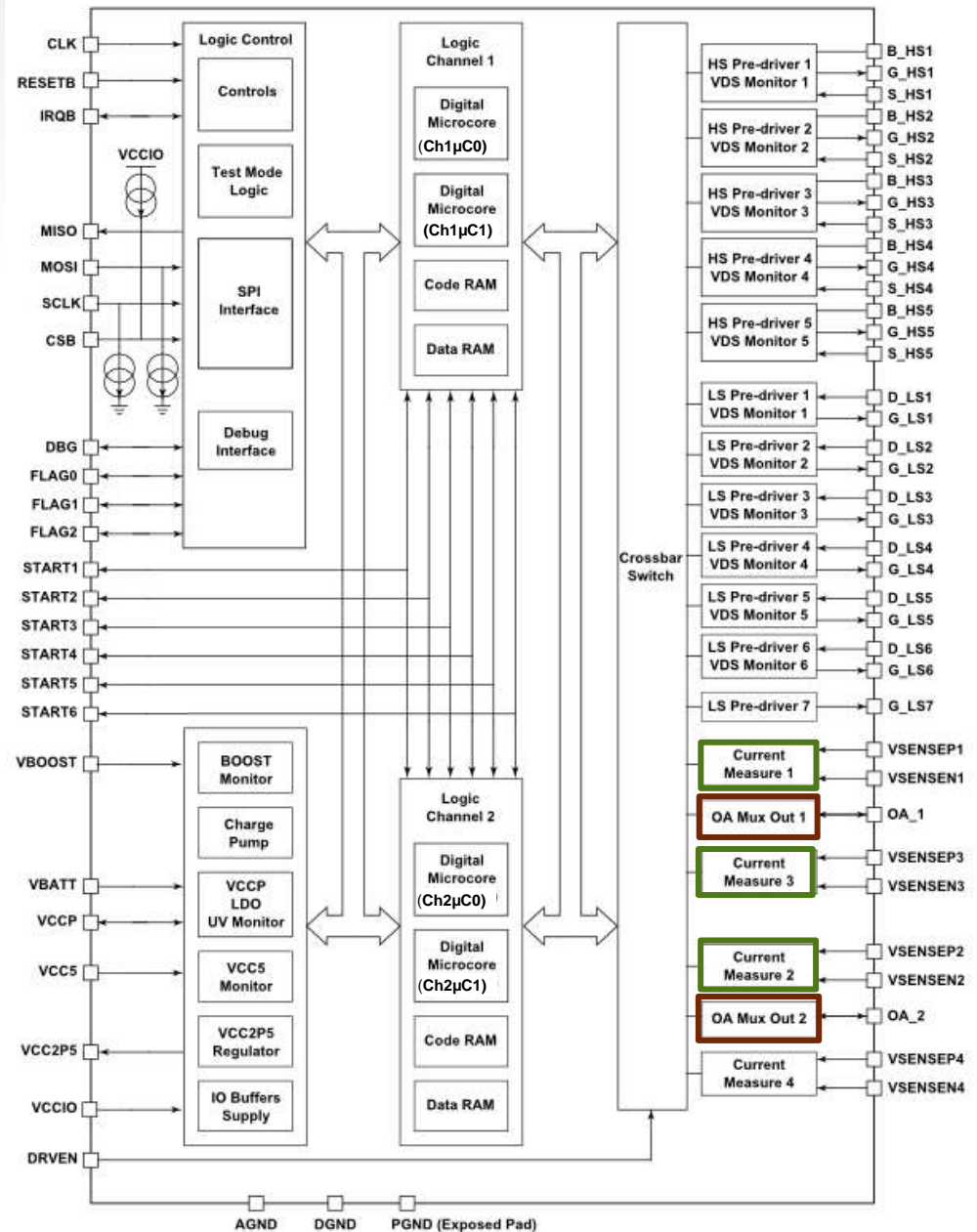
# Crossbar Switch

- The Crossbar Switch configures connections between  $\mu$ Cores and analog resources
- Maps  $\mu$ Core control of:
  - Outputs
  - Slew rate
  - Biasing
  - VDS Monitors
  - Current sense blocks gain
  - Current sense blocks feedback gain
  - VBOOST DAC setting
  - VBOOST current feedback gain

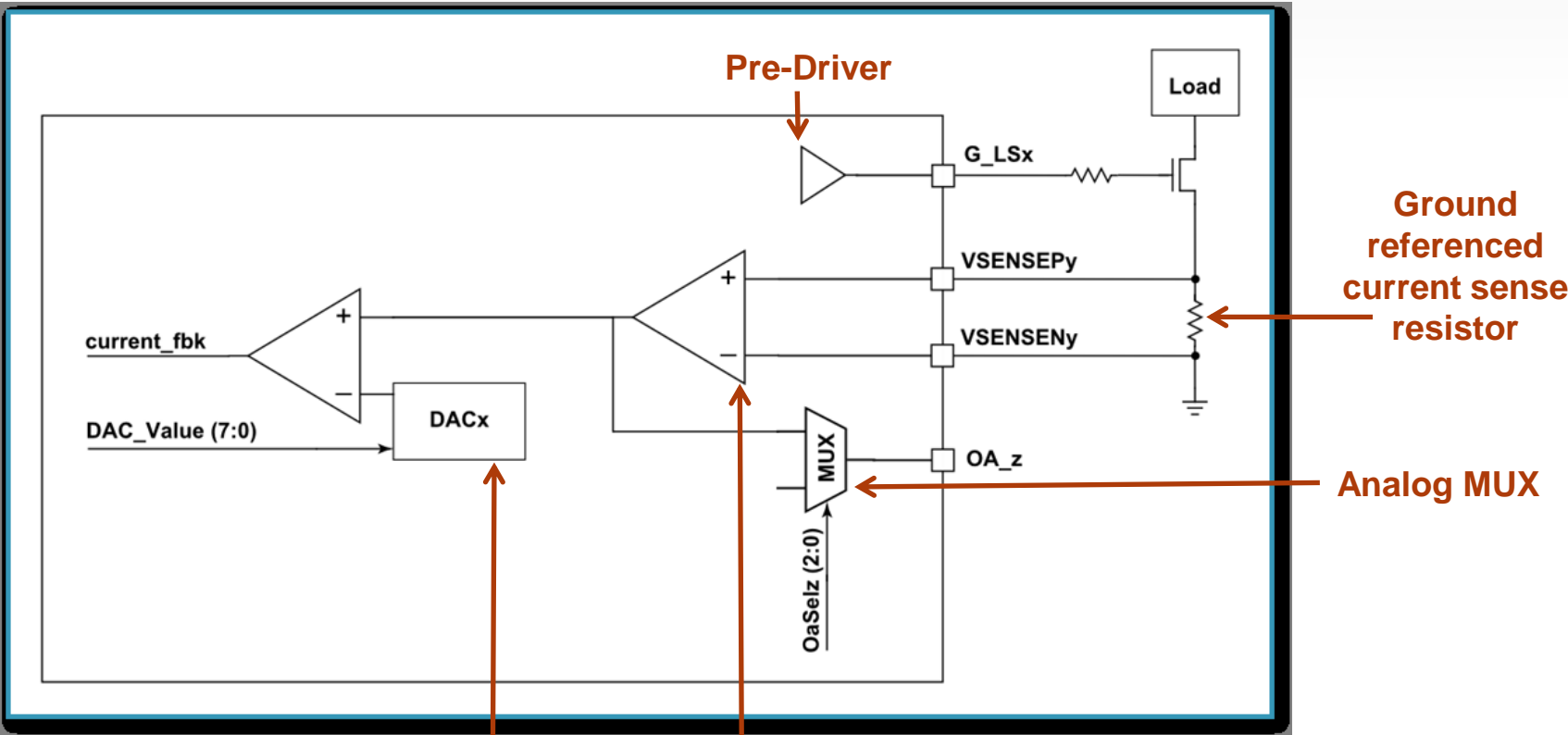


# Current Sense 1,2 & 3 and OA\_x Blocks

- 3 standard current measurement blocks – single threshold current comparison
- ‘Shunt resistor, connected to ground’ topology must be used
- 4 values of gain, selected by microcode or SPI
- VSENSE differential voltage can also be routed to OA\_x pin  
ADC mode  
Automatic offset compensation



# Current Sense Blocks 1,2 & 3 and OA\_x Blocks



8 bit DAC sets thresholds via SPI or  $\mu$ Core

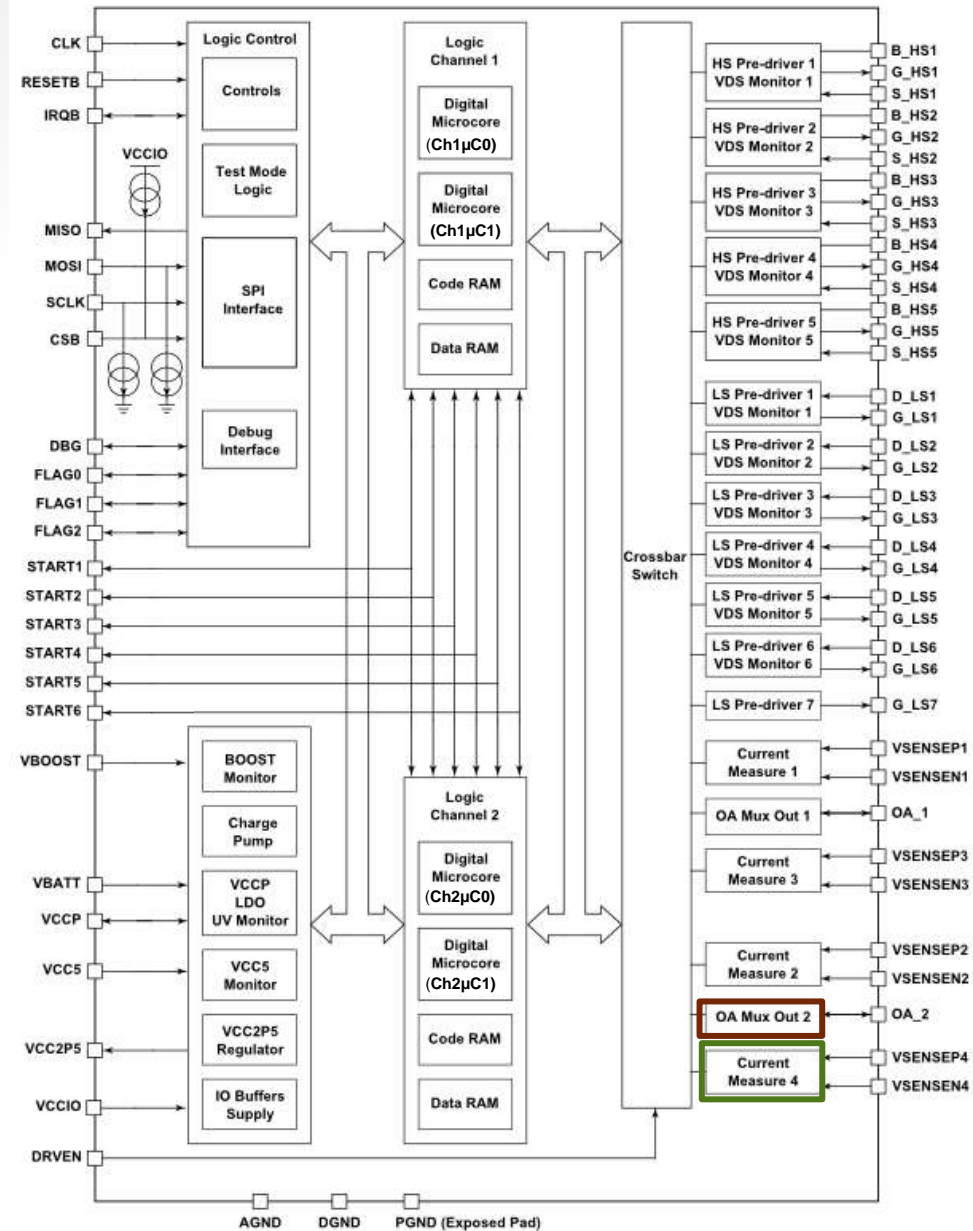
Differential amplifier

Ground referenced current sense resistor

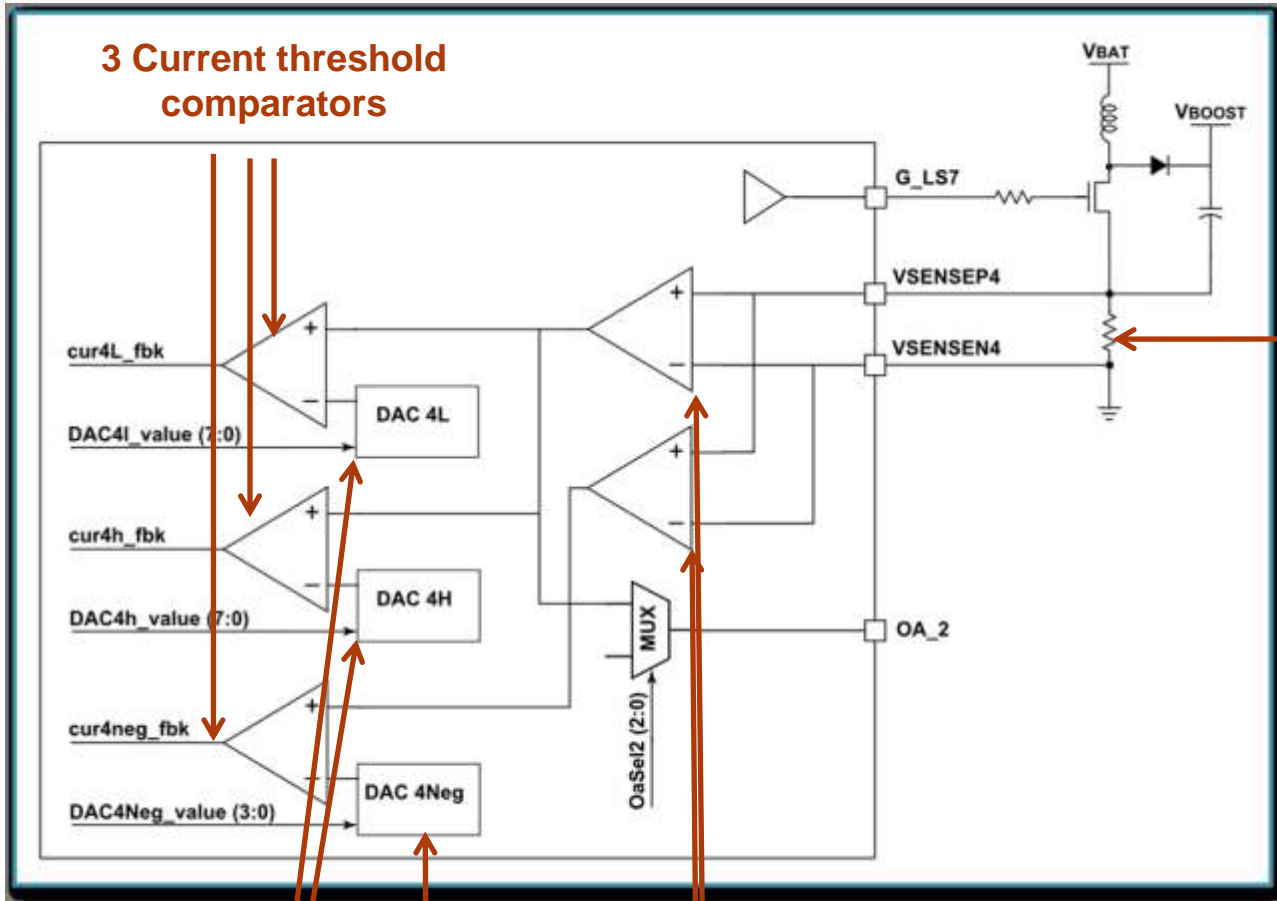
Analog MUX

# Current Sense 4 for DC/DC Converter

- 1 specific current measurement block – triple threshold current comparison for DC-DC conversion in current mode
- ‘Shunt resistor, connected to ground’ topology must be used
- 4 values of gain selected by microcode or SPI
- VSENSE differential voltage can be routed to OA\_2 pin only



# Current Sense Block 4 for DC/DC Converter



Ground referenced current sense resistor

8 bit DACs set high and low thresholds via SPI or  $\mu$ Core

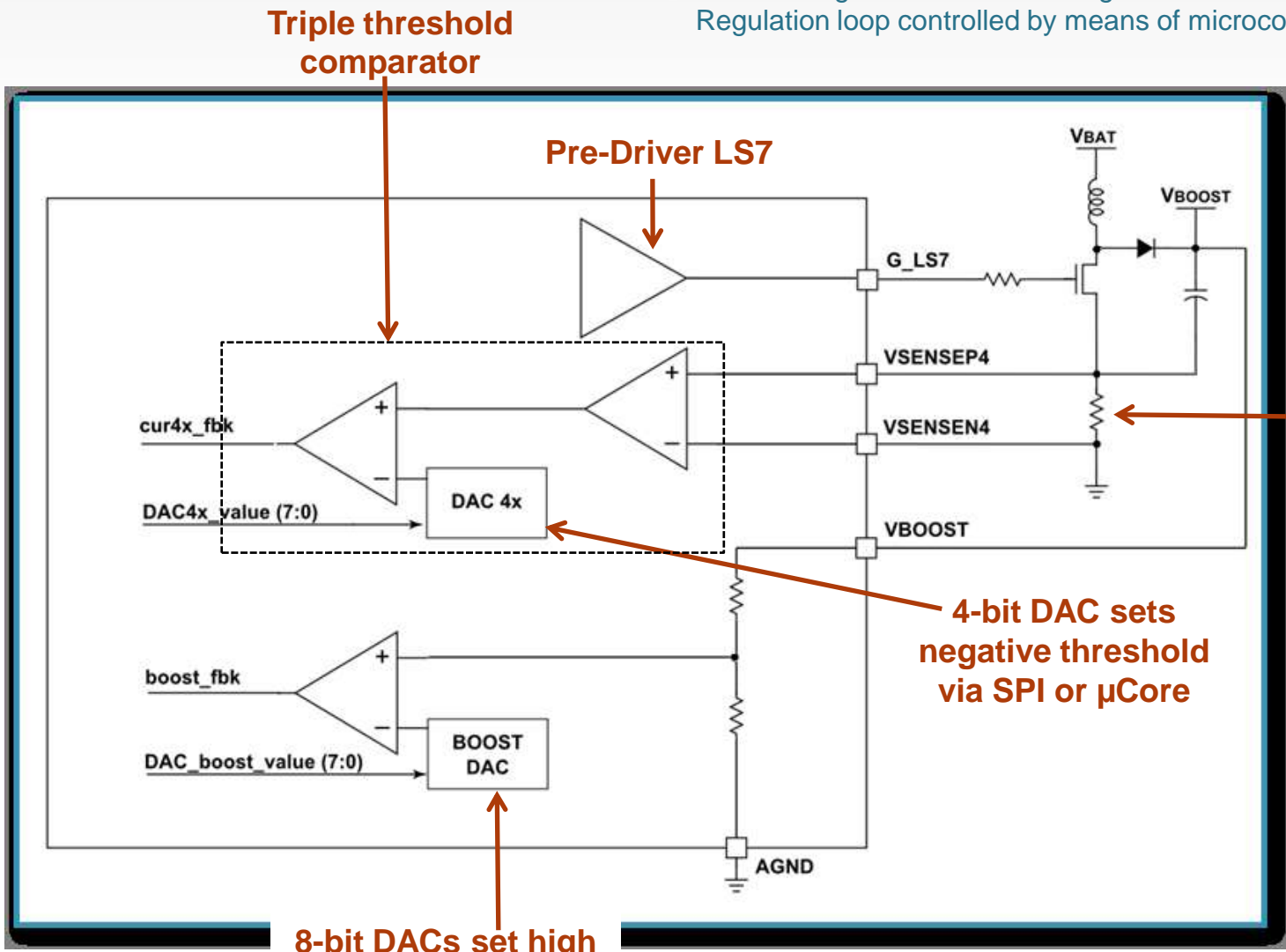
4 bit DAC sets negative threshold via SPI or  $\mu$ Core

Differential amplifiers



# DC/DC Converter Circuitry

LS7 dedicated to BOOST DC/DC Converter  
 Current measurement block 4 dedicated to BOOST converter  
 Boost voltage feedback with integrated divider  
 Regulation loop controlled by means of microcode



Triple threshold comparator

Pre-Driver LS7

Ground referenced current sense resistor

4-bit DAC sets negative threshold via SPI or µCore

8-bit DACs set high and low thresholds via SPI or µCore





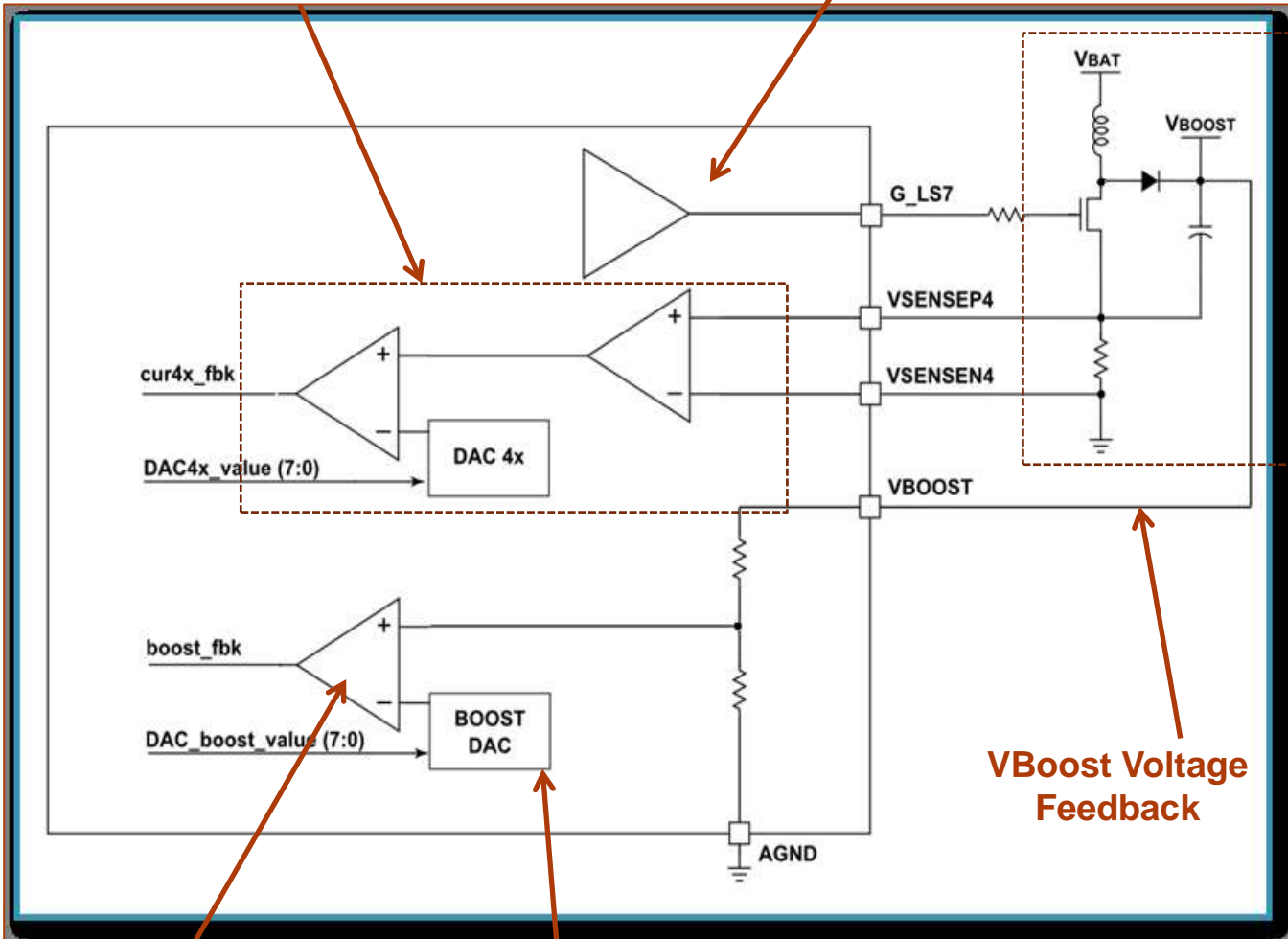


# DC/DC Converter Circuitry

Triple threshold current comparator

Pre-Driver LS7

External circuitry



VBoost Voltage Comparator

8-bit DAC sets VBOOST voltage via SPI or  $\mu$ Core

VBoost Voltage Feedback



# Additional MC33816 features

- Ground loss detection
  - Loss of connection between one or more ground pins and the system ground will result in a fault indication
- Code RAM memory BIST
- Contents of Code RAM is checked via a Check Sum for bit corruption
- Cipher encryption of Code RAM at download
  - Code RAM must be loaded with encrypted code
  - Code RAM contents read-out in encrypted form only
- External digital input-output (I/O) able to sustain up to 18V
  - Extra safety provided by higher maximum voltage rating on digital I/O pins
  - Normal external digital I/O on other products are only able to sustain 7.0 volts

# Flags

## MC33816 Internal Flag Bus:

- 16 internal digital flags available
- Certain flags are the bitwise-AND'd from each of the 4  $\mu$ Cores
- Default value is '0'
- Some flags routed to external pins, some are internal
- Used for Inter-processor and Intra-processor communication
- Flag definitions

Flag number (0-15)	Pin name assigned	Defined function type
0 to 2	FLAGx (x=1 to 3)	Digital Inputs/Outputs
3 to 8	STARTx (x=1 to 6)	Digital Inputs
9	IRQ (Interrupt Request)	Digital Output
10 and 11	OA_x (x=1, 2)	Analog Outputs
12	DBG (Debug)	Digital Output
13 to 15	No pin assigned (3 Internal flags)	

# Flags

- STARTx Flags (inputs to the MC33816)
  - For MCU to command the MC33816 to begin an injection cycle
  - Six STARTx inputs for up to six separate injectors
- FLAGx Flags (Inputs/Outputs to/from the MC33816)
  - Can be used by the MC33816 to tell the MCU an injection cycle is completed
  - Provide three signals between the MCU and the MC33816
- IRQB Flag (Interrupt Request from the MC33816)
  - Initiates MCU interrupt for time critical updates such as fault conditions
- OA Flags (Analog Outputs from Current measurement blocks)
  - Provides analog voltage to MCU A/D converter for resolution and accuracy
  - Two outputs mux'd to provide outputs from all 4 measurement blocks
- DBG Flag (One debug digital output from MC33816)
  - Provides indication of branches taken during program execution
- General Purpose Internal Flags (for inter-core communication)
  - Can provide three internal semaphores

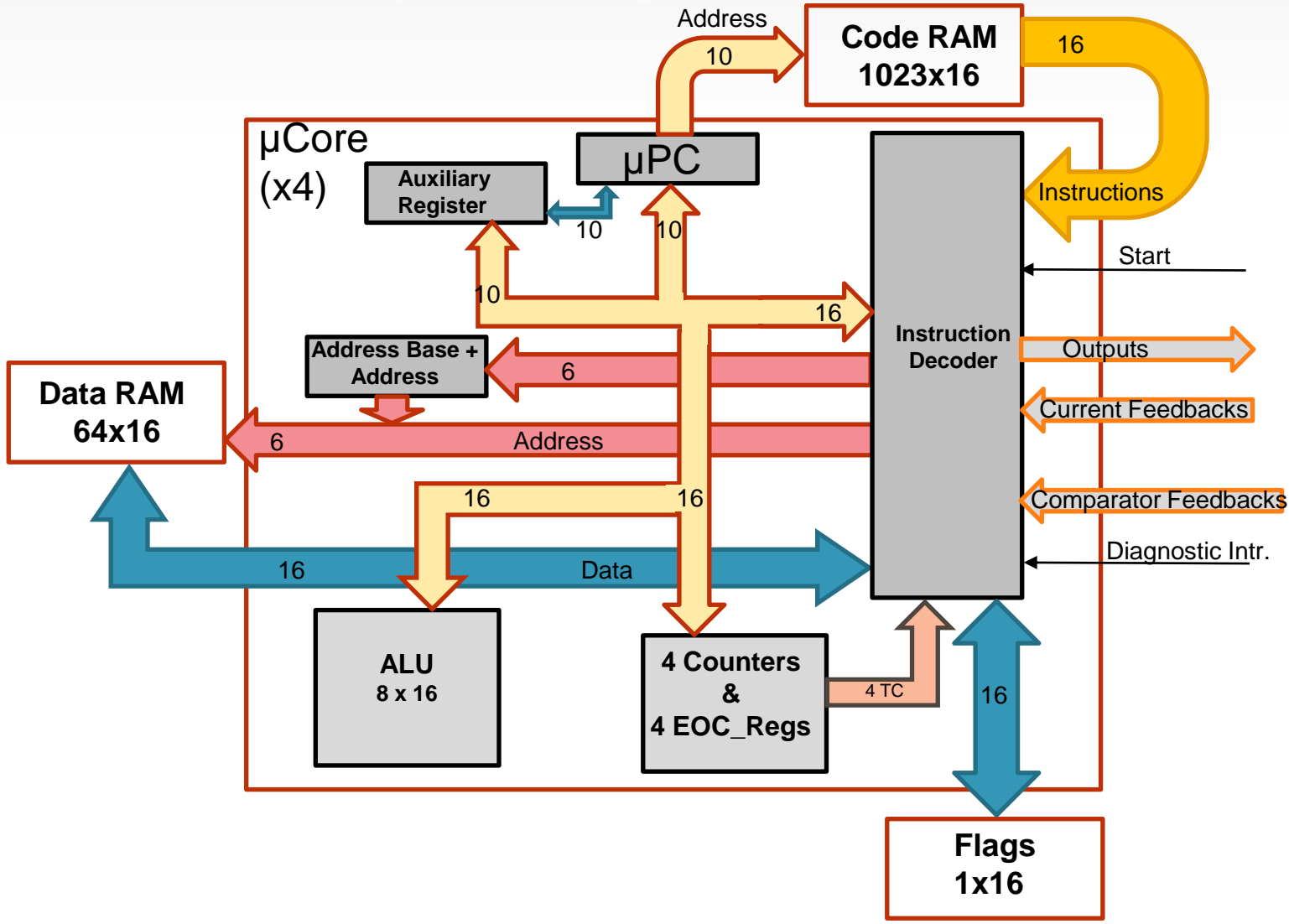


# Agenda

- Purpose for introducing the MC33816
- Overview of MC33816
  - Hardware – Circuit functions and features
  - Software – Instruction set and tools
- MC33816 Applications and Examples
  - 3, 4 and 6 cylinder examples
  - Peak and Hold waveform generation
  - Boost voltage regulator



# Software – Programming Model



# Software – Instruction Set Overview

- **93 instructions in 7 categories:**
  - **Arithmetic Logic Unit (27)**
  - **Configuration (29)**
  - **Diagnostic (3)**
  - **Interrupt and Subroutine (6)**
  - **Jump (16)**
  - **Load (9)**
  - **Wait (3)**

# 27 Arithmetic Logic Unit (ALU) Instructions

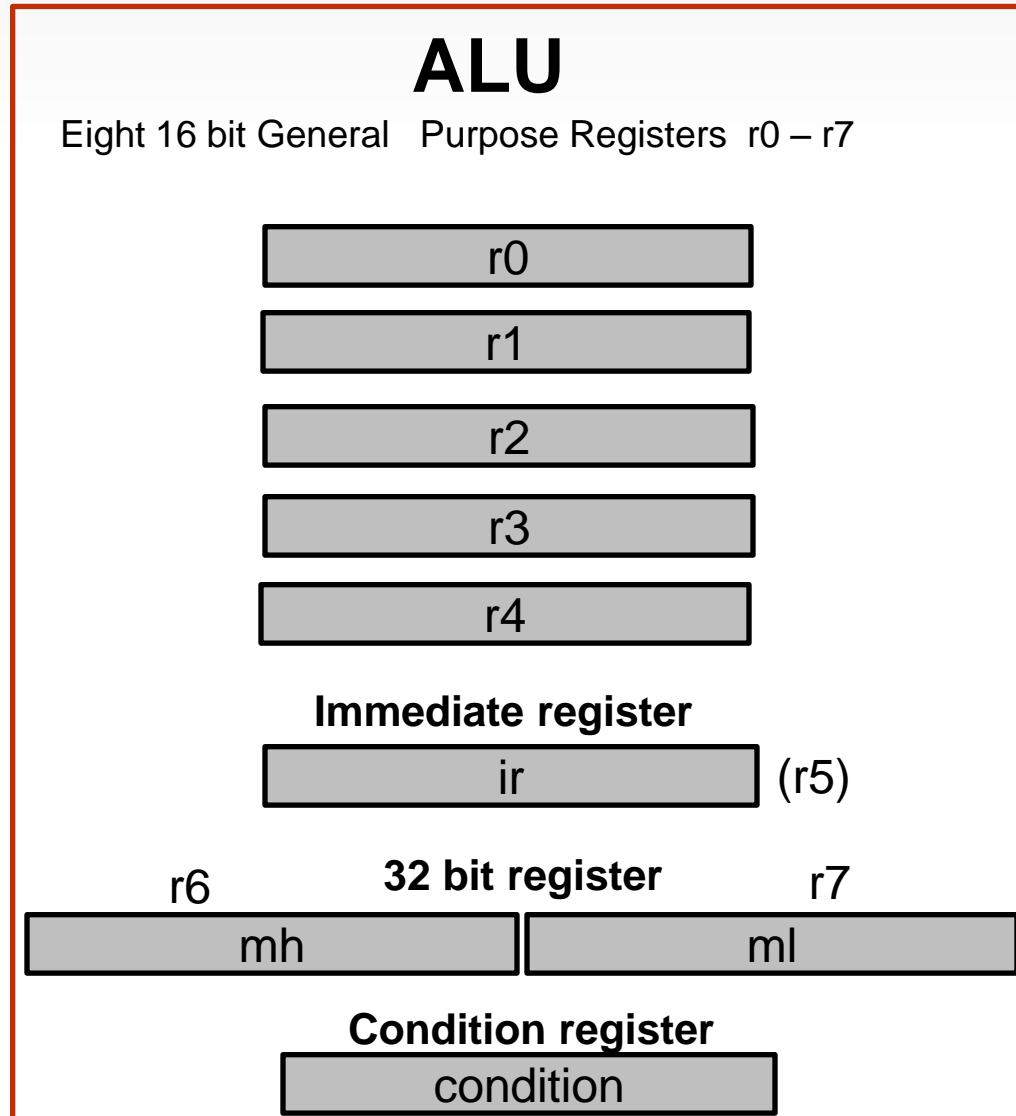
1. ***add*** – addition of two registers
2. ***addi*** – addition with immediate register
3. ***and*** – mask AND with immediate register
4. ***mul*** – multiplication of two registers
5. ***muli*** – multiplication with immediate value
6. ***not*** – invert contents of a register
7. ***or*** – mask OR with immediate register
8. ***sub*** – subtraction of two registers
9. ***subi*** – subtraction with immediate register
10. ***swap*** – swap high and low bytes of a register
11. ***toc2*** – performs 2's complement on register
12. ***toint*** – converts 2's complement to integer
13. ***xor*** – mask XOR with immediate register



## 27 ALU Instructions (con't)

14. ***sh32l*** – shift 32-bit register left (register)
15. ***sh32li*** – shift 32-bit register left immediate
16. ***sh32r*** – shift 32-bit register right (register)
17. ***sh32ri*** – shift 32-bit register right immediate
18. ***shl*** – shift register left (register)
19. ***shl8*** – shift register left 8 positions
20. ***shli*** – shift register left immediate
21. ***shls*** – shift register left signed (register)
22. ***shlsi*** – shift register left signed immediate
23. ***shr*** – shift register right (register)
24. ***shr8*** – shift register right 8 positions
25. ***shri*** – shift register right immediate
26. ***shrs*** – shift register right signed (register)
27. ***shrsi*** – shift register right signed immediate

# ALU – Programming Model



## ALU: General Purpose Registers ( $r0 - r4$ )

The ALU general purpose registers ( $r0 - r4$ ) can be used as:

- Source (operands) for arithmetic and logical operations
- Destination (results) for arithmetic and logical operations

## ALU: 32-bit Register $m$ ( $m_h = m$ high & $m_l = m$ low) also known as ( $r6$ and $r7$ )

The ALU 32-bit register( $m_h, m_l$ ) can be used as:

- Two general purpose registers ( $r6$  and  $r7$ )
- One 32-bit register used for 32-bit shift instructions
- One 32-bit register used in multiplication instructions

## ALU: Immediate Register (*ir*)

also known as (*r5*)

The ALU immediate register (*ir*) can be used as:

- General purpose register
- SPI address for the SPI backdoor operations
- Offset to modify the address towards the Data Ram.
- Mask value for logic operations (AND, OR, XOR)
- Constant value

## ALU: Immediate Value (*imm*)

The ALU immediate value is a 4-bit (0 – 15) number that can be used as:

- A factor in a multiplication
- An operand in other ALU instructions

# ALU: 16-bit Condition Register

BIT	NAME	DESCRIPTION
15	SHIFT_OUT	Shifted out bit
14	CONV_SIGN	Last conversion sign
13	CARRY	Carry over bit
12-11	ARITH_LOGIC	Arithmetic logic
10	MASK_MIN	Mask result 0x0000
9	MASK_MAX	Mask result 0xFFFF
8	MUL_SHIFT_OVR	Multiplication shift overflow
7	MUL_SHIFT_LOSS	Multiplication shift precision loss
6	RES_ZERO	Addition or subtraction result is zero
5	RES_SIGN	Addition or subtraction sign result
4	UNSIGNED_UND	Unsigned underflow
3	UNSIGNED_OVR	Unsigned overflow
2	SIGNED_UND	Signed underflow
1	SIGNED_OVR	Signed overflow
0	OP_DONE	Operation complete

## 29 Configuration Instructions

1. ***bias*** – enables/disables a single bias structure
2. ***chth*** – changes threshold on feedback comparator
3. ***dfcsct*** – define shortcut for current feedback
4. ***dfsct*** – define shortcut for outputs
5. ***rdspi*** – request SPI read
6. ***rstreg*** – reset register
7. ***rstsl*** – reset start latch register
8. ***slab*** – select address base
9. ***slfbk*** – select feedback source
10. ***slsa*** – selects which register to use as SPI address
11. ***stab*** – load value in address base register
12. ***stadc*** – enables/disables ADC conversion on specified current measurement block

## 29 Configuration Instructions (con't)

13. ***stal*** – set ALU mode
14. ***stcrb*** – set control register bit
15. ***stcrt*** – set channel communication register
16. ***stdcct*** – set DC/DC control mode
17. ***stdm*** – set DAC register access mode
18. ***stdrm*** – set DRAM read mode
19. ***steoa*** – enable end of actuation mode
20. ***stf*** – set flag
21. ***stfw*** – set freewheeling mode
22. ***stgn*** – set Op Amp gain
23. ***stirq*** – set interrupt request output pin
24. ***sto*** – set single output
25. ***stoc*** – set offset compensation
26. ***stos*** – set output shortcut

## 29 Configuration Instructions (con't)

- 27. ***stslew*** – set slew rate
- 28. ***stsrbit*** – set status register bit
- 29. ***wrspi*** – request SPI write (backdoor)



## 3 Diagnostic Instructions

1. ***endiag*** – enable diagnostics (single)
2. ***endiaga*** – enable diagnostics (all)
3. ***endiags*** – enable diagnostic shortcut

## 3 Interrupt Instructions

1. ***iconf*** – interrupt configuration
2. ***iret*** – return from interrupt
3. ***reqi*** – request for software interrupt

## 3 Subroutine Instructions

1. ***jtsf*** – jump to subroutine far
2. ***jtsr*** – jump to subroutine relative
3. ***rfs*** – return from subroutine

# 16 Jump Instructions

1. ***jarf*** – jump on **arith**metic register **far**
2. ***jarr*** – jump on **arith**metic register **relative**
3. ***jcrf*** – jump on **control** register **far**
4. ***jcrr*** – jump on **control** register **relative**
5. ***jfbkf*** – jump on **feedback** **far**
6. ***jfbkr*** – jump on **feedback** **relative**
7. ***jmpf*** – unconditional **jump** **far**
8. ***jmpr*** – unconditional **jump** **relative**
9. ***jocf*** – jump **on** flag **condition** **far**
10. ***jocr*** – jump **on** flag **condition** **relative**
11. ***joidf*** – jump **on**  $\mu$ core **id** **far**
12. ***joidr*** – jump **on**  $\mu$ core **id** **relative**
13. ***joslf*** – jump **on** **start-latch** **far**

## 16 Jump Instructions (con't)

14. ***joslr*** – jump on **start-latch** relative
15. ***jsrf*** – jump on **status register far**
16. ***jsrr***– jump on **status register relative**

## 9 Load Instructions

1. ***cp*** – copy source to destination register
2. ***ldca*** – load counter from register and set outputs
3. ***ldcd*** – load counter from DRAM and set outputs
4. ***ldirh*** – load immediate register high-byte
5. ***ldirl*** – load immediate register low-byte
6. ***ldjr1*** – load jump register 1
7. ***ldjr2*** – load jump register 2
8. ***load*** – load data from DRAM to register
9. ***store*** – store data from register to DRAM

## 3 Wait Instructions

1. ***cwef*** – create wait table entry far
2. ***cwer*** – create wait table entry relative
3. ***wait*** – wait until a condition is verified

# Tools

1. MC33816 Data sheet for hardware reference
2. MC33816 Software programming reference manual
3. MC33816 EVB with 4 Cylinder and DC/DC software
4. Assembler with built-in encryptor
5. SPIGen register setting with software download capability
6. Software simulator with debug capability\*
7. Execution tracing module and software to use the dbg flag pin.\*

\*coming soon

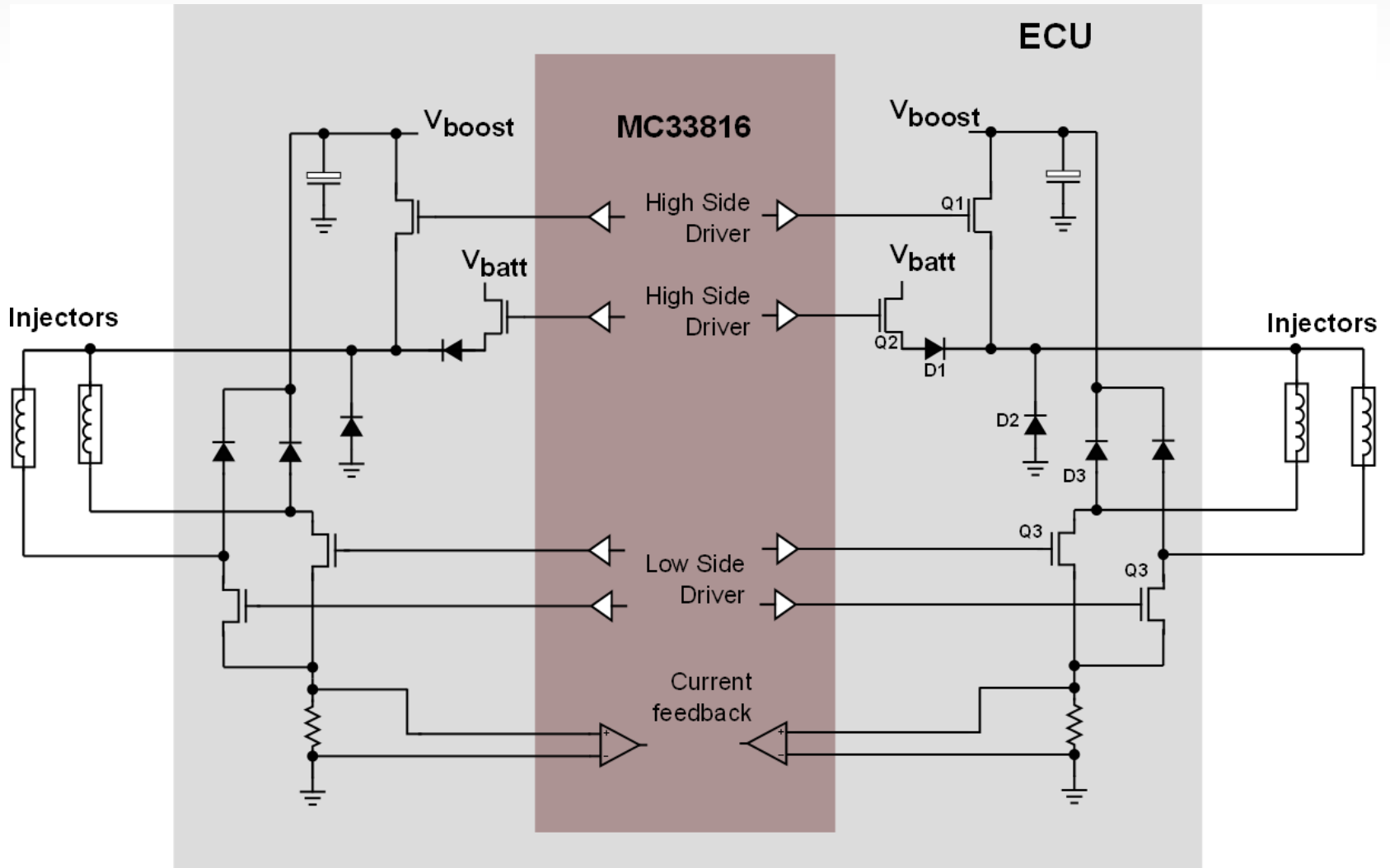


# Agenda

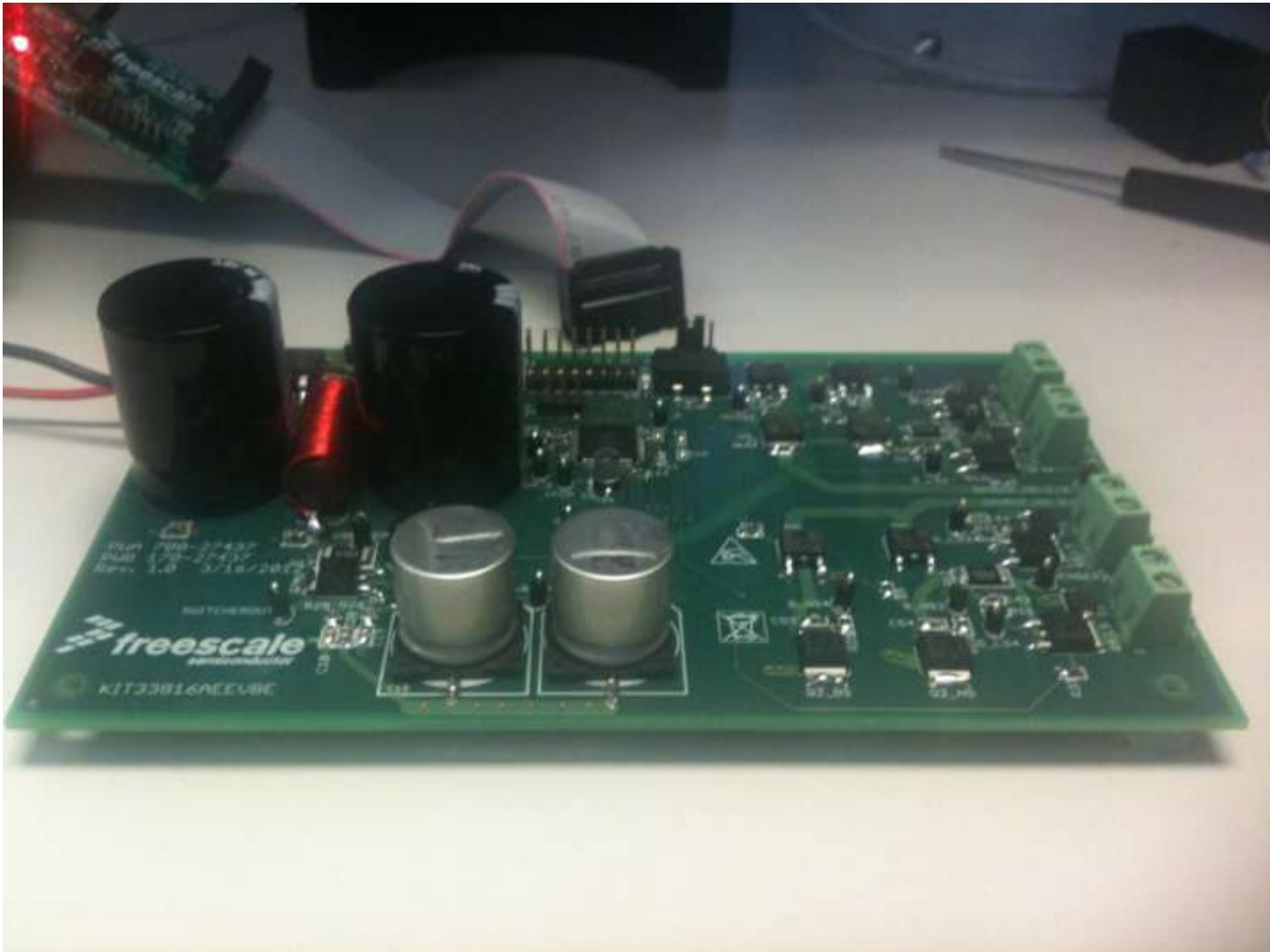
- Purpose for introducing the MC33816
- Overview of MC33816
  - Hardware – Circuit functions and features
  - Software – Instruction set and tools
- MC33816 Applications and Examples
  - 4 cylinder example
  - Peak and Hold waveform generation
  - Boost voltage regulator



# 4 cylinder configuration, 4 injectors, 2 banks

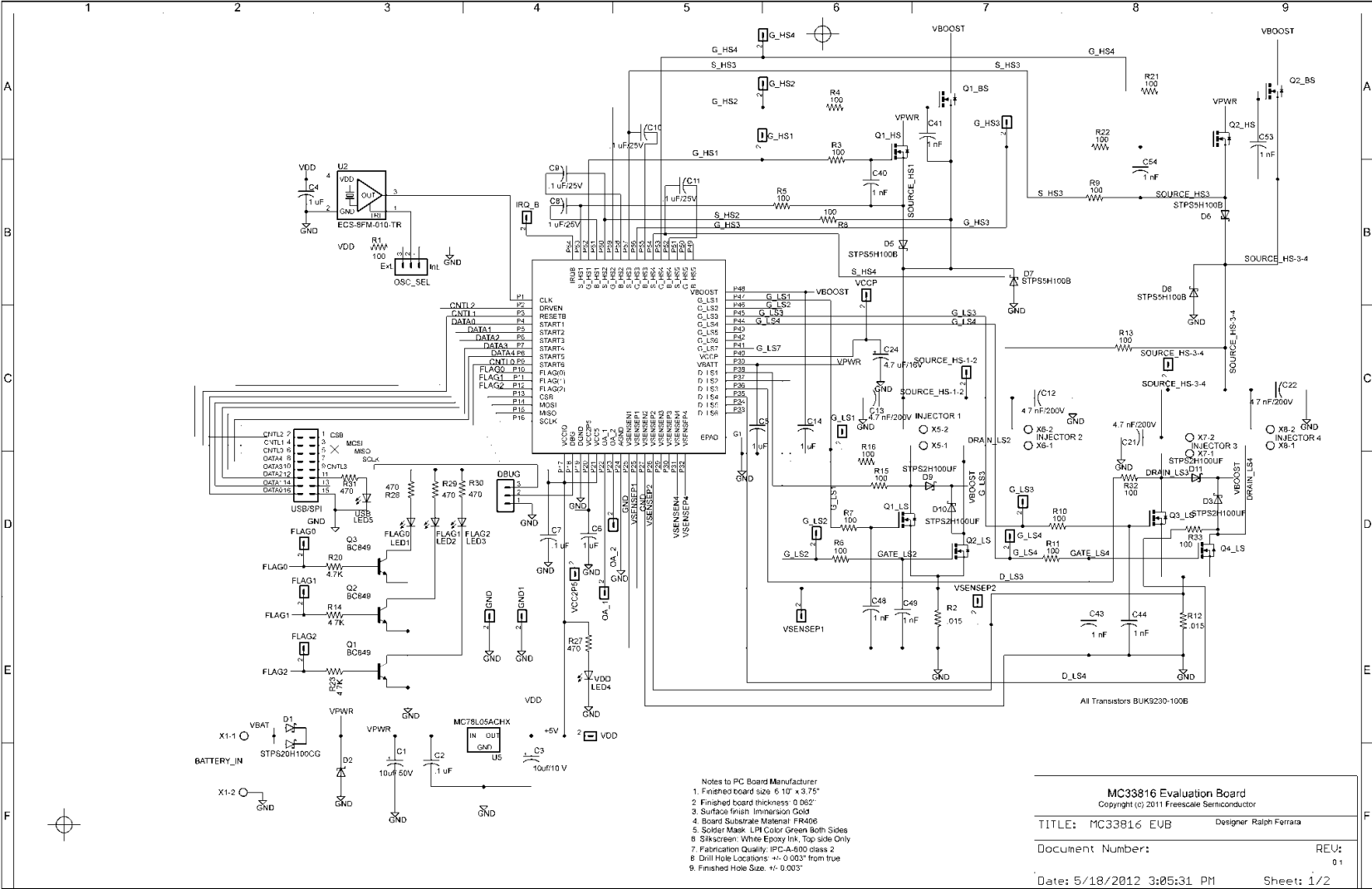


# MC33816 Evaluation Board (EVB)





# MC33816 Evaluation Board (EVB) Schematic 1 of 2

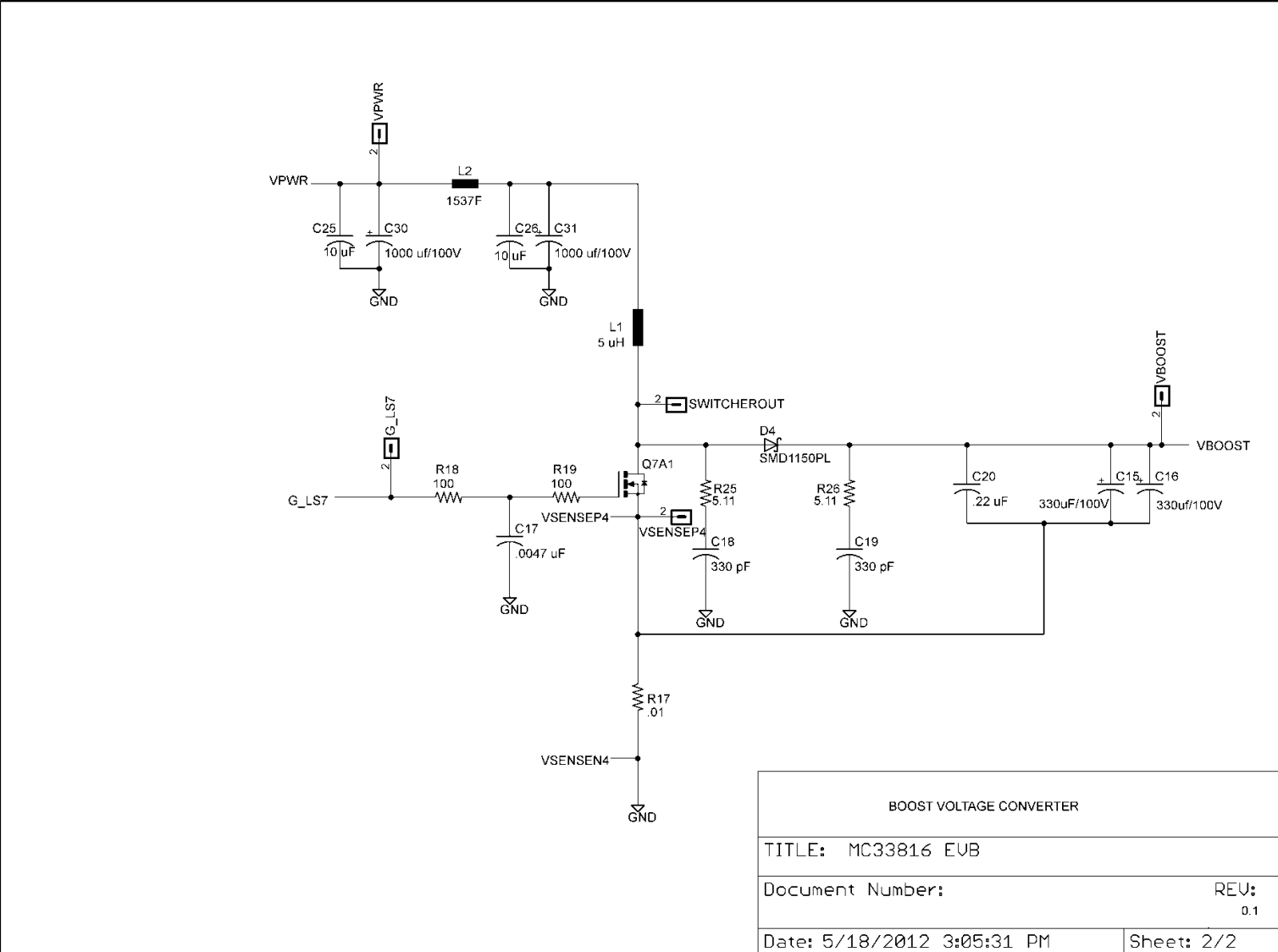


- Notes to PC Board Manufacturer
1. Finished board size 6.10" x 3.75"
  2. Finished board thickness 0.062"
  3. Surface finish Immersion Gold
  4. Board Substrate Material FR406
  5. Solder Mask LPI Color Green Both Sides
  6. SilkScreen White Epoxy Ink, Top side Only
  7. Fabrication Quality: IPC-A-600 class 2
  8. Drill Hole Locations: +/- 0.003" from true
  9. Finished Hole Size: +/- 0.003"

<b>MC33816 Evaluation Board</b>	
Copyright (c) 2011 Freescale Semiconductor	
TITLE: MC33816 EVB	Designer: Ralph Ferrara
Document Number:	REV: 01
Date: 5/18/2012 3:05:31 PM	Sheet: 1/2



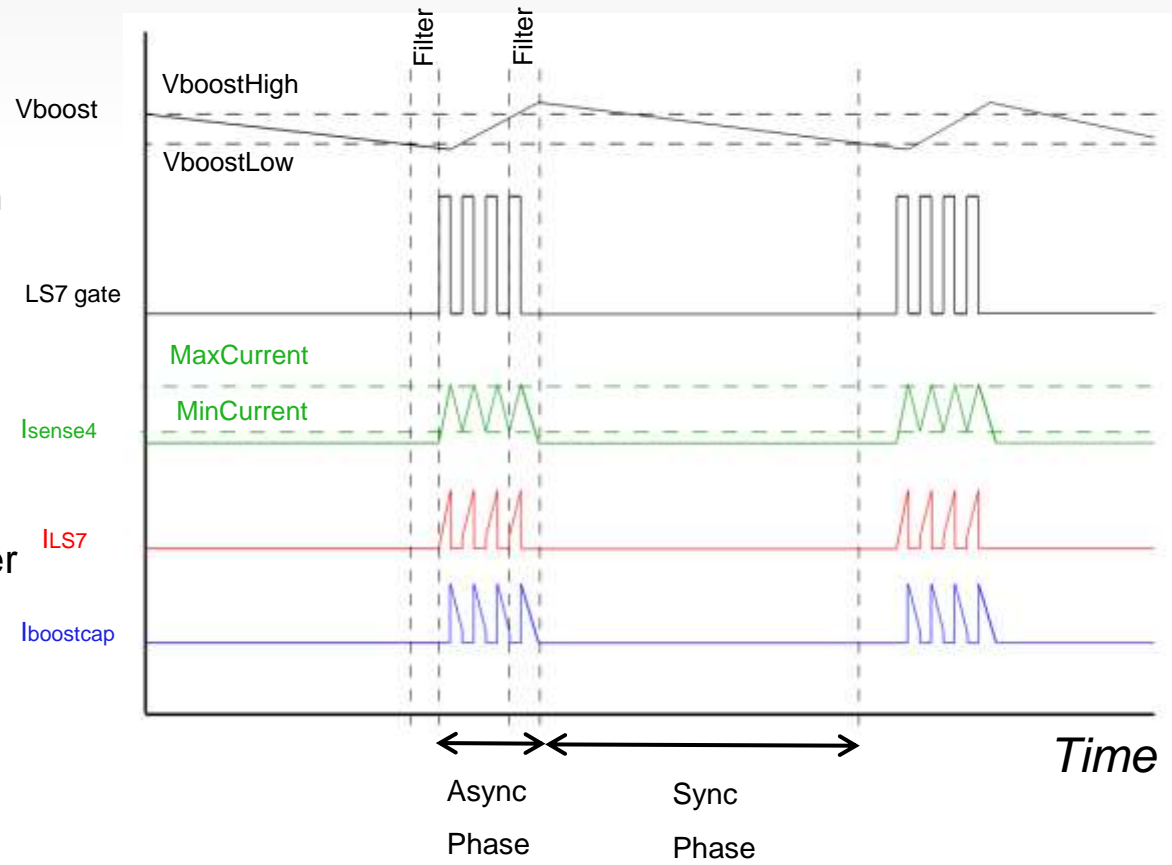
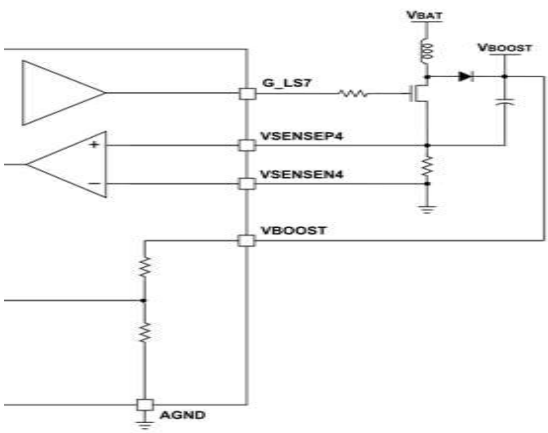
# MC33816 Evaluation Board (EVB) Schematic 2 of 2



BOOST VOLTAGE CONVERTER	
TITLE: MC33816 EUB	
Document Number:	REV: 0.1
Date: 5/18/2012 3:05:31 PM	Sheet: 2/2

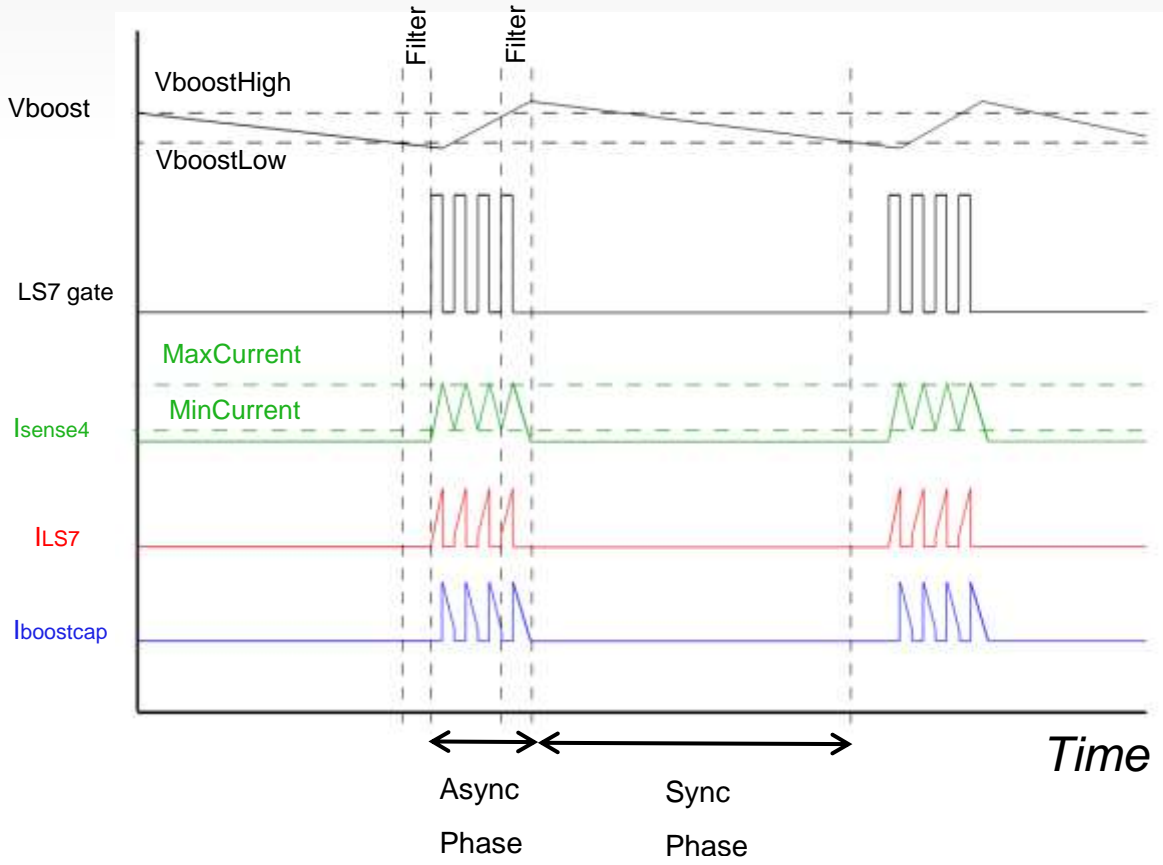
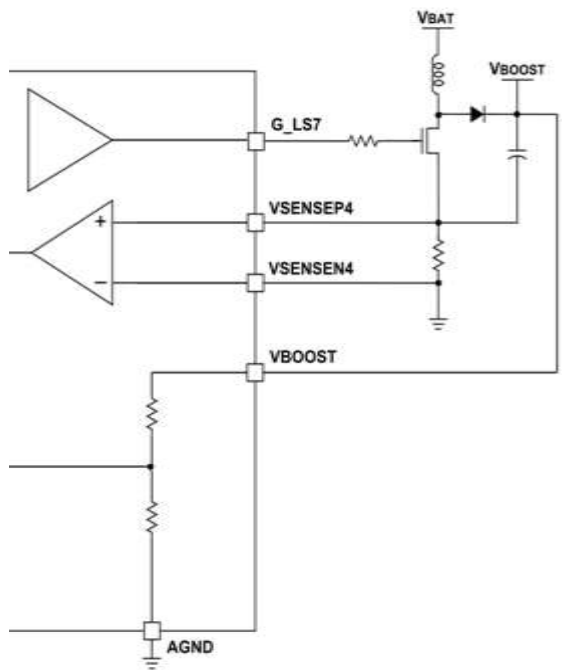
# DC/DC Converter Parameters Definition

- Hysteretic conversion mode: variable frequency
- Async Phase: current automatic regulation between MaxCurrent and MinCurrent up to  $V_{boost} > V_{boostHigh}$
- Sync Phase: MOSFET off up to  $V_{boost} < V_{boostLow}$
- MaxCurrent must be lower than the inductor saturation current
- Filter (usTime) is used to filter  $V_{boost}$  feedback



# DC/DC Converter Parameters Definition (con't)

- Initialization phase:
  - Set current threshold
  - Set boost voltage condition



```

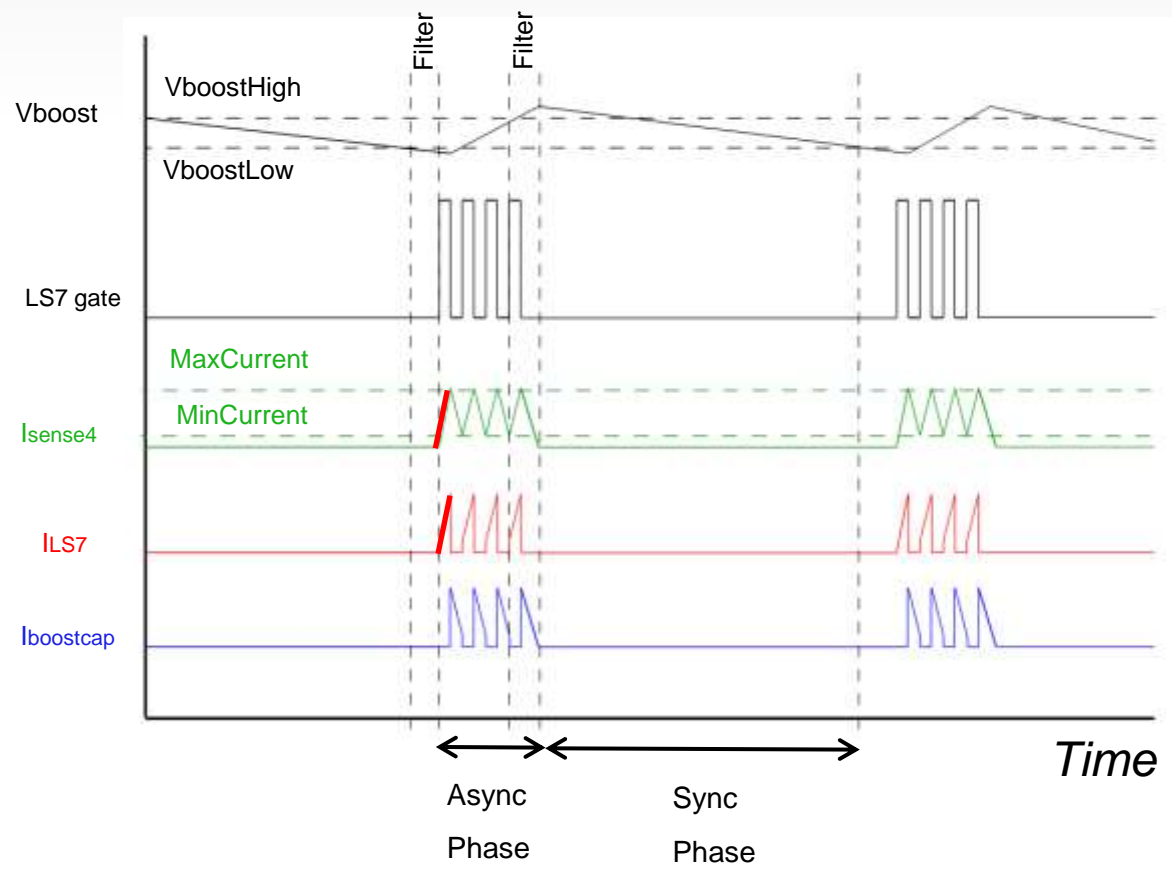
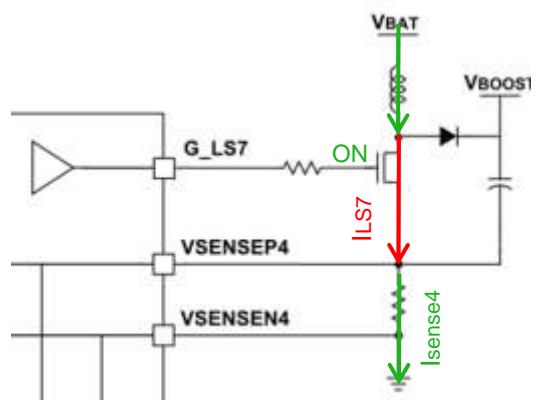
stdm dac;
load MinCurrent dac_osscc_ofs;
load MaxCurrent dac4h4n_ofs;
stdm null;

cwer dcdc_on _vb row1;
cwer dcdc_off vb row2;
    
```

- \* Select dac\_access\_mode
- \* Load the min current threshold in DAC 4L
- \* Load the max current threshold in DAC 4H
- \* Select dac\_boost\_access\_mode
- \* Wait table entry for under threshold condition
- \* Wait table entry for over threshold condition

# DC/DC Converter: Async Phase

- LS7 is ON until  $I_{sense4} > MaxCurrent$
- Current flowing through the inductor and the LS7 MOSFET
- Async phase up to  $V_{boost} > V_{boostHigh}$



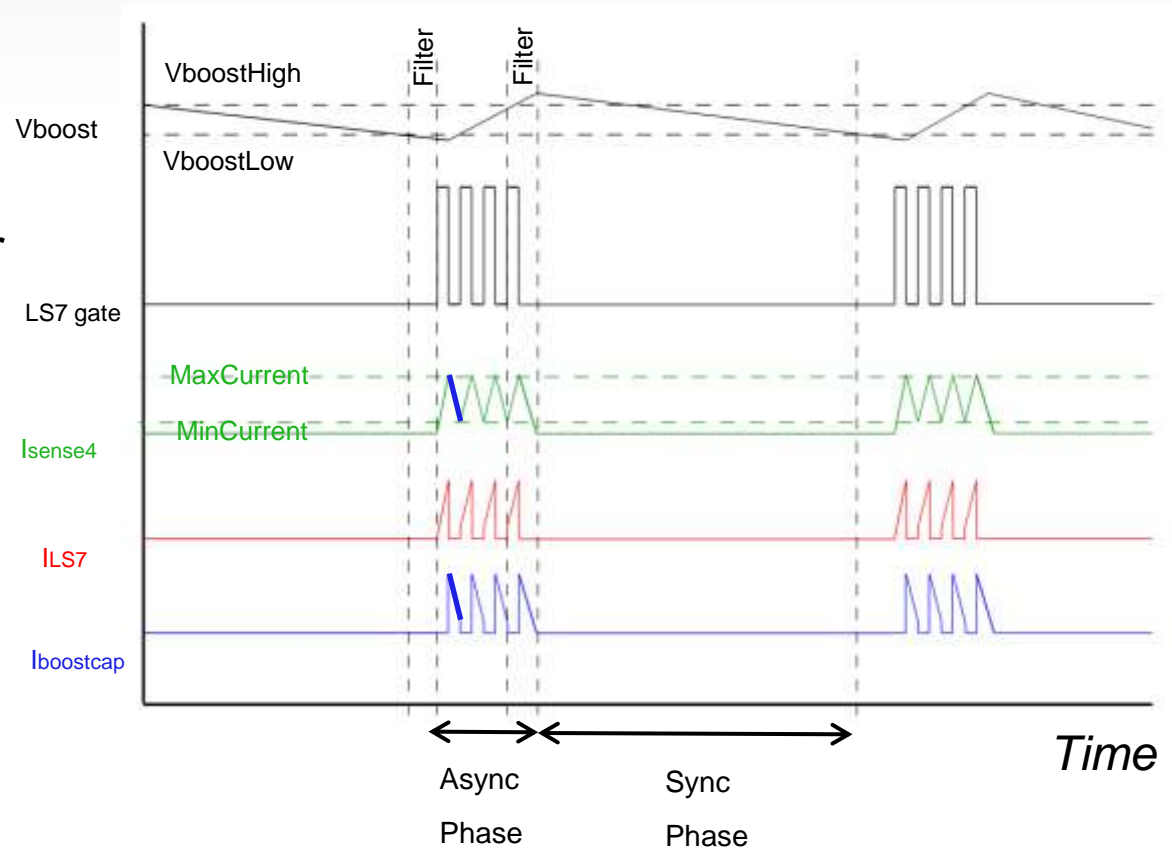
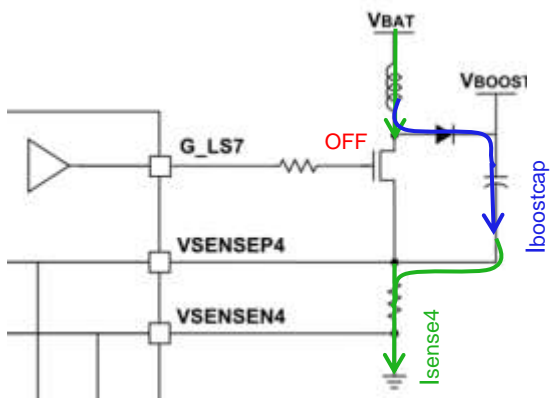
```

dcdc_on:    load VboostHigh dac4h4n_ofs;
            jtsr Wait_us;
            stdctl async;
            wait row2;
    
```

- \* Load the upper Vboost threshold in vboost\_dac register
- \* Wait here, mandatory due to long filter on boost voltage
- \* Enable HW PWM
- \* Wait for vboost over threshold condition

# DC/DC Converter: Async Phase (con't)

- LS7 is OFF until  $I_{sense4} < MinCurrent$
- Current flowing through the diode and charging the capacitor
- Async phase up to  $V_{boost} > V_{boostHigh}$



```

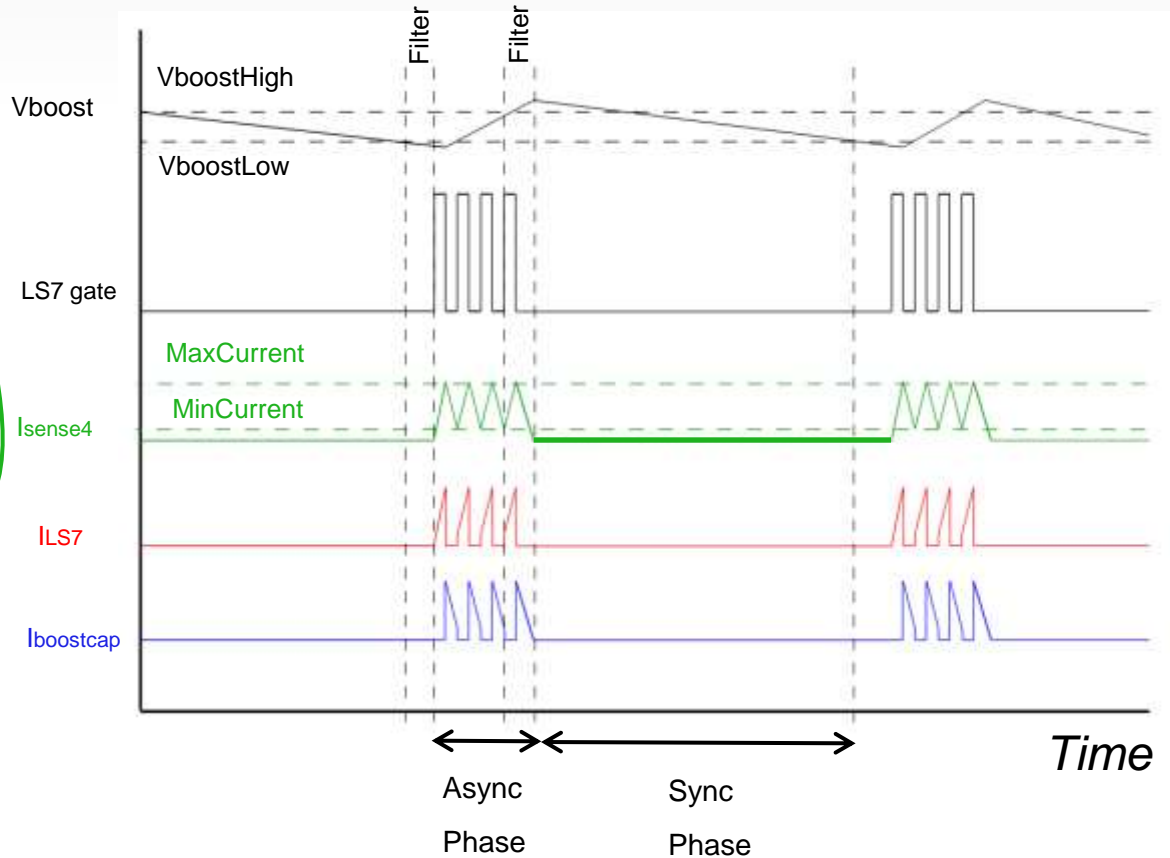
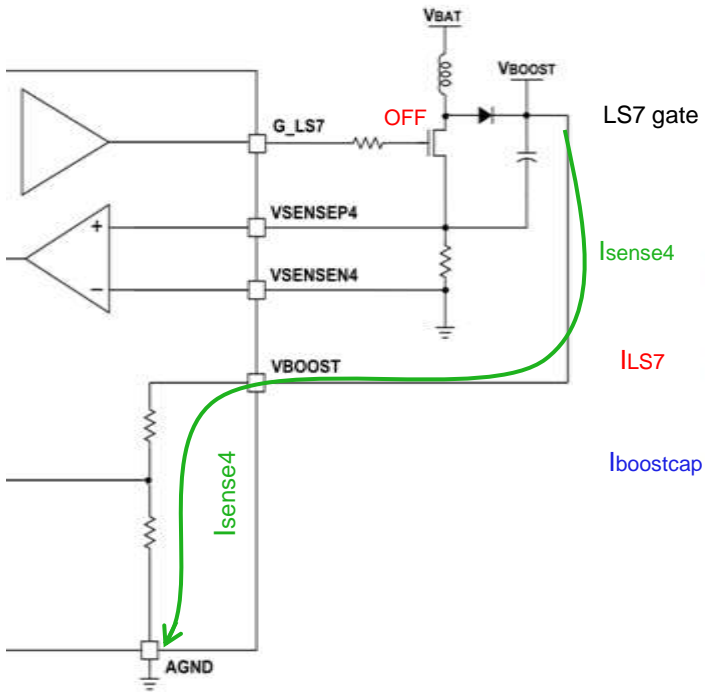
dcdc_on:    load VboostHigh dac4h4n _ofs;
            jtsr Wait_us;
            stdctl async;
            wait row2;
    
```

- \* Load the upper Vboost threshold in vboost\_dac register
- \* Wait here, mandatory due to long filter on boost voltage
- \* Enable HW PWM
- \* Wait for vboost over threshold condition



# DC/DC Converter: Sync Phase

- LS7 is OFF until  $V_{boost} < V_{boostLow}$

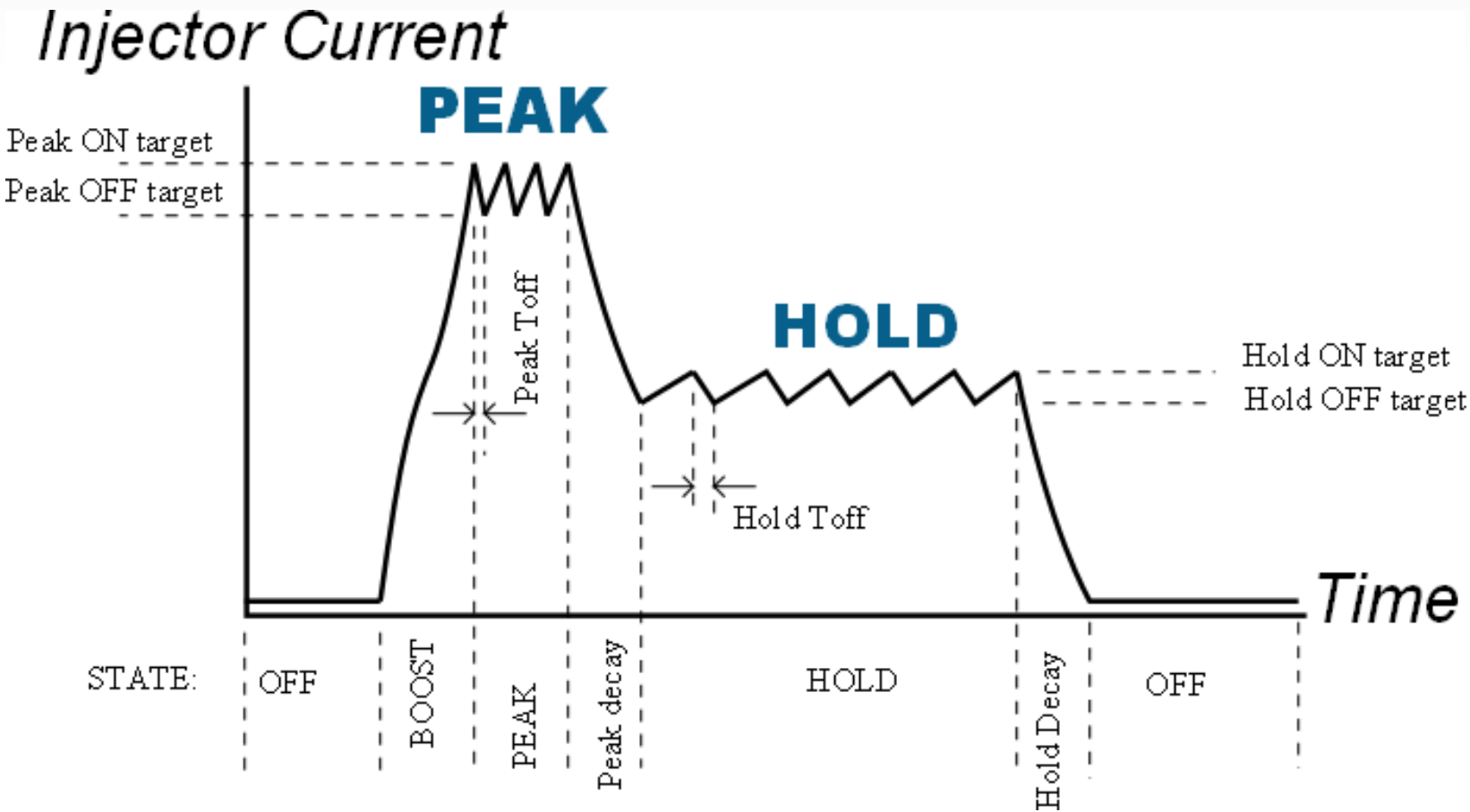


```

dcdc_off:  load VboostLow dac4h4n_ofs;
           jtsr Wait_us;
           stdctl sync;
           wait row1;
    
```

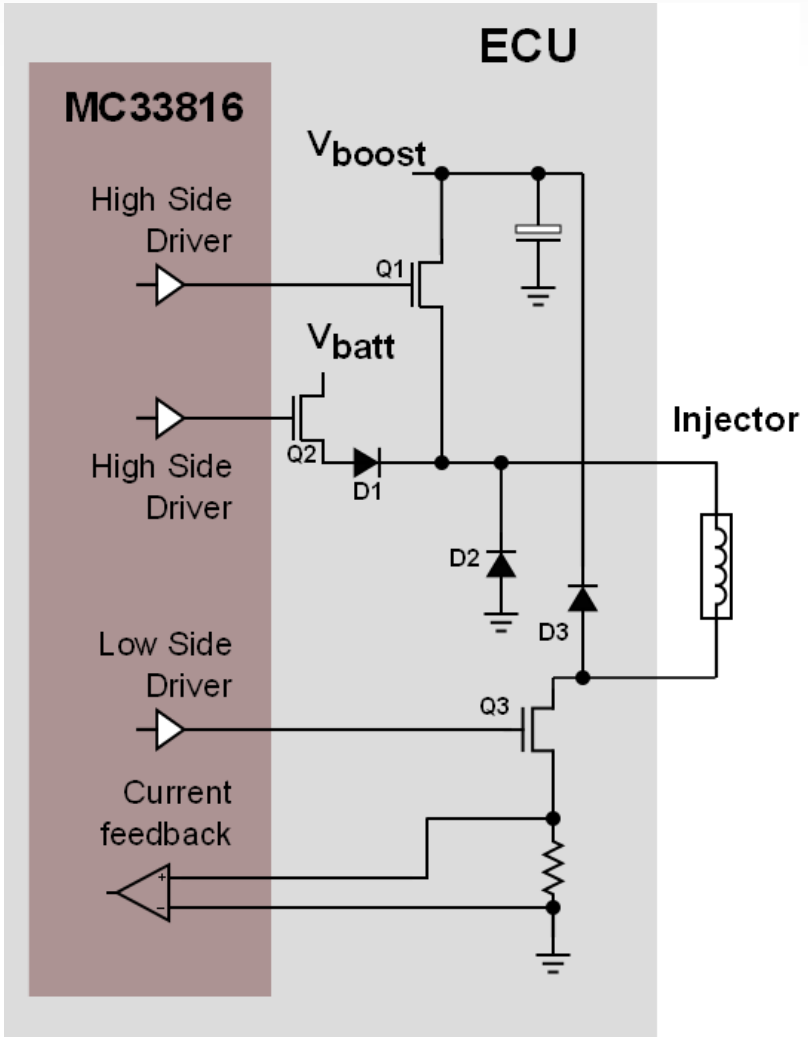
- \* Load the upper Vboost threshold in vboost\_dac register
- \* Wait here, mandatory due to long filter on boost voltage
- \* Disable HW PWM
- \* Wait for vboost under threshold condition

# Injector Current parameter definition

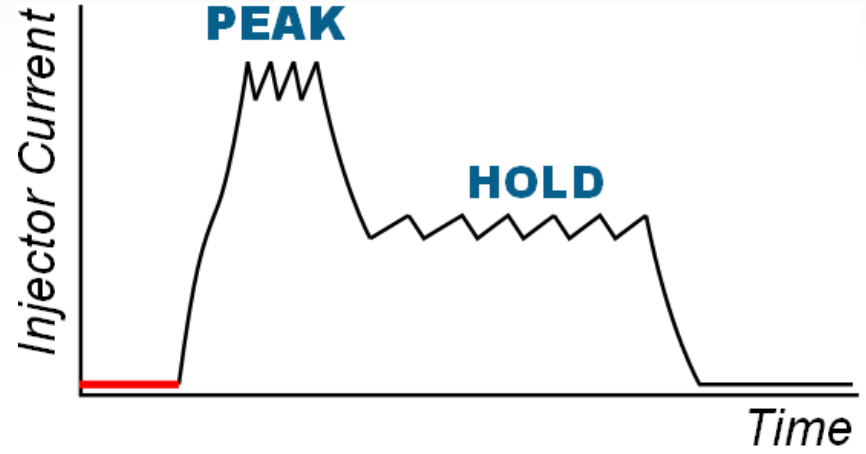
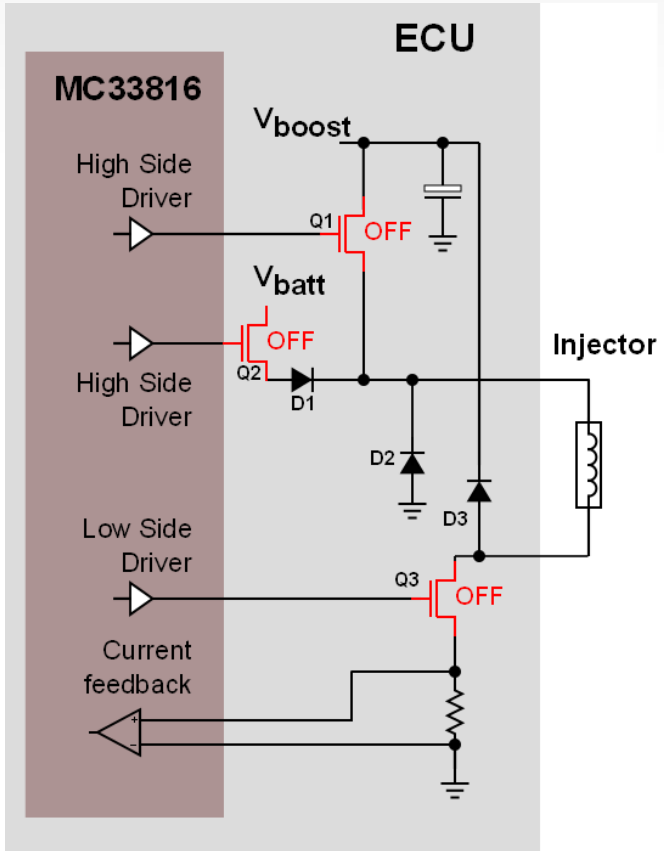


# Boosted, banked, direct injector architecture

- One injector shown.
- Common (banked) features are high side supplies and current feedback
- Individual cylinders get individual low side (LS) FETs

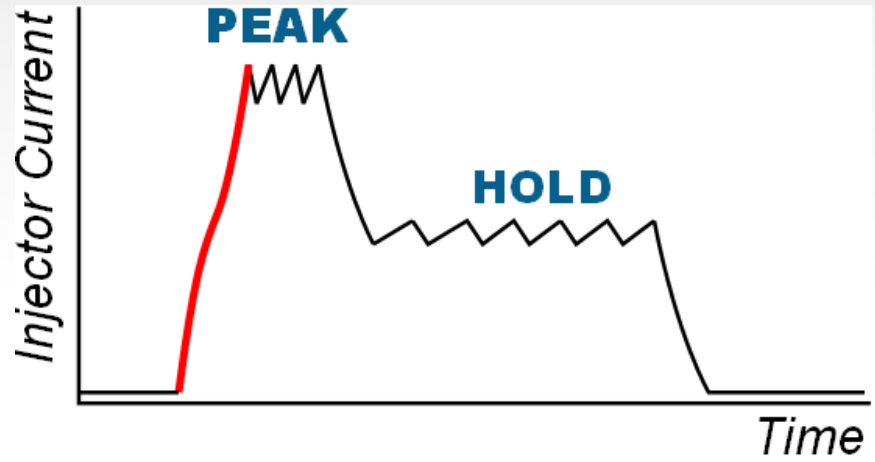
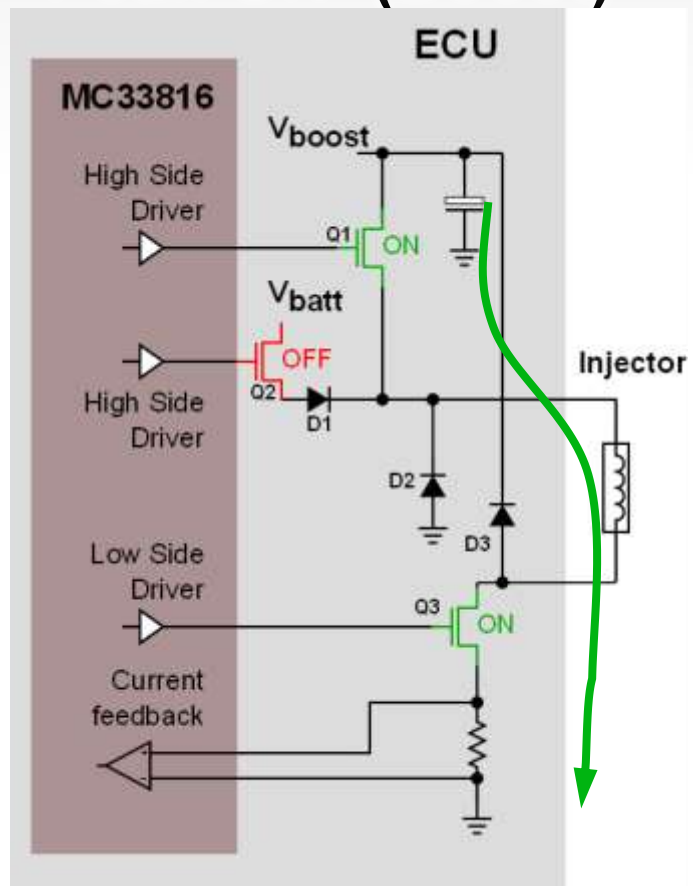


# Peak and hold sequencing: OFF state



- All FETs off
- No current flow

# Peak ON (Boost)



- Bank 1 Boost ON
- Inj1 HS Bat OFF
- Inj1 Low side ON
- Vbat blocked by diode D1
- Current rises to peak target

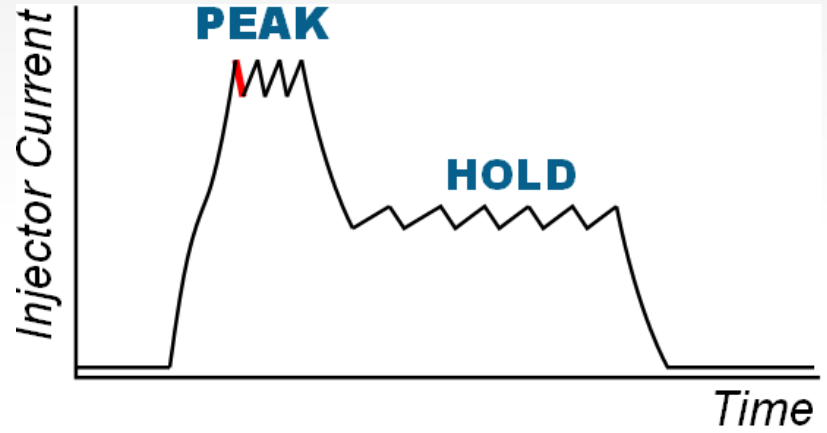
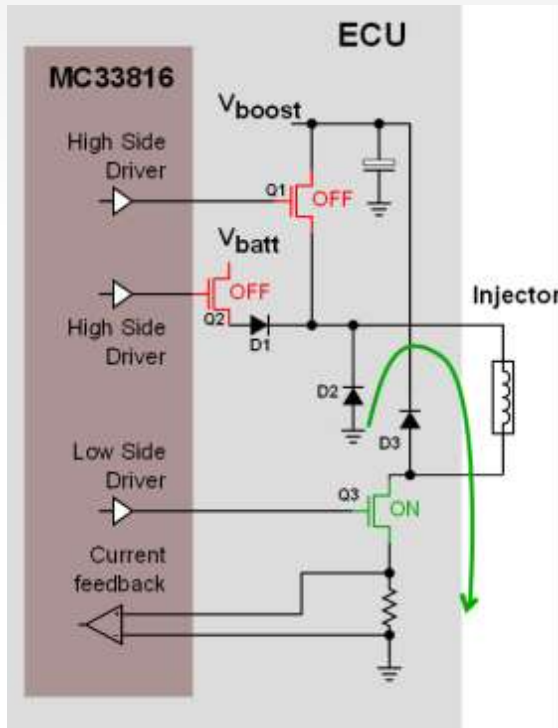
```

### launch phase enable boost ###
boost0:    load I_boost dac_sssc_ofs;
           cwer peak0 ocur_row2;
           cwef jr1_start row5;
           stos off on on;
           wait row25;
    
```

- \* Load the boost phase current threshold
- \* Jump to peak phase when current is over threshold
- \* If the start goes low, go to eoinj phase
- \* Turn VBAT off, BOOST on, LS on
- \* Wait for one of the 2 previously defined conditions

# Peak OFF

- Bank 1 Boost OFF
- Inj1 HS Bat OFF
- Inj1 Low side ON
- Current recirculates through D2 and decays
- Peak phase is combination of Peak ON and Peak OFF



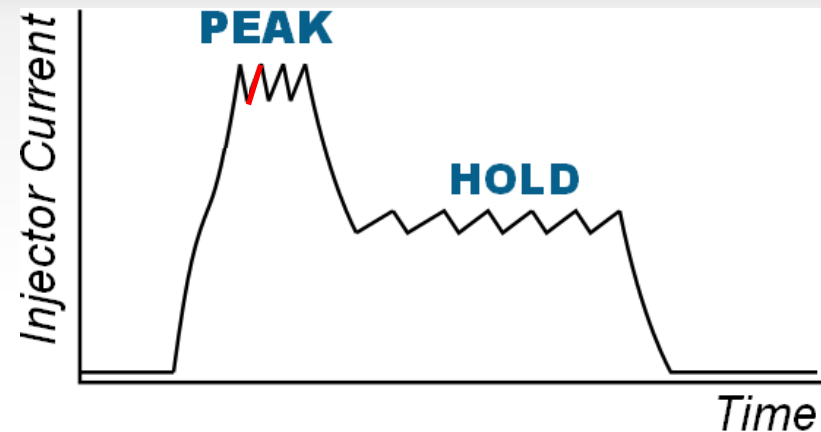
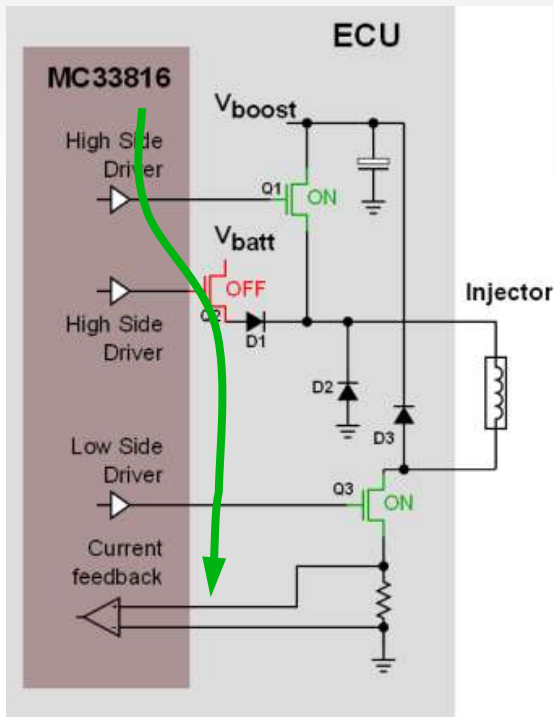
```

#### peak phase continue on boost ####
peak0:      ldcd rst_ofs keep keep Tpeak_tot c1; * Load the length of the total peak phase in counter 1
            cwer peak_on0 tc2 row1;           * Jump to peak_on when tc2 reaches end of count
            cwer peak_off0 ocur row2;        * Jump to peak_off when current is over threshold
            cwer off_phase0 tc1 row3;        * Jump to off_phase when tc1 reaches end of count
            load I_peak dac_sssc_ofs;        * Load the peak current threshold

peak_on0:   stos off on on;                  * Turn VBAT off, BOOST on, LS on
            wait row235;                    * Wait for one of the previously defined conditions

peak_off0:  ldcd rst_ofs keep keep Tpeak_off c2; * Load in the counter 2 the length of the peak_off phase
            stos off off on;                * Turn VBAT off, BOOST off, LS on
            wait row135;                    * Wait for one of the previously defined conditions
    
```

# NXP Peak ON



- Bank 1 Boost OFF
- Inj1 HS Bat OFF
- Inj1 Low side ON
- Current recirculates through D2 and decays
- Peak phase is combination of Peak ON and Peak OFF

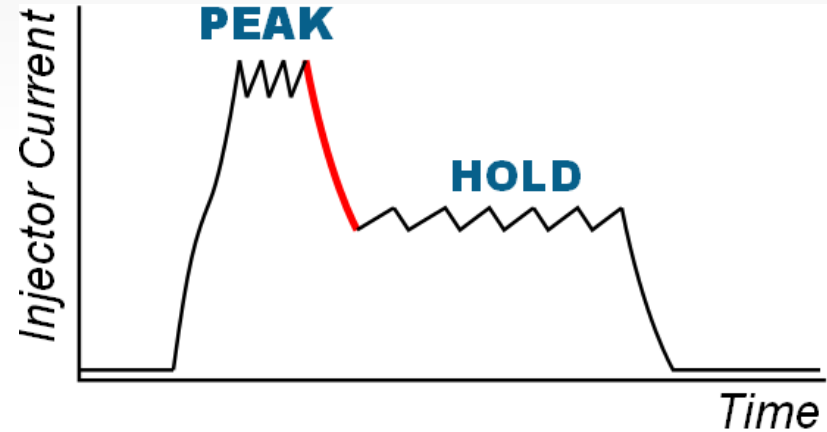
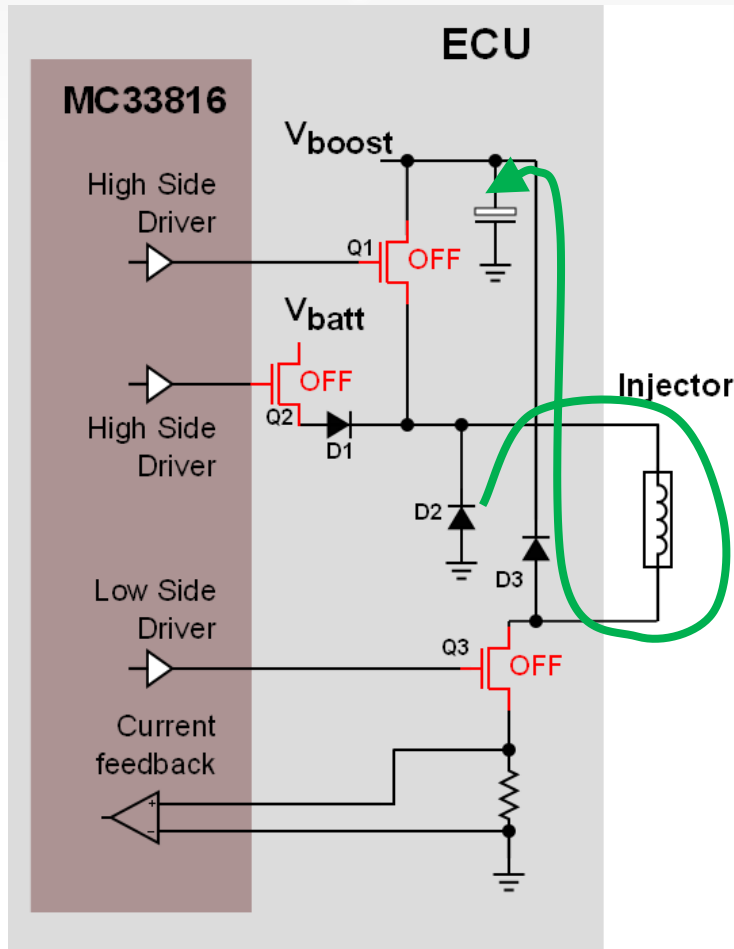
```
### peak phase continue on boost ###
```

```
peak0:      ldcd rst _ofs keep keep Tpeak_tot c1;* Load the length of the total peak phase in counter 1
            cwer peak_on0 tc2 row1;          * Jump to peak_on when tc2 reaches end of count
            cwer peak_off0 ocur row2;        * Jump to peak_off when current is over threshold
            cwer off_phase0 tc1 row3;        * Jump to off_phase when tc1 reaches end of count
            load I_peak dac_sssc _ofs;       * Load the peak current threshold
```

```
peak_on0:   stos off on on;                  * Turn VBAT off, BOOST on, LS on
            wait row235;                    * Wait for one of the previously defined conditions
```

```
peak_off0:  ldcd rst ofs keep keep Tpeak_off c2;* Load in the counter 2 the length of the peak_off phase
            stos off off on;                * Turn VBAT off, BOOST off, LS on
            wait row135;                    * Wait for one of the previously defined conditions
```

# Peak decay



- Bank 1 HS Boost OFF
- Bank 1 HS Bat OFF
- Inj1 LS OFF
- Current re-circulates through D2 and D3
- High voltage energy recovery to boost supply capacitor
- Faster decay rate than Peak OFF

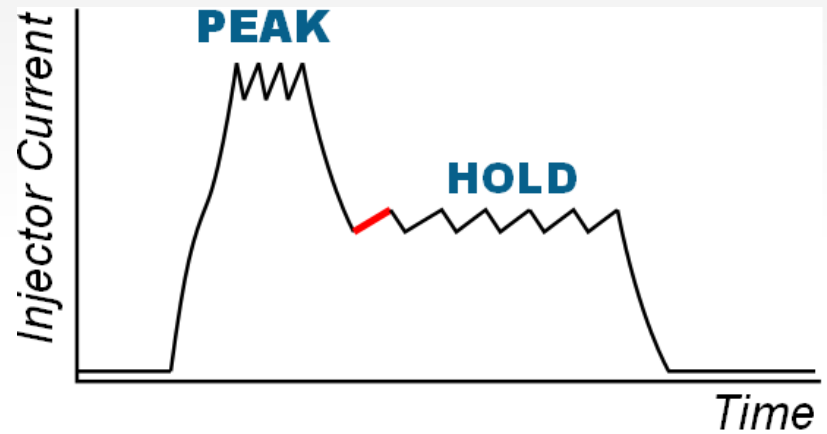
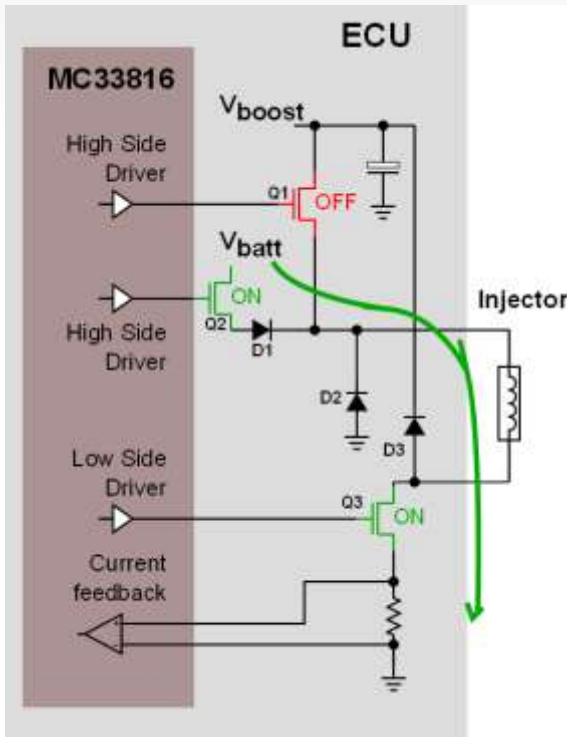
\*\*\* off phase(decay) determined by timer \*\*\*

```

off_phase0: ldcd rst ofs keep keep Toff c3; * Load in the counter 3 the length of the off_phase phase
            stos off off off; * Turn VBAT off, BOOST off, LS off
            cwer hold0 tc3 row4; * Jump to hold when tc3 reaches end of count
            wait row45; * Wait for one of the previously defined conditions
    
```



# Hold ON



- Bank 1 HS Boost OFF
- Bank 1 HS Bat ON
- Inj1 LS ON
- Current rises to hold target

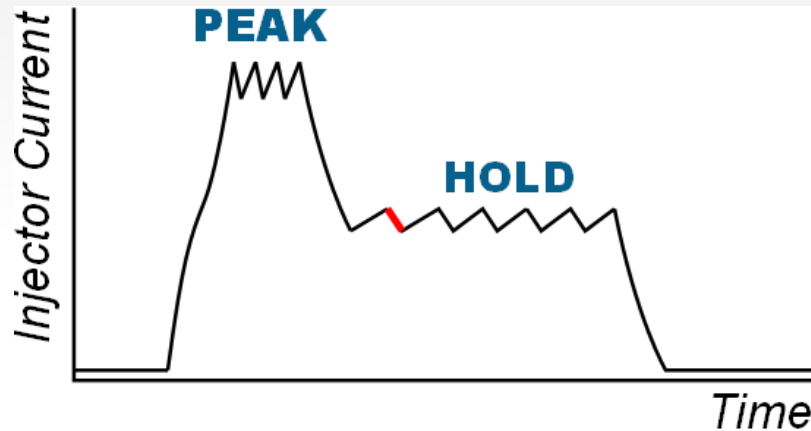
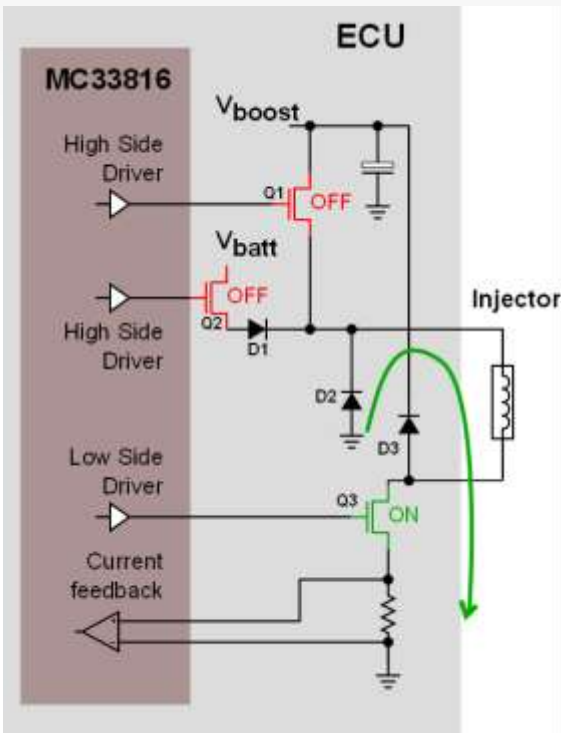
```

#### hold phase on Vbat ###
hold0:    ldcd rst_ofs keep keep Thold_tot c2; * Load the length of the total hold phase in counter 2
          load I_hold dac_sssc_ofs;          * Load the hold current threshold in the DAC
          cwer hold_off0 ocur row1;          * Jump to hold_off when current is over threshold
          cwer hold_on0 tc1 row2;           * Jump to hold_on when tc1 reaches end of count
          cwer eoinj0 tc2 row3;             * Jump to eoinj when tc2 reaches end of count

hold_on0: stos on off on;                   * Turn VBAT on, BOOST off, LS on
          wait row135;                       * Wait for one of the previously defined conditions

hold_off0: ldcd rst_ofs keep keep Thold_off c1; * Load the length of the hold_off phase in counter 1
          stos off off on;                   * Turn VBAT off, BOOST off, LS on
          wait row235;                       * Wait for one of the previously defined conditions
    
```

# Hold OFF



- Same as Peak OFF
  - Bank 1 HS Boost OFF
  - Bank 1 HS Bat OFF
  - Inj1 LS ON
- Current re-circulates through D2 and decays
- Hold phase is combination of Hold ON and Hold OFF

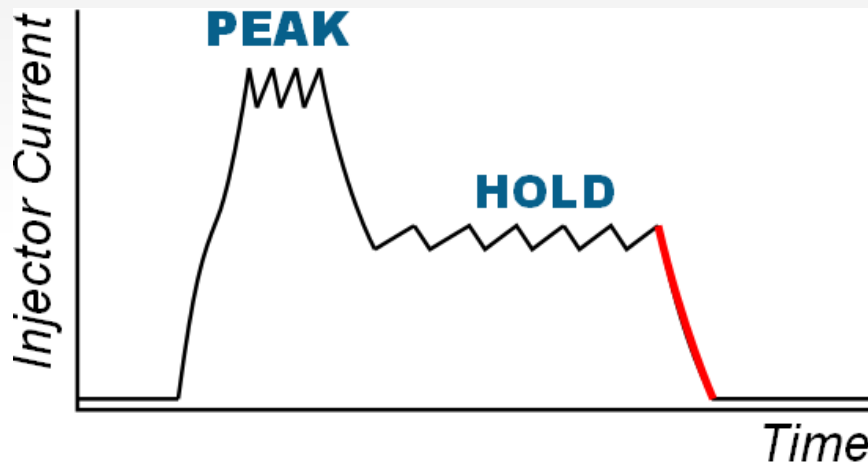
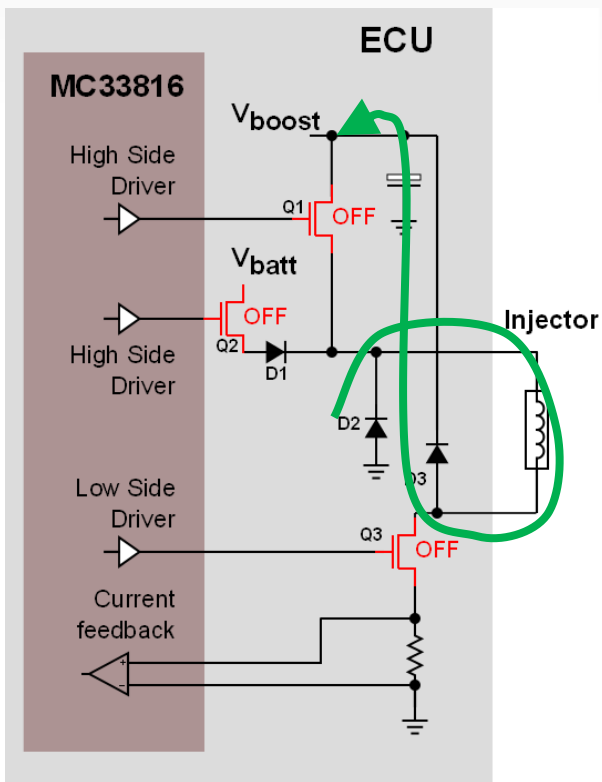
```

### hold phase on Vbat ###
hold0:  lddc rst_ofs keep keep Thold_tot c2; * Load the length of the total hold phase in counter 2
        load I_hold dac_sssc_ofs; * Load the hold current threshold in the DAC
        cwer hold_off0 ocur row1; * Jump to hold_off when current is over threshold
        cwer hold_on0 tc1 row2; * Jump to hold_on when tc1 reaches end of count
        cwer eoinj0 tc2 row3; * Jump to eoinj when tc2 reaches end of count

hold_on0: stos on off on; * Turn VBAT on, BOOST off, LS on
          wait row135; * Wait for one of the previously defined conditions

hold_off0: lddc rst_ofs keep keep Thold_off c1; * Load the length of the hold_off phase in counter 1
          stos off off on; * Turn VBAT off, BOOST off, LS on
          wait row235; * Wait for one of the previously defined conditions
    
```

# End of Injection



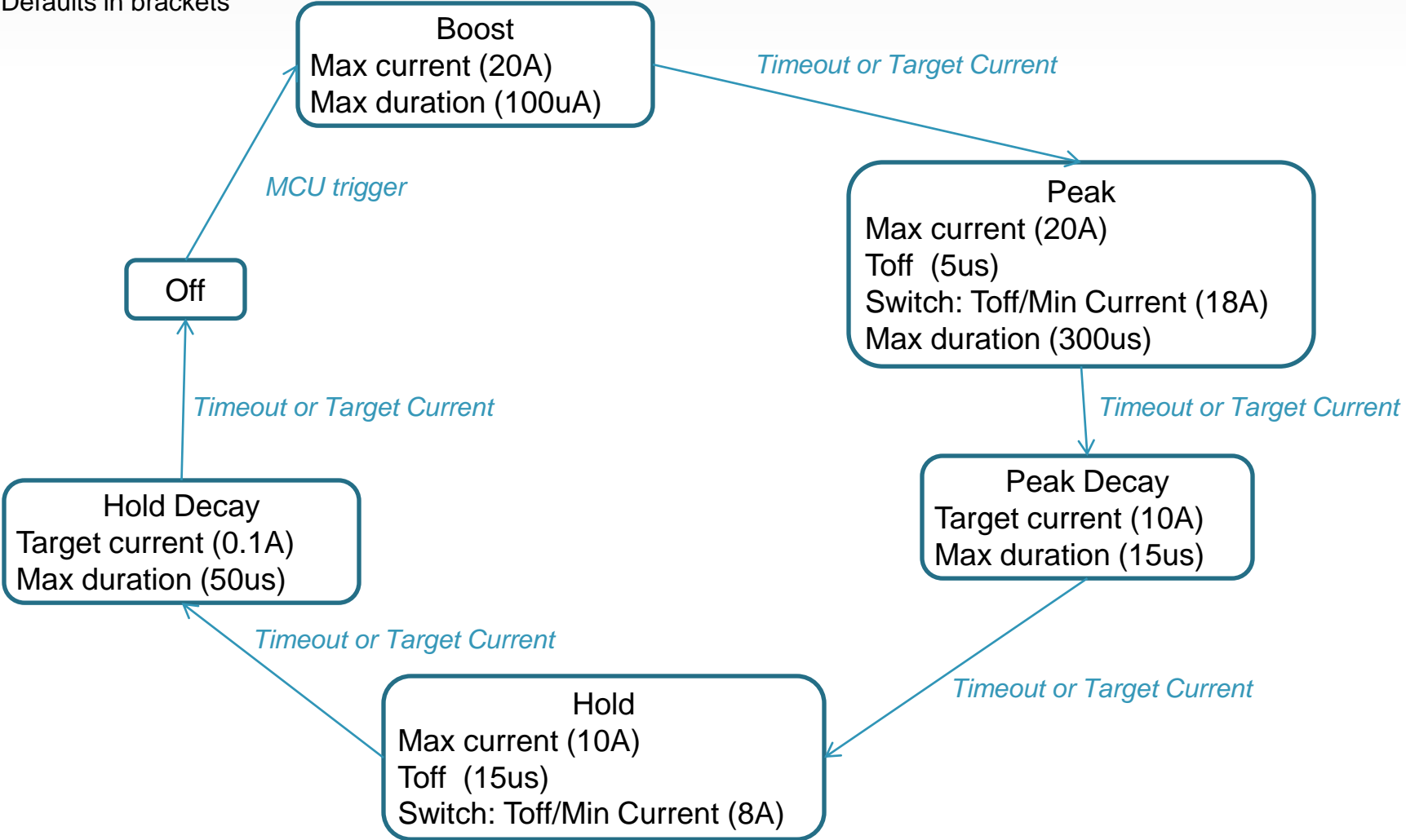
- Same as Peak Decay
  - Bank 1 HS Boost OFF
  - Bank 1 HS Bat OFF
  - Inj1 LS OFF
- Current re-circulates through D2 and D3
- High voltage energy recovery to boost supply capacitor

### finished cycle turn off everything jump to idle phase ###

```
eoinj0:      stos off off off;          * Turn VBAT off, BOOST off, LS off
            ;                      * Jump back to IDLE PHASE
            jmpf jr2;
```

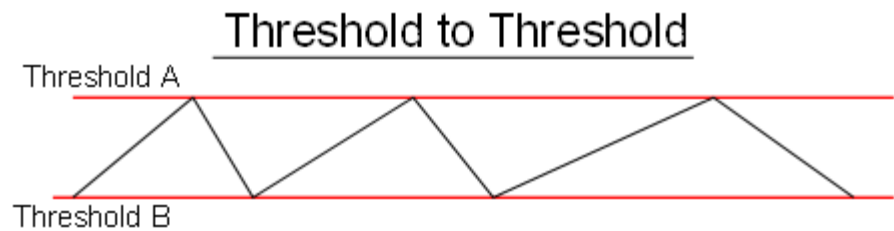
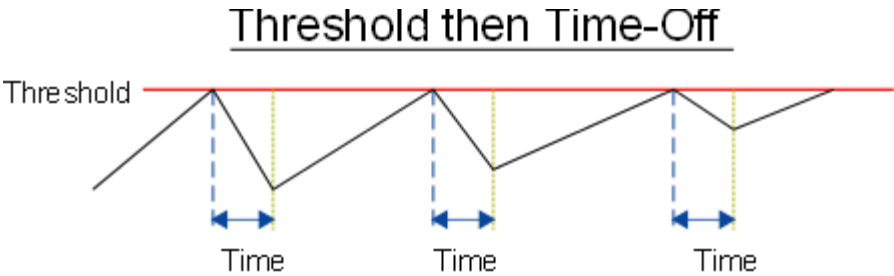
# Peak and hold programmable parameters

Defaults in brackets



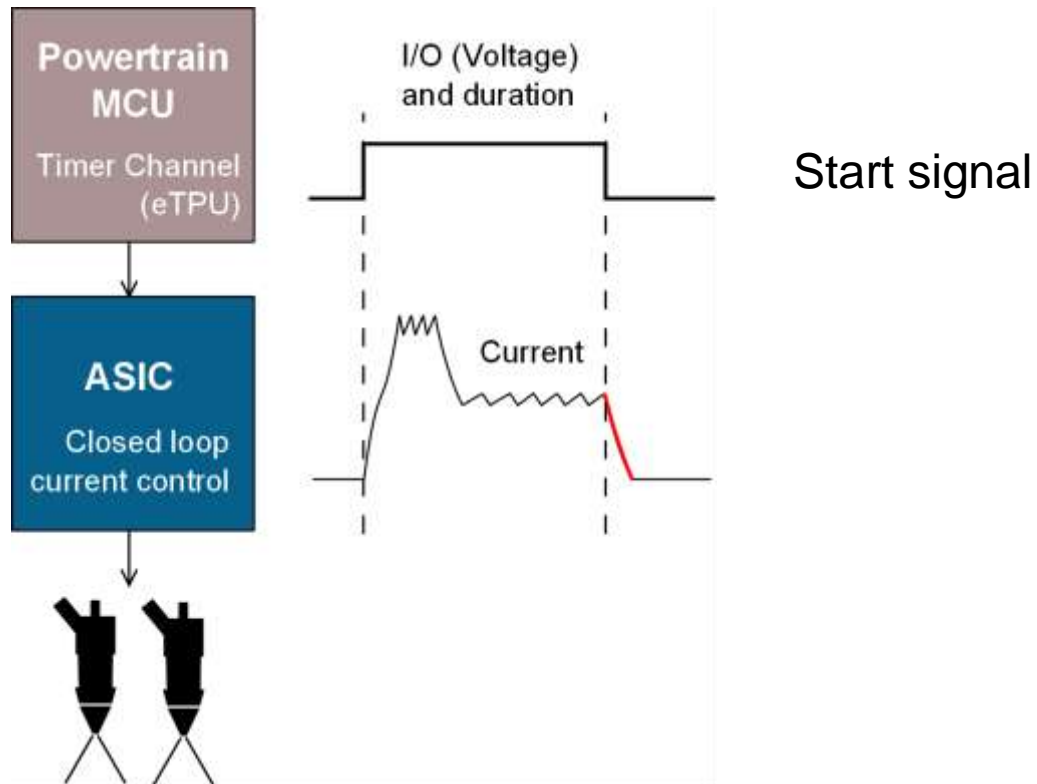
# Software routines

- Manage transitions. Current may be above or below threshold on entry, use appropriate state ON or OFF
- Option on current control:
  - Threshold then Toff
  - Threshold to threshold (Current measurement 4 only)



# Implementation on ECU

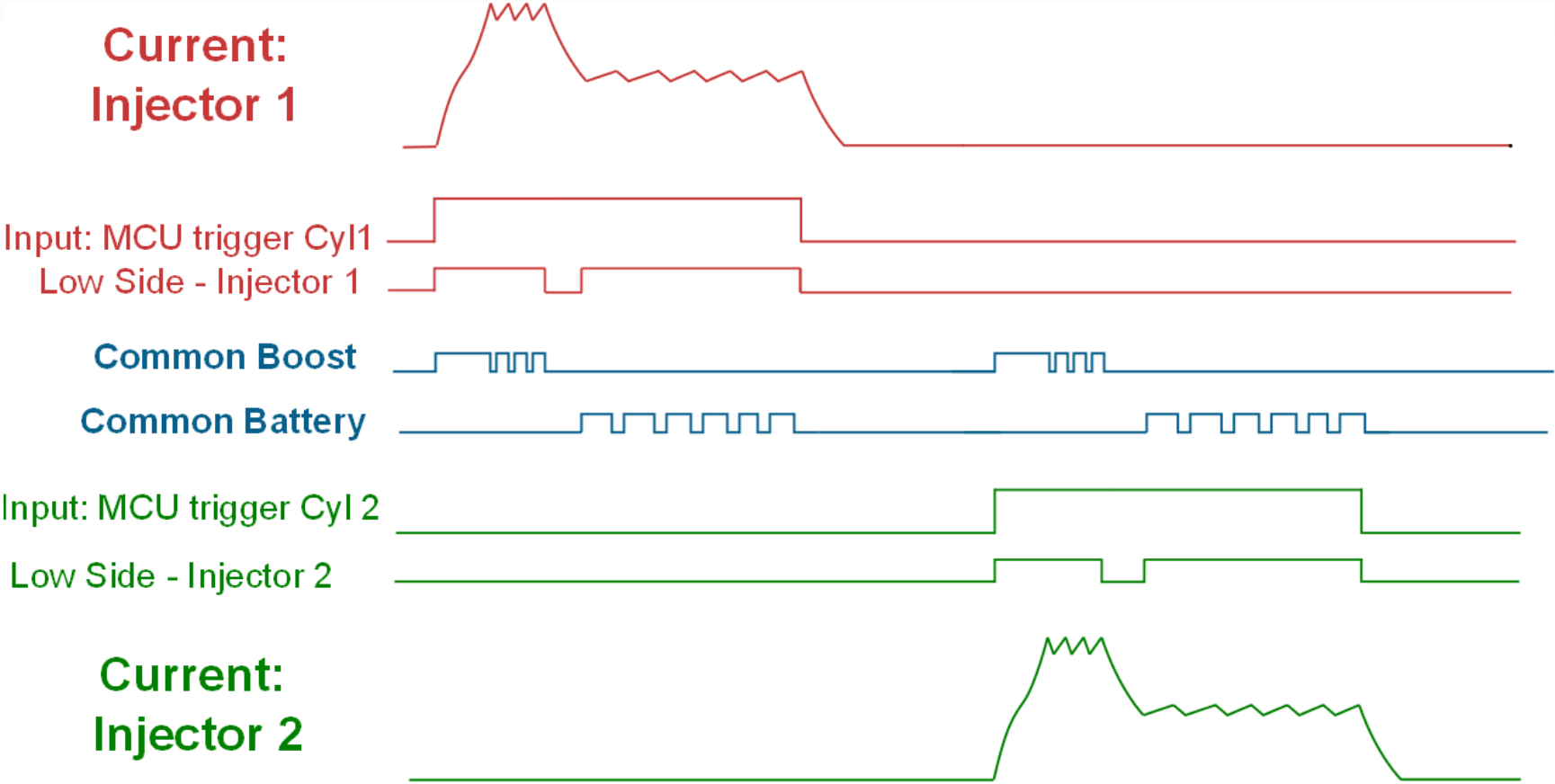
- Currents and default durations programmed over SPI
- Start signal from MCU rising triggers start of injection pulse
- Start signal from MCU triggers end of HOLD phase



# Channels required for EVB

- High side Boost supply FET with PWM
  - Two on EVB
- High side Battery supply FET with PWM
  - Two on EVB
- Low side FET with PWM
  - Four on EVB
- Current sense low side channel
  - Four on EVB

# Waveforms: banked mode







# Agenda

- Purpose for introducing the MC33816
- Overview of MC33816
  - Hardware – Circuit functions and features
  - Software – Instruction set and tools
- MC33816 Applications and Examples
  - 4 and 6 cylinder examples
  - Peak and Hold waveform generation
  - Boost voltage regulator



# Thank You!

- For more information see the following additional slides or contact the authors.

[Facebook.com/Freescale](https://www.facebook.com/Freescale)

Tag yourself in photos  
and upload your own!



Twisting?

Please use hashtag  
**#FTF2012**



**Session materials will be posted @ [www.freescale.com/FTF](http://www.freescale.com/FTF)**

Look for announcements in the FTF Group on LinkedIn or follow Freescale on Twitter



**FTF** | FREESCALE TECHNOLOGY FORUM  
POWERING INNOVATION

# Additional Backup Slides and Information



Freescale, the Freescale logo, AlliVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, mobileGT, PowerQUICC, Processor Expert, QorIQ, Qorivva, StarCore, Symphony and VortiQa are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, CoreNet, Flexis, MagniV, MXC, Platform in a Package, QorIQ Qonverge, QUICC Engine, Ready Play, SafeAssure, the SafeAssure logo, SMARTMOS, TurboLink, Vybrid and Xtrinsic are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © 2012 Freescale Semiconductor, Inc.

# Additional Backup Slides and Information

- Diagnostics and FMEN
- Flag Configuration Details
- Instructions Detailed Description

# Diagnostics and FMEM #1

- Boost FET
  - Open: report, leave FET on
  - Short to Vbat: report, turn FET off and retry next event
  - Short to Gnd: report, turn FET off and retry next event
- Battery FET
  - Open: report, leave FET on
  - Short to Vbat: report, leave FET on
  - Short to Gnd: report, turn FET off and retry next event
- Low side FET
  - Open: report, leave FET on
  - Short to Vbat: report, turn FET off and retry next event
  - Short to Gnd: report, leave FET on

## Diagnostics and FMEM #2

- Failure to reach Boost voltage: report and continue
- Failure to reach target current in prescribed duration (any phase): report and continue

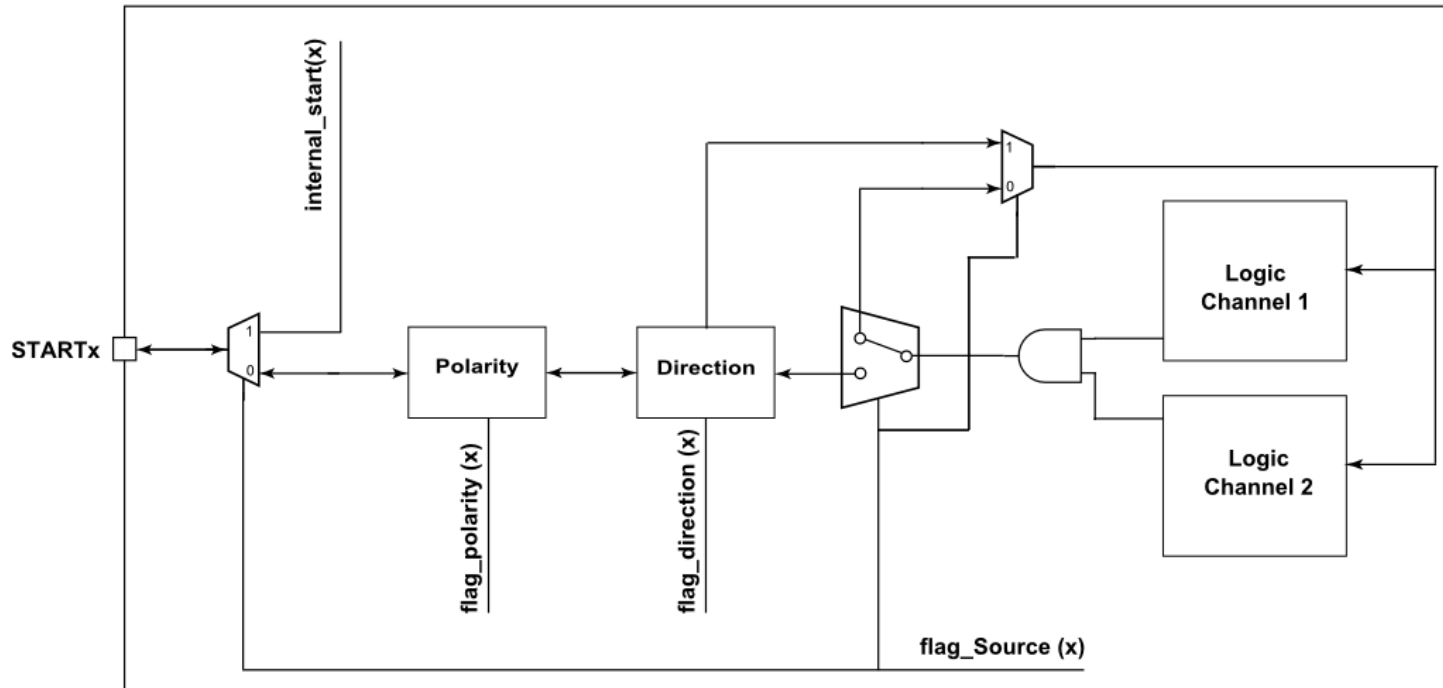


# Additional Backup Slides and Information

- Diagnostics and FMEN
- Flag Configuration Details
- Instructions Detailed Description

# Flags 3 to 8 (STARTx pins)

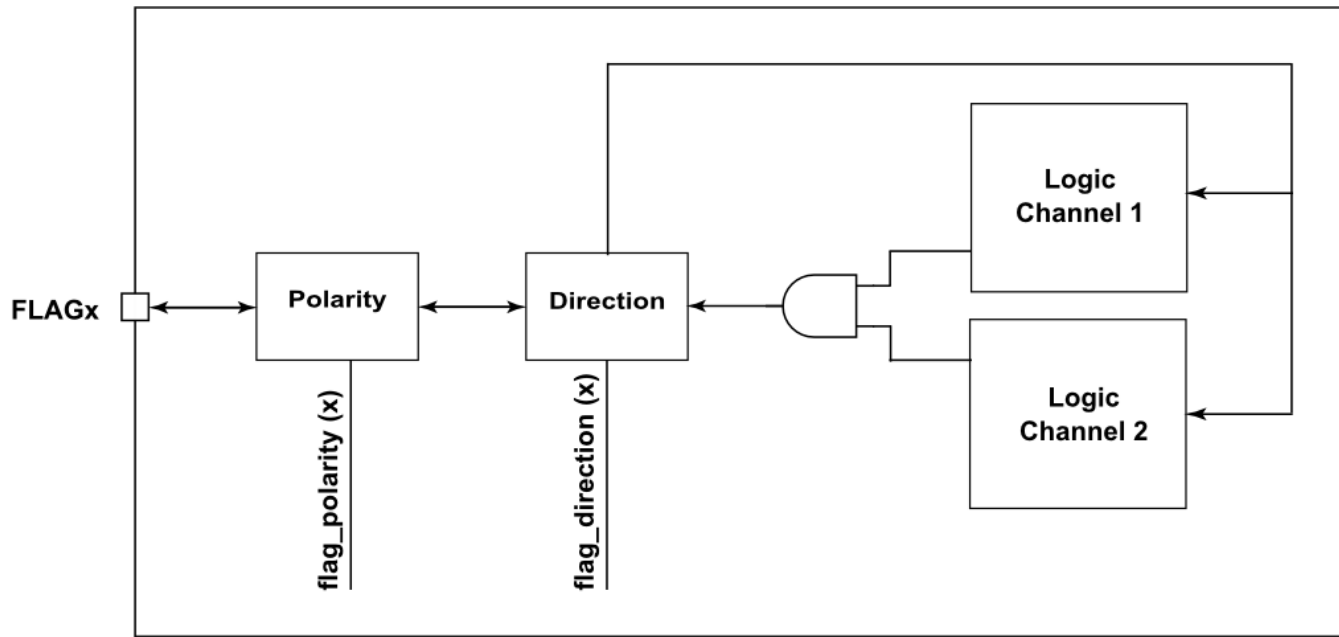
- ✓ Can be connected to the STARTx pins (digital input pins)
- ✓ Can provide direct digital inputs to the  $\mu$ Cores (programmable)
- ✓ Set as output combination of the Logic Channels 1 and 2 local flags
- ✓ Set as internal flag the STARTx pin keep their primary functions.





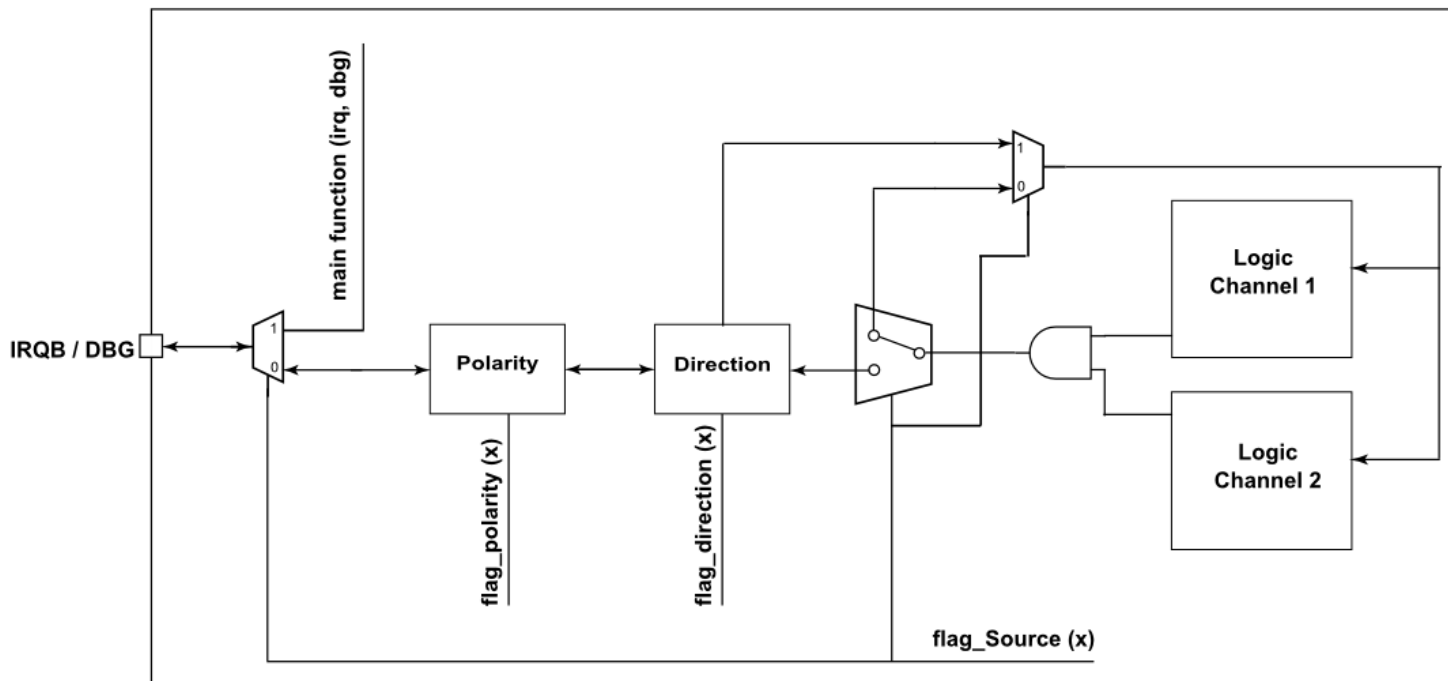
# Flag 0 to Flag 2 (FLAGx pins)

- ✓ **Always connected to FLAGx pins (digital input pins)**
- ✓ **Set as input directly feedbacks the  $\mu$ Cores**
- ✓ **Set as output combination of the Logic Channels 1 and 2 'local' flags**



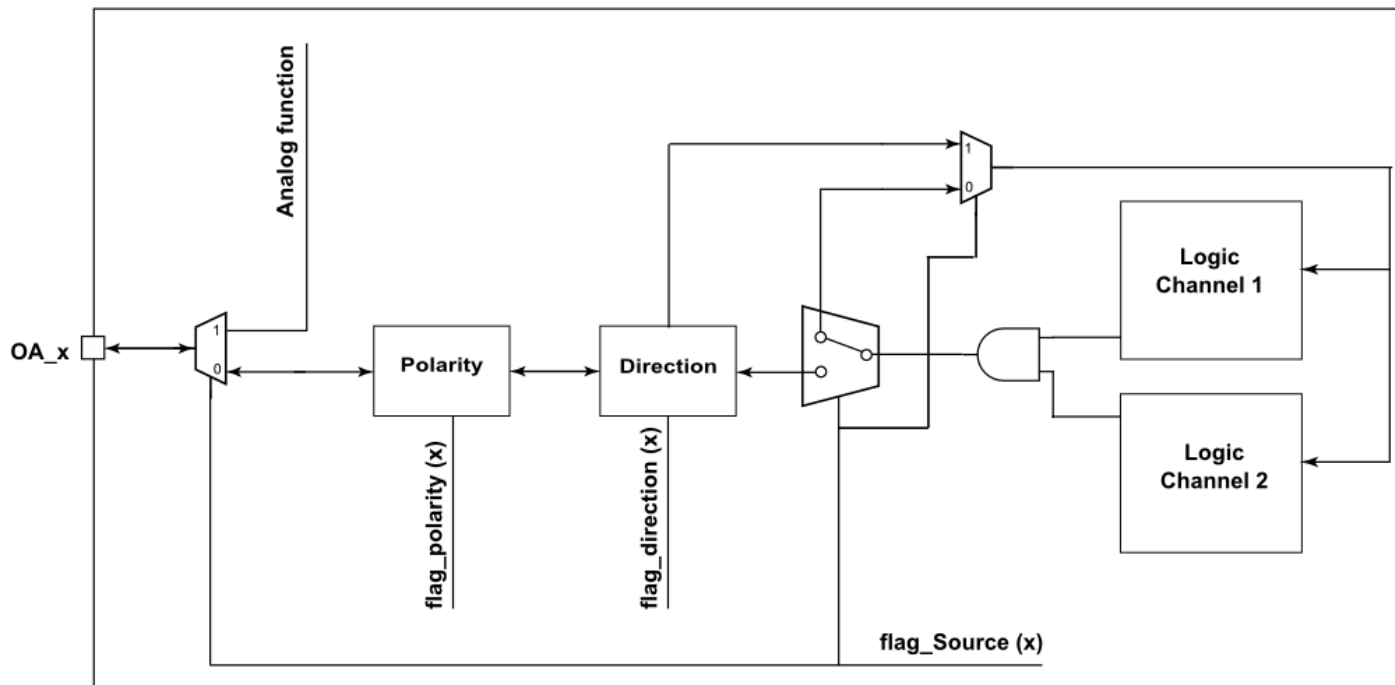
# Flag 9 and Flag 12 (pins IRQB and DBG)

- ✓ Can be connected to the IRQB and DBG pins (digital pins)
- ✓ Set as input directly feedbacks the  $\mu$ Cores
- ✓ Set as output combination of the Logic Channels 1 and 2 'local' flags
- ✓ Set as internal flag the STARTx pin keep their primary functions.



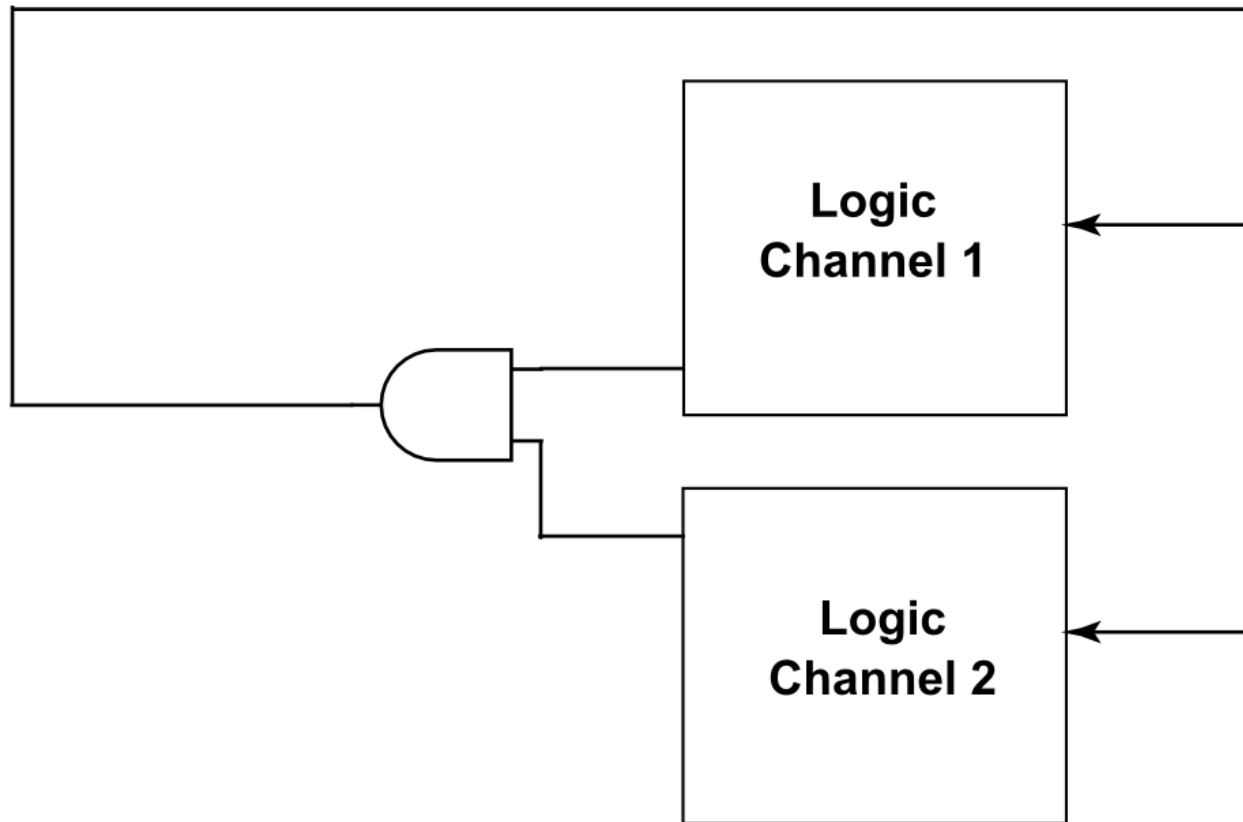
# Flag 10 and Flag 11 (pins OA\_x)

- ✓ Can be connected to the OA\_x pins (analog output pins)
- ✓ Set as input directly feedbacks the  $\mu$ Cores
- ✓ Set as output combination of the Logic Channels 1 and 2 'local' flags
- ✓ Set as internal flag the OA\_x pin keep their primary functions.



# Flag 13 to Flag 15

- **Internal flags only**



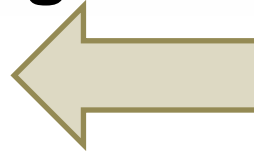
# Additional Backup Slides and Information

- Diagnostics and FMEN
- Flag Configuration Details
- Instructions Detailed Description

# Software - Instruction Set Overview

## ✓ 93 instructions in 7 categories:

1. **Arithmetic Logic Unit (27)**



2. **Configuration (29)**

3. **Diagnostic (3)**

4. **Interrupt and Subroutine (6)**

5. **Jump (16)**

6. **Load (9)**

7. **Wait (3)**

## 27 Arithmetic Logic Unit (ALU) Instructions

1. ***add*** – addition of two registers
2. ***addi*** – addition with immediate register
3. ***and*** – mask AND with immediate register
4. ***mul*** – multiplication of two registers
5. ***muli*** – multiplication with immediate value
6. ***not*** – invert contents of a register
7. ***or*** – mask OR with immediate register
8. ***sub*** – subtraction of two registers
9. ***subi*** – subtraction with immediate register
10. ***swap*** – swap high and low bytes of a register
11. ***toc2*** – performs 2's complement on register
12. ***toint*** – converts 2's complement to integer
13. ***xor*** – mask XOR with immediate register

## 27 ALU Instructions (con't)

14. ***sh32l*** – shift 32-bit register left (register)
15. ***sh32li*** – shift 32-bit register left immediate
16. ***sh32r*** – shift 32-bit register right (register)
17. ***sh32ri*** – shift 32-bit register right immediate
18. ***shl*** – shift register left (register)
19. ***shl8*** – shift register left 8 positions
20. ***shli*** – shift register left immediate
21. ***shls*** – shift register left signed (register)
22. ***shlsi*** – shift register left signed immediate
23. ***shr*** – shift register right (register)
24. ***shr8*** – shift register right 8 positions
25. ***shri*** – shift register right immediate
26. ***shrs*** – shift register right signed (register)
27. ***shrsi*** – shift register right signed immediate



# ALU Instructions: Addition and Subtraction

The ALU contains a single-cycle 16-bit adder. Overflow and underflow are managed according to the configuration called *arithmetic\_logic*.

Instruction “`stal arithLogic`”:

- Set the ARITH\_LOGIC(1:0) bits of the ALU configuration register:
  - `a11, C2` : number are considered to be 2-complement. Overflow limit is 0x7FFF, underflow limit is 0x8000.
  - `a12, C2sat` : number are considered to be 2-complement. Overflow limit is 0x7FFF, underflow limit is 0x8000. In case of overflow the result is saturated to 0x7FFF, in case of underflow the result is saturated to 0x8000.
  - `a13, Pos` : number are considered to be positive. Overflow limit is 0xFFFF, underflow limit is 0x0000.
  - `a14, PosSat` : number are considered to positive. Overflow limit is 0xFFFF, underflow limit is 0x0000. In case of overflow the result is saturated to 0xFFFF, in case of underflow the result is saturated to 0x0000.

Condition register bits affected:

- ARITH\_LOGIC (12-11)

# ALU Instructions: Addition and Subtraction (con't)

Instruction “`add op1 op2 res`”:

- $\text{Reg}[\text{res}] = \text{Reg}[\text{op1}] + \text{Reg}[\text{op2}]$

Instruction “`addi op1 immediate res`”:

- $\text{Reg}[\text{res}] = \text{Reg}[\text{op1}] + \text{imm}$

Instruction “`sub op1 op2 res`”:

- $\text{Reg}[\text{res}] = \text{Reg}[\text{op1}] - \text{Reg}[\text{op2}]$

Instruction “`subi op1 immediate res`”:

- $\text{Reg}[\text{res}] = \text{Reg}[\text{op1}] - \text{imm}$

Condition register bits affected:

- CARRY (13), RES\_ZERO (6), RES\_SIGN(5), UNSIGNED\_UND (4),  
UNSIGNED\_OVR (3), SIGNED\_UND (2), SIGNED\_OVR (1)

Note: The operand ‘imm’ is a 4-bit immediate value a an direct instruction parameter.

## ALU Instructions: Multiplication (con't)

The ALU contains a single-cycle 16-bit x 16-bit multiplier. The multiplier produces a 32-bit result, so overflow is not possible. The result is correct only if the operands are represented as positive numbers. The result is available after 32 clock cycles.

Instruction “`mul op1 op2`”:

- $m = \text{Reg}[\text{op1}] * \text{Reg}[\text{op2}]$

Instruction “`muli op1 immediate`”:

- $\text{GPR6} \ \& \ \text{GPR7} = \text{Reg}[\text{op1}] * \text{imm}$

Condition register bits affected:

- `MUL_SHIFT_OVR` (8), `MUL_SHIFT_LOSS` (7), `OP_DONE` (0)

# ALU Instructions: Multiplication (con't)

To multiply 2-complement represented numbers, they must first be converted to positive representation.

Instruction “`toint op1 rst`”:

- Extract the module from a 2-complement represented number. The sign can either replace or be accumulated in the CONV\_SIGN bit. The operation consumes the operand.

Instruction “`toc2 op1`”:

- Convert the operand from positive to 2-complement representation by adding the sign contained in the CONV\_SIGN bit.
- Example:

```

toint r0 rst;      * convert the first operand => rst => *
                  * CONV_SIGN = signof(r0)           *
toint r1 _rst;    * convert the first operand =>_rst => *
                  * CONV_SIGN = CONV_SIGN * signof(r1) *
mul r0 r1;
toc2 mh;          * takes as a result 16 MSBs of the result *
                  * and add the the sign CONV_SIGN = signof(r1) * signof(r2) *

```

Condition register bit affected:

- CONV\_SIGN (14)

# ALU Instructions: Shift

The result of these shift operation is available after a number of clock cycles equal to the number of shift positions required.

Instruction “`shr op1 op2`”:

- `Reg[op1] = Reg[op1] >> Reg[op2]`. The bits shifted in are all ‘0’.

Instruction “`shl op1 op2`”:

- `Reg[op1] = Reg[op1] << Reg[op2]`. The bits shifted in are all ‘0’.

Instruction “`shri op1 immediate`”:

- `Reg[op1] = Reg[op1] >> immediate`. The bits shifted in are all ‘0’.

Instruction “`shli op1 immediate`”:

- `Reg[op1] = Reg[op1] << immediate`. The bits shifted in are all ‘0’.

ALU Condition register bits affected:

- `SHIFT_OUT` (15), `OP_DONE` (0)

# ALU Instructions: Shift (con't)

The result of these shift operation is available after a number of clock cycles equal to the number of shift positions required. In all these instruction the MSB is not shifted.

Instruction “`shrs op1 op2`”:

- $\text{Reg}[\text{op1}] = \text{Reg}[\text{op1}] \gg \text{Reg}[\text{op2}]$ . The bits shifted in are equal to the MSB.

Instruction “`shls op1 op2`”:

- $\text{Reg}[\text{op1}] = \text{Reg}[\text{op1}] \ll \text{Reg}[\text{op2}]$ . The bits shifted in are all ‘0’.

Instruction “`shrsi op1 immediate`”:

$\text{Reg}[\text{op1}] = \text{Reg}[\text{op1}] \gg \text{immediate}$ . The bits shifted in are equal to the MSB.

Instruction “`shlsi op1 immediate`”:

- $\text{Reg}[\text{op1}] = \text{Reg}[\text{op1}] \ll \text{immediate}$ . The bits shifted in are all ‘0’.

ALU Condition register bits affected:

- SHIFT\_OUT (15), OP\_DONE (0)

# ALU Instructions: Shift (con't)

The result of these shift operation is available after a number of clock cycles equal to the number of shift positions required. These instructions operate only on the 32-bit register.

Instruction “`sh32r op1`”:

- $m = mh \& ml \gg \text{Reg}[op1]$ . The bits shifted in are all ‘0’.

Instruction “`sh32l op1`”:

- $m = mh \& ml \ll \text{Reg}[op1]$ . The bits shifted in are all ‘0’.

Instruction “`sh32ri immediate`”:

- $m = mh \& ml \gg \text{immediate}$ . The bits shifted in are all ‘0’.

Instruction “`sh32li immediate`”:

- $m = mh \& ml \ll \text{immediate}$ . The bits shifted in are all ‘0’.

ALU Condition register bits affected:

- SHIFT\_OUT (15), OP\_DONE (0)

# ALU Instructions: Shift (con't)

These operations are all single cycle. Useful for 8 bit operations.

Instruction “`sh8r op1`”:

- $\text{Reg}[\text{op1}] = \text{Reg}[\text{op1}] \gg 8$ . The bits shifted in are all '0'.

Instruction “`sh8l op1`”:

- $\text{Reg}[\text{op1}] = \text{Reg}[\text{op1}] \ll 8$ . The bit shifted in are all '0'.

Instruction “`swap op1`”:

- $\text{Reg}[\text{op1}] = (\text{Reg}[\text{op1}] \ll 8) + (\text{Reg}[\text{op1}] \gg 8)$ .

ALU Condition register bits affected:

- None



# ALU Instructions: Logic operations

Logic operations always consumes the operand. If a mask is needed, the `immediate register` is used (`r5`).

Instruction “`not op1`”:

- Every bit of `Reg[op1]` is inverted.

Instruction “`and op1`”:

- `Reg[op1] = Reg[op1] & r5`.

Instruction “`or op1`”:

- `Reg[op1] = Reg[op1] | r5`.

Instruction “`xor op1`”:

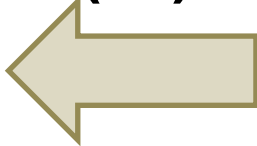
- `Reg[op1] = Reg[op1] ^ r5`.

Condition register bits affected:

- `MASK_MIN(10)`, `MASK_MAX(9)`

# Software - Instruction Set Overview

## ✓ 93 instructions in 7 categories:

1. **Arithmetic Logic Unit (27)**
2. **Configuration (29)** 
3. **Diagnostic (3)**
4. **Interrupt and Subroutine (6)**
5. **Jump (16)**
6. **Load (9)**
7. **Wait (3)**

## 29 Configuration Instructions

1. ***bias*** – enables/disables a single bias structure
2. ***chth*** – changes threshold on feedback comparator
3. ***dfcsct*** – define shortcut for current feedback
4. ***dfsct*** – define shortcut for outputs
5. ***rdspi*** – request SPI read
6. ***rstreg*** – reset register
7. ***rstsl*** – reset start latch register
8. ***slab*** – select address base
9. ***slfbk*** – select feedback source
10. ***slsa*** – selects which register to use as SPI address
11. ***stab*** – load value in address base register
12. ***stadc*** – enables/disables ADC conversion on specified current measurement block

## 29 Configuration Instructions (con't)

13. **stal** – set ALU mode (ARITH\_LOGIC(1:0) bits )
14. **stcrb** – set control register bit
15. **stcrt** – set channel communication register
16. **stdcctl** – set DC/DC control mode
17. **stdm** – set DAC register access mode
18. **stdrm** – set DRAM read mode
19. **steoa** – enable end of actuation mode
20. **stf** – set flag
21. **stfw** – set freewheeling mode
22. **stgn** – set Op Amp gain
23. **stirq** – set interrupt request output pin
24. **sto** – set single output
25. **stoc** – set offset compensation
26. **stos** – set output shortcut

## 29 Configuration Instructions (con't)

- 27. ***stslew*** – set slew rate
- 28. ***stsrbit*** – set status register bit
- 29. ***wrspi*** – request SPI write (backdoor)

# Bias, Threshold and Slewrate Instructions

The action of all these instructions is masked by the “output\_access” register of the  $\mu$ Core.

## Instruction “`bias sel ctrl`”:

- Biasing can be configured as described below.
  - a single bias structure
  - all HS bias structures
  - all LS bias structures
  - all bias structures

## Instruction “`stslew sel`”:

- Force or unforce the value of the slewrate to the maximum value.

## Instruction “`chth sel level`”:

- Set the threshold of a voltage feedback comparator.

# Configuration Instructions: Internal Registers and Data RAM

Instruction “`stab addrbase`”:

- Set the Data RAM Address Base.

Instruction “`slab sel`”:

- Force the ALU *GPR5* (immediate register) as Data RAM offset instead of the Address Base. By default the Address Base is used.

# Configuration Instructions: Internal Registers and Data RAM (con't)

Instruction “`stdrm sel`”:

- Select the “data ram read mode” per the combinations below:
  - `word`: when a “load” instruction is executed, all the 16 bit of the data ram value are written in the destination register. This is the default value.
  - `lowbyte`: when a “load” instruction is executed, all the 8 LSBs of the data ram value are written in the 8 LSBs of the destination register. The 8 MSBs of the destination register are filled with ‘0’.
  - `highbyte`: when a “load” instruction is executed, all the 8 MSBs of the data ram value are written in the 8 LSBs of the destination register. The 8 MSBs of the destination register are filled with ‘0’.
  - `swapbyte`: when a “load” instruction is executed, all the 8 MSBs and the 8 LSBs of the data ram value are written swapped in the destination register.



# Configuration Instructions: SPI ‘Backdoor’

Reading or writing an SPI accessible register always requires a multi-instruction set.

Instruction “`slsa sel`”:

- Select if the address to be used for the operation is *spi\_add* or *GPR5*. By default the address used is *spi\_add*.

Instruction “`rdspi`”:

- Read the SPI register at the address  $0x100 +$  the selected address. The result is available at in the register *spi\_data*.

Instruction “`wrspi`”:

- Write the value contained in the internal register *spi\_data* into the SPI register at the address  $0x100 +$  the selected address.

# Shortcut Instructions

In order to optimize the code size 3 output shortcuts and 1 current feedback shortcut can be defined.

## Instruction “`dfcsct short`”:

- Define one of the current feedback as shortcut. For this purpose, only current feedbacks 1 to 4L can be selected (define 4H and 4Neg is not allowed). The instructions that can use the current feedback defined as shortcut are:
  - conditional jump/wait on feedbacks (“`jocr/f`”, “`cwer/f`”) (own current feedback high/low)

## Instruction “`dfsct short1 short2 short3`”:

- Define up to 3 output shortcuts. More than 1 shortcut can point to the same output. The instructions that can use the output defined as shortcuts are:
  - load complex (“`ldcd`”, “`ldca`”)
  - set output shortcuts (“`stos`”)
  - jump/wait condition on feedbacks (“`jocr/f`”, “`cwer/f`”)

# Status and Control Registers Instructions

The status register is 16-bit wide. It can be read/written by the  $\mu$ Core, while the external micro-controller can only read this register through SPI.

The register can be used as an output communication channel towards an external device or as temporary register. A combination of the two is possible.

Instruction “`stsrb value sel`”:

- Set the value of one of the bits of the status register.
- The status register can be read/written as a word (using “`cp`”, “`load`” or “`store`”) or a single bit test (using “`jsrr/f`”).

# Status and Control Registers Instructions (con't)

The control register is made of two 8-bit part:

- CR[7:0], can only be read by the  $\mu$ Core, while the external micro-controller can read or write this slice. This slice can be used as an input communication channel from an external device.
- CR[15:8] have a configurable behavior. This slice can be used like either the lower slice (CR[7:0]) or as the status register.

Instruction “`stcrb value sel`”:

- Set the value of one of the bits of the upper slice of the control register. This operation is successful only if the slice is configured to behave like the status register.

The control register can be read as a whole (using “`cp`”, “`load`” or “`store`”) or a single bit can be tested (using “`j_crr/f`”), regardless of the configuration of the CR[15:8] slice.

# Configuration Instructions: Flags Instructions

- The channel flags can be set all at once with the `cp` or `load` instructions, one at time with the `stf` instruction.

## Instruction “`stf value sel`”:

- Set the value of one of channel flags.
- The flags can be read as a whole (using “`cp`” or “`store`”) or a single bit can be tested (using “`jocr/f`” or “`cwer/f`”).

# Configuration Instructions: Output Drivers

The outcome of all these instructions is masked by the “output\_access” registers of the  $\mu$ Core.

**Instruction “`sto sel outValue`”:**

- Set the status of a single output driver.

**Instruction “`stos outValue1 outValue2 OutValue3`”:**

- Set the status of the outputs defined as shortcuts.

**Instruction “`ldcd rst ofs outValue1 outValue2 counter address`”:**

- Load a counter from Data DAM and set the outputs defined as shortcut1 and shortcut2.

**Instruction “`ldca rst outValue1 outValue2 counter address`”:**

- Load a counter from an ALU register and set the outputs defined as shortcut1 and shortcut2

# Configuration Instructions: Output Drivers (con't)

Each HS MOSFET can be configured as automatic free function. There are 5 pre-defined pairing HS-FW drivers implemented.

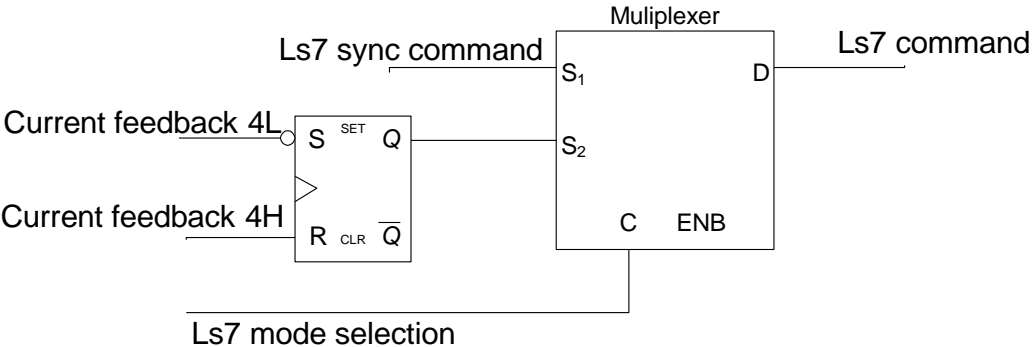
## Instruction “`stfw mode`”:

- This instruction is effective only if the output shortcut 1 is an HS, and if the  $\mu$ Core has the access right to drive the FW mosfet related to that HS. In this case, the instruction activates or deactivates automatic FW driving for that HS-FW pair.

# Configuration Instructions: Output Drivers (con't)

## Instruction “`stdcctl mode`”:

- The LS7 can be switched from the default synchronous mode (output driven by one of the  $\mu$ Cores), to asynchronous mode. In this condition it is completely independent from the code flow and its output value is determined only by the unfiltered current feedbacks.



## Instruction “`steoa diag mode`”:

- Activate end of actuation mode (used to measure when the current reaches 0A) for all the HS for which the  $\mu$ Core has access rights. It can also enable or disable the diagnosis on the related Vsource comparators.



# Configuration Instructions: DAC

7 registers are dedicated to DACs

- 4 are managed independantly –Dac4h, Dac4neg and Boost\_dac are managed as a group
- R/W access using “cp”, “load” and “store ”
- The instructions is masked by the “current\_access” register of the µCore.
- The value 15 corresponds to 0A threshold. Lower is a negative threshold, higher is positive.

Address (Hex)	Description
19B	Boost_dac
19E	Dac1_value
19F	Dac2_value
1A0	Dac3_value
1A1	Dac4l_value
1A2	Dac4h_value
1A3	Dac4neg_value

Instruction “`stdm sel`”:

- Select the “dac mode”. It can be `null`, `dac(default)`, `offset` or `full`.

# Configuration Instructions: Current Measurement Control

The action of all these instructions is masked by the “current\_access” register of the  $\mu$ Core.

Instruction “`stgn target value`”:

- Set the gain of the operational amplifier of one of the current measure blocks.

Instruction “`stadc target mode`”:

- Enable or disables adc mode on one of the current measure blocks. While the mode is active, the acquired value can be read (with “`cp`” or “`store`”) in the internal memory map at the address normally used for the corresponding DAC.

Instruction “`stoc target mode`”:

- Enable or disables automatic offset compensation on one of the current measure blocks. This mode should be active only if the differential input of the current measure block is zero.

# Configuration Instructions: Channel Communication Register

Each of the four  $\mu$ Core s has a “channel communication register” (CCR), useful to exchange 16-bit data between different  $\mu$ Cores. When this register is written (“cp” or “load”), the  $\mu$ Core writes its own register. When the register is read (“cp” or “store”), the  $\mu$ Core can read its own or the one belonging to another  $\mu$ Core.

## Instruction “`stcrt seqId`”:

- Configures a link between the  $\mu$ Core and the CCR of another  $\mu$ Core. It is used when the CCR is read. It is possible to link a  $\mu$ Core with his own CCR, to use the CCR a temporary register.

# Configuration Instructions: Interrupts

## Instruction “`rstreg target`”:

- This instruction can reset one or more of the following register.
  - Status register. It includes the higher slice of the control register, if it is configured to behave like the status register.
  - Control register. The upper slice is excluded if it is configured to behave like the status register.
  - Automatic diagnosis register, thus enabling latching the status of the output drivers when the next fault occurs.
  - Automatic diagnosis interrupt generation. Re-enables the generation of interrupt from the automatic diagnosis block.

## Instruction “`slfbk mode diag`”:

- HS2 and HS4 have 2 different Vds comparator each, one which uses Vboost pin as drain, the other which uses Vbat pin as drain. This pin selects which is the enabled comparator between the 2 and can enable or disable the diagnosis on that comparators.

# Configuration Instructions: Start Management

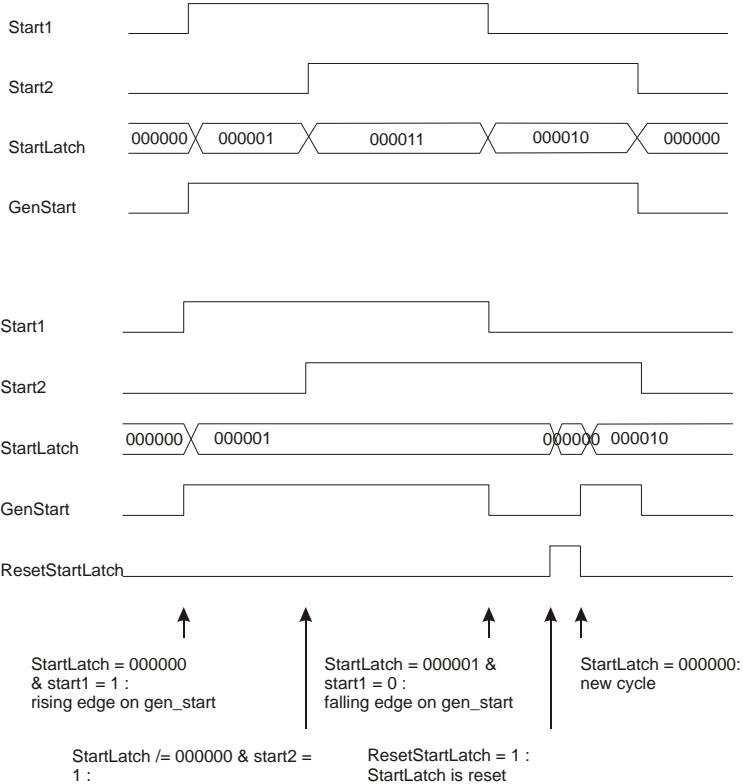
The 6 start signals can be used to trigger the actuations but each  $\mu$ Core can access only a configurable subset of these signals: `gen_start` and `start_latch`. The generation of these signals depend of the start mode selected. Each of the 4  $\mu$ Core has its own `gen_start` and `start_latch` signals set.

### Normal start mode:

- `gen_start` : logical "OR" of all the accessible start signals.
- `start_latch` : 'living copy' of all the accessible start signals.

### Smart start mode:

- `gen_start`: active when one of the start is active and `start_latch` is zero. Deactivated when the start that triggered its rising edge goes low.
- `start_latch` saves the status of the starts if one of the accessible start signal is active and the previous value of `start_latch` is zero. The `start_latch` is reset when the "`rstsl`" instruction is executed.




### Instruction "`rstsl`":

- Reset the `start_latch`, enabling the enablement to other start signals.

# Software - Instruction Set Overview

## ✓ 93 instructions in 7 categories:

1. Arithmetic Logic Unit (27)
2. Configuration (29)
3. Diagnostic (3) 
4. Interrupt and Subroutine (6)
5. Jump (16)
6. Load (9)
7. Wait (3)

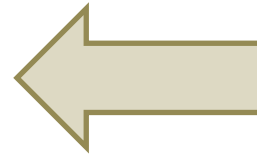
## 3 Diagnostic Instructions

1. ***endiag*** – enable diagnostics (single)
2. ***endiaga*** – enable diagnostics (all)
3. ***endiags*** – enable diagnostic shortcut

# Software - Instruction Set Overview

## ✓ 93 instructions in 7 categories:

1. Arithmetic Logic Unit (27)
2. Configuration (29)
3. Diagnostic (3)
4. Interrupt and Subroutine (6)
5. Jump (16)
6. Load (9)
7. Wait (3)





## 3 Interrupt Instructions

1. ***iconf*** – interrupt configuration
2. ***iret*** – return from interrupt
3. ***reqi*** – request for software interrupt

# Interrupt Instructions: Concept

Each  $\mu$ Core has 3 interrupt sources:

- **automatic diagnostics.** It can be enabled/disabled through SPI registers and instructions.
  - **driver disabled or loss of clock.** It can be enabled/disabled through SPI register.
  - **software interrupt.** It includes **gen\_start** rising and falling edge. The software interrupts related to **gen\_start** can be enabled/disabled through SPI registers, the others must be requested by an instruction.
- 
- When an interrupt is received, the code execution is halted and the corresponding interrupt routine is executed (address in Channel Parameter Register).
  - All interrupt routines cannot be interrupted by other interrupts. Interrupts received during this state are cached and can be serviced after the end of the currently executing routine.
  - The interrupt service routine is ended by an “`iret`” instruction.

# Interrupt Instructions

## Instruction “`iconf mode`”:

- It configures the  $\mu$ Core enablement by automatic interrupt return request.
  - `none` : automatic interrupt return request are ignored.
  - `restart` : if a matching automatic interrupt return request is received, the interrupt routine ends and the code execution restart from the entry point.
  - `continue` : if an matching automatic interrupt return request is received, the interrupt routine ends and the code execution resume from the value saved in the `irq_status` register.

## Instruction “`iret dest rst`”:

- It ends the interrupt service routine. It is possible to reset the irq buffer. The code execution continues according to the `dest` parameter:
  - `restart` : the code execution restarts from the entry point.
  - `continue` : the code execution resumes from the value saved in the `irq_status` register.

## 3 Subroutine Instructions

1. ***jtsf*** – jump to subroutine far
2. ***jtsr*** – jump to subroutine relative
3. ***rfs*** – return from subroutine

# Flow Control Instructions: Subroutine

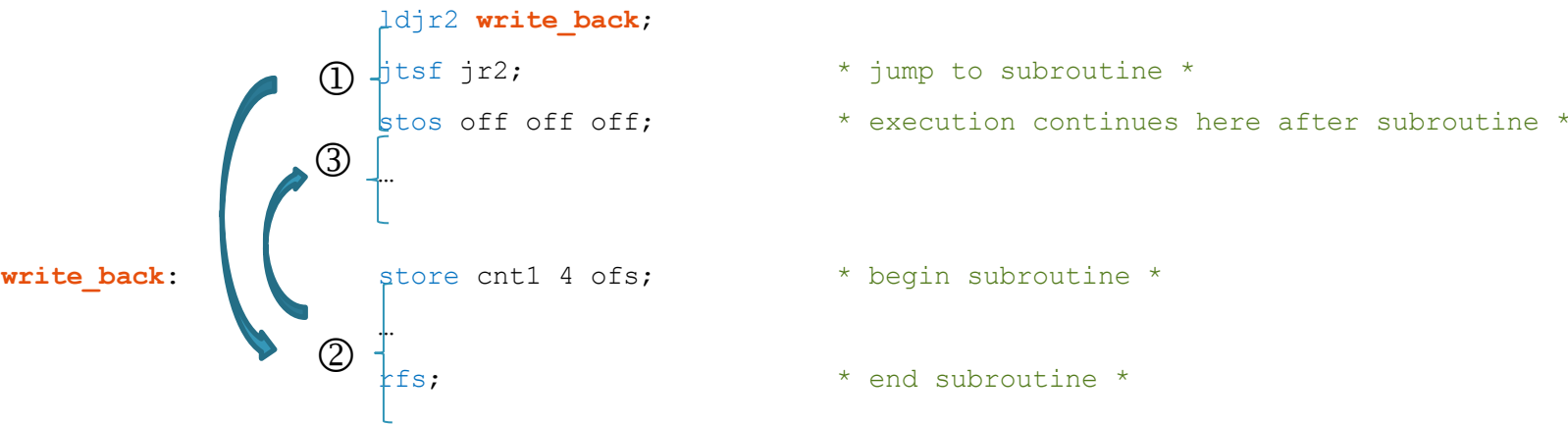
Instruction “`jtsr/f dest`”:

- Begin the subroutine at the specified address. The return address is saved into the “auxiliary” register.

Instruction “`rfs`”:


- End the interrupt routine. Code execution resumes at the value saved into the auxiliary register + 1. Return address can be manipulated.

Example:



# Software - Instruction Set Overview

## ✓ 93 instructions in 7 categories:

1. Arithmetic Logic Unit (27)
2. Configuration (29)
3. Diagnostic (3)
4. Interrupt and Subroutine (6)
5. Jump (16) 
6. Load (9)
7. Wait (3)

# 16 Jump Instructions

1. ***jarf*** – jump on **arith**metic register **far**
2. ***jarr*** – jump on **arith**metic register **relative**
3. ***jcrf*** – jump on **control** register **far**
4. ***jcrr*** – jump on **control** register **relative**
5. ***jfbkf*** – jump on **feedback** **far**
6. ***jfbkr*** – jump on **feedback** **relative**
7. ***jmpf*** – unconditional **jump** **far**
8. ***jmpr*** – unconditional **jump** **relative**
9. ***jocf*** – jump **on** flag **condition** **far**
10. ***jocr*** – jump **on** flag **condition** **relative**
11. ***joidf*** – jump **on**  $\mu$ core **id** **far**
12. ***joidr*** – jump **on**  $\mu$ core **id** **relative**
13. ***joslf*** – jump **on** **start-latch** **far**

## 16 Jump Instructions (con't)

14. ***joslr*** – jump on **start-latch** relative
15. ***jsrf*** – jump on **status register** far
16. ***jsrr***– jump on **status register** relative



# Jump Instructions: Test Conditions

- ✓ Flags (high or low)
- ✓ Internal counters
- ✓ Start trigger signals (high or low)
- ✓ Voltage feedback signals (high or low)
- ✓ Boost voltage feedback signals (high or low)
- ✓ Current feedback signals (high or low)
- ✓ ALU condition register bits
- ✓ Status register bits (internal status) (high or low)
- ✓ Control register bits (commands from microcontroller) (high or low)
- ✓  $\mu$ Core ID (test for a specific version of  $\mu$ Core)

# Jump Instructions: Code RAM addressing modes

## ✓ Relative addressing (-r):

- The Code line address is expressed as an offset (-16 and 15 lines) relative to the current code line
- The operation requires only 1 instruction only and is preferred when applicable.
- Example:

```
idle_diag:          jocr idle_diag_fail _sclv;    * jump offset +1 *
                   jmpr idle_diag;           * jump offset -1 *

idle_diag_fail:    reqi 1;
```

## ✓ Absolute addressing (-f):

- The code line address is expressed as an absolute value (10 bits)
- The value must be pre-charged into a jump register (jr1 or jr2)
- The operation requires 2 instruction – must be used if relative addressing is not applicable
- Example:

```
init:              stgn gain19.4 sssc;        * set the maximum gain *
                   ldjrl tail_start;        * Jr1 = end of the actuation (tail_start) *
                   jmpf jr1;                * unconditional jump to tail_start *
```

...

\* Any number of instructions – not limited within the Code RAM range\*

```
tail_start:       endiags off off off off;
```

## Instruction “ldjrl/2 address”:

- Load the address parameter into one of the jump registers.

# Jump Instructions (con't)

Instruction “`jmprr/f dest`”:

- Unconditional jump.

Instruction “`jarrr/f dest sel`”:

- Jump using a bit of the arithmetic register as condition.

Instruction “`jcrrr/f dest sel value`”:

- Jump using a bit of the control register as condition.

Instruction “`jsrrr/f dest sel value`”:

- Jump using a bit of the status register as condition.

Instruction “`jfbkr/f dest sel value`”:

- Jump using one of voltage feedbacks as condition

Instruction “`joslr/f dest value`”:

- Jump if the start latch has the required value

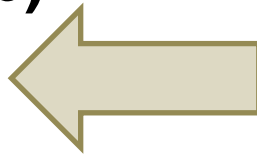
Instruction “`joidr/f dest value`”:

- Jump if the  $\mu$ Core id is the required one

# Software - Instruction Set Overview

✓ **93 instructions in 7 categories:**

1. **Arithmetic Logic Unit (27)**
2. **Configuration (29)**
3. **Diagnostic (3)**
4. **Interrupt and Subroutine (6)**
5. **Jump (16)**
6. **Load (9)**
7. **Wait (3)**



## 9 Load Instructions

1. ***cp*** – copy source to destination register
2. ***ldca*** – load counter from register and set outputs
3. ***ldcd*** – load counter from DRAM and set outputs
4. ***ldirh*** – load immediate register high-byte
5. ***ldirl*** – load immediate register low-byte
6. ***ldjr1*** – load jump register 1
7. ***ldjr2*** – load jump register 2
8. ***load*** – load data from DRAM to register
9. ***store*** – store data from register to DRAM

# Load Instructions: Internal Registers and Data RAM

Instruction “`cp source dest`”:

- Copy the value from a register to another, both of which must be in the internal memory map.

Instruction “`load source dest ofs`”:

- Copy the value from the Data RAM to a register in the internal memory map. For the Data RAM address only, an offset can added to the requested address (Address Base).

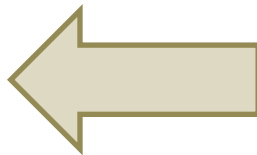
Instruction “`store source dest ofs`”:

- Copy the value from a register in the internal memory map to the Data RAM. For the Data RAM address only, an offset can added to the requested address (Address Base).

# Software - Instruction Set Overview

## ✓ 93 instructions in 7 categories:

1. Arithmetic Logic Unit (27)
2. Configuration (29)
3. Diagnostic (3)
4. Interrupt and Subroutine (6)
5. Jump (16)
6. Load (9)
7. Wait (3)



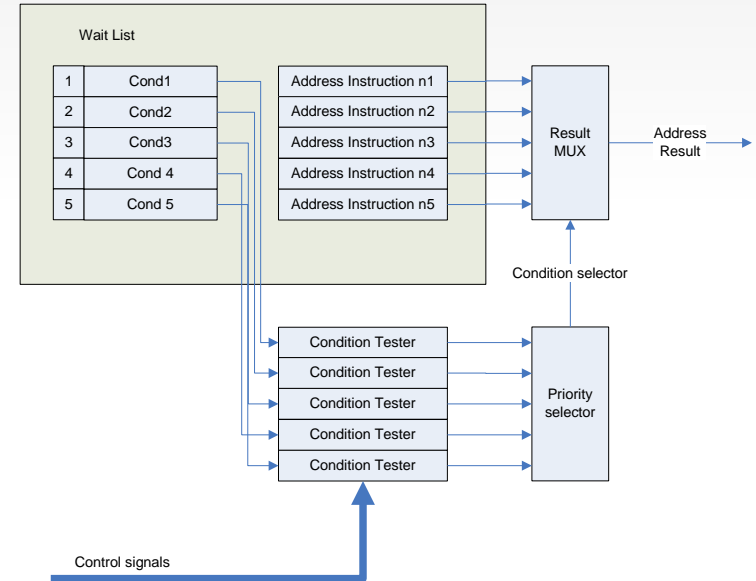
## 3 Wait Instructions

1. ***cwef*** – create wait table entry far
2. ***cwer*** – create wait table entry relative
3. ***wait*** – wait until a condition is verified



# Flow Control Instructions: Wait

- Each  $\mu$ Core has a 5 row “wait table”.
- Each row can be loaded with a condition and a destination (“cwer/f”).
- When a “wait” instruction is executed, the flow is stopped until one of the enabled condition is true. The flow resumes at the address in the same row of the fulfilled condition.



## Instruction “cwer/f dest cond row”:

- Load the selected line in the wait table with the destination address and condition specified as parameters.

## Instruction “wait mask”:

- Stop the code execution and enables the row of the wait table specified by mask parameter. When the condition on one of the enabled rows is met, the code execution resumes at the corresponding address.

```
cwer peak_cl_high tc2 row1;
cwer hold_start tc1 row2;
cwer peak_cl_low ocur row3;
```

```
peak_cl_low: ldcd rst_ofs off on TbOFF c2; * turn off HS and load TbOFF in counter 2 *
wait row12; * wait until row1: counter 2 tc *
* row2: peak phase is finished *
peak_cl_high: ldcd rst_ofs on on 0 c2; * turn on HS *
wait row23; * wait until row3: current above threshold *
* row2: peak phase is finished *
```

