

## Application Note

# Interfacing the CS5525/6/9 to the 68HC05

By Keith Coffey

### INTRODUCTION

This application note details the interface of Crystal Semiconductor's CS5525/6/9 Analog-to-Digital Converter (ADC) to a Motorola 68HC05 microcontroller. This note takes the reader through a simple example describing how to communicate with the ADC. All algorithms discussed are included in the **Appendix** at the end of this note.

### ADC DIGITAL INTERFACE

The CS5525/6/9 interfaces to the 68HC05 through either a three-wire or a four-wire interface. Figure 1 depicts the interface between the two devices. Though this software was written to interface to the *SPI<sup>TM</sup>* on the 68HC05, the algorithms can be easily modified to work in the four-wire format.

The ADC's serial port consists of four control lines:  $\overline{CS}$ , SCLK, SDI, and SDO.

$\overline{CS}$ , Chip Select, is the control line which enables access to the serial port.

SCLK, Serial Clock, is the bit-clock which controls the shifting of data to or from the ADC's serial port.

SDI, Serial Data In, is the data signal used to transfer data from the 68HC05 to the ADC.

SDO, Serial Data Out, is the data signal used to transfer output data from the ADC to the 68HC05.

### SOFTWARE DESCRIPTION

This note presents algorithms to initialize the 68HC05 and the CS5525/6/9, perform a self-offset calibration, modify the CS5525/6/9's gain register, and acquire a conversion. Figure 2 depicts a block

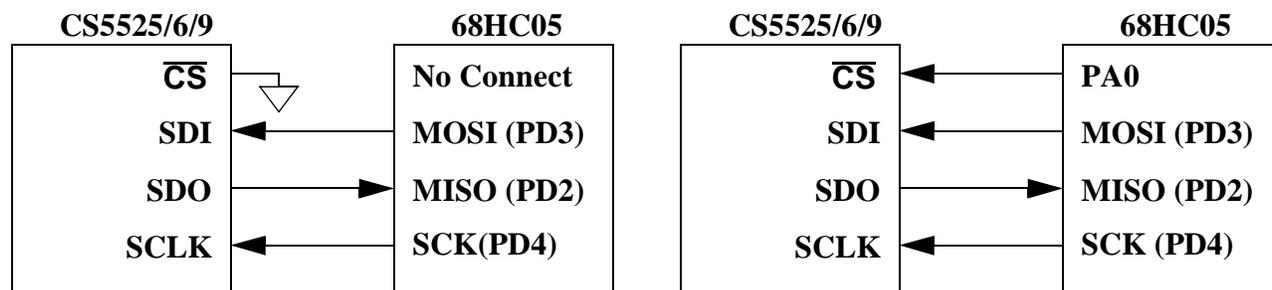


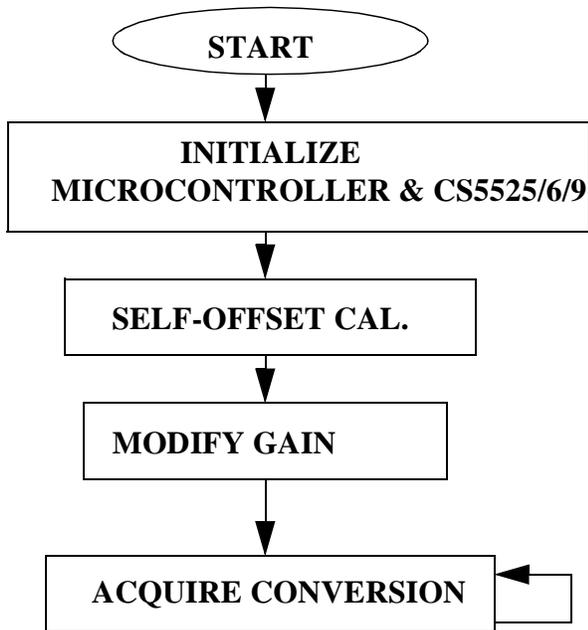
Figure 1. 3-Wire and 4-Wire Interfaces

diagram. While reading this application note, please refer to the **Appendix** for the code listing.

**Initialize**

*Initialize* is a subroutine that configures the *SPI™* on the 68HC05 and places the CS5525/6/9 in the command-state. Figure 1 depicts how the interface is configured (for more information on configuring the *SPI™* refer to Motorola’s M68HC05 Application Guide). After configuring the *SPI™*, the controller enters a delay state to allow time for the CS5525/6/9’s power-on-reset and oscillator to start-up (oscillator start-up time is typically 500 mS). The last step is to reinitialize the serial port on the ADC (reinitializing the serial port is unnecessary here, the code was added for demonstration purposes only). This is implemented by sending the converter sixteen bytes of logic 1’s followed by one final byte, with its LSB at logic 0. Once sent, the sequence places the serial port of the ADC into the command-state, where it awaits a valid command.

After returning to *main*, the software demonstrates how to calibrate the converter’s offset.



**Figure 2. CS5525/6/9 Software Flowchart**

**Self-Offset Calibration**

*Calibrate* is a subroutine that calibrates the converter’s offset. *Calibrate* first sends 0x000001 (Hex) to the configuration register. This instructs the converter to perform a self-offset calibration. Then the Done Flag (DF) bit in the configuration register is polled until set. Once DF is set, it indicates that a valid calibration is performed. To minimize digital noise (while performing a calibration or a conversion), many system designers may find it advantageous to add a software delay equivalent to a conversion or calibration cycle before polling the DF bit.

**Read/Write Gain Register**

To modify the gain register the command-byte and data-byte variables are first initialized. This is accomplished with the LDA and STA opcodes. The subroutine *write\_register* uses these variables to set the contents of the gain register in the CS5525/6/9 to 0x800000 (HEX). To do this, *write\_register* calls *send\_spi* four times (once for the command-byte and three additional times for the 24 bits of data). *Send\_spi* is a subroutine used to transfer a byte of information from the 68HC05 to the CS5525/6/9 via the *SPI™*. A byte is transferred one bit at a time, MSB (most significant bit) first. Figure 3 depicts the timing diagram for the write-cycle in the CS5525/6/9’s serial port. This algorithm demonstrates how to write to the gain register. It does not perform a gain calibration. To perform a gain calibration, follow the procedures outlined in the data sheet.

To verify if 0x800000(HEX) was written to the gain register, *read\_register* is called. It duplicates the read-cycle timing diagram depicted in Figure 4. *Read\_register* calls *send\_spi* once to transfer the command-byte to the CS5525/6/9. This places the converter into the data-state where it waits until data is read from its serial port. *Read\_register* calls *receive\_spi* three times and transfers three bytes of

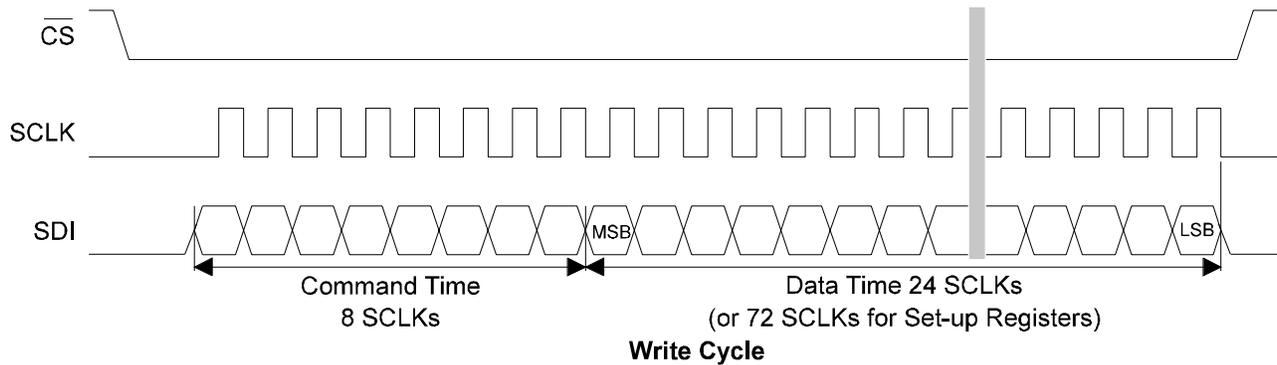
information from the CS5525/6/9 to the 68HC05 via the *SPI*<sup>TM</sup>. Similar to *send\_spi*, *receive\_spi* receives a byte one bit at a time MSB first. When the transfer is complete, highbyte, midbyte, and low-byte byte contain the CS5525/6/9's 24-bit gain register.

**Acquire Conversion**

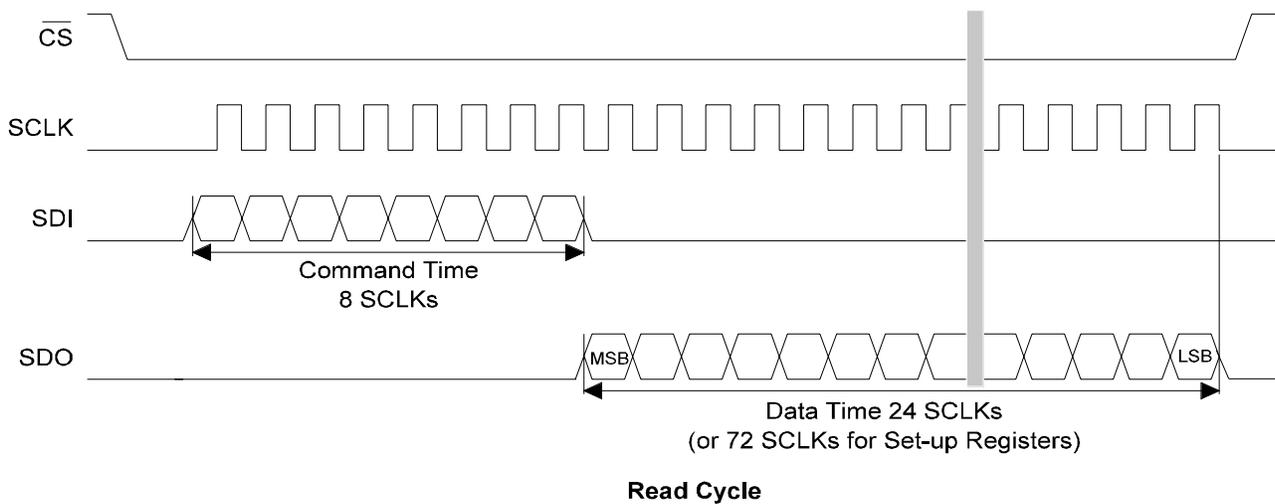
To acquire a conversion the subroutine *convert* is called. *Convert* sends the command-byte 0x0C to the converter instructing it to perform a single conversion. Then the Done Flag bit in the configuration register is polled. When DF is set, it indicates

that a conversion was performed. The 68HC05 then reads the conversion data register to acquire the conversion. Figure 6 depicts how 16-bit and 20-bit conversion words are stored in the 68HC05.

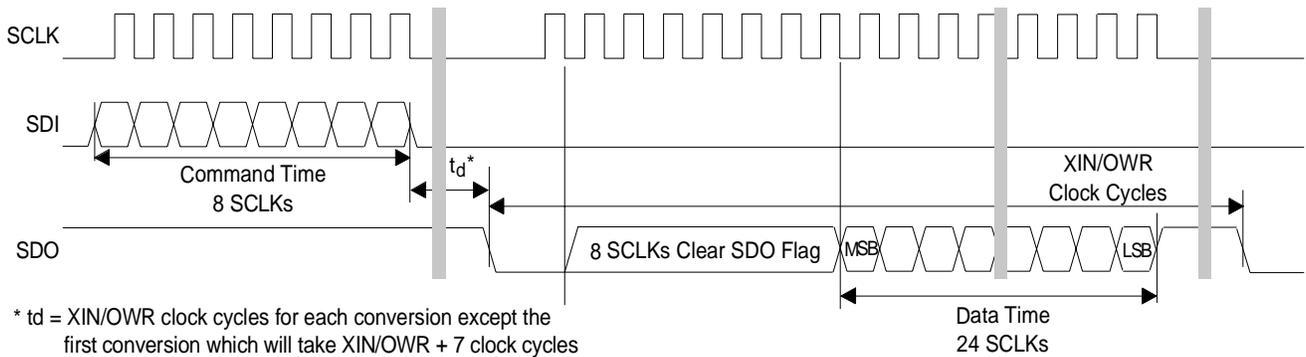
An alternate method can be used to acquire a conversion. By setting the Port Flag bit (PF, the fifth bit in the configuration register), SDO's function is modified to fall to logic 0 when a conversion is complete (refer to Figure 5). By tying SDO to the controller's interrupt pin, conversions can be acquired via an interrupt service routine.



**Figure 3. Write-Cycle Timing**



**Figure 4. Read-Cycle Timing**



**Data SDO Continuous Conversion Read (PF bit = 1)**

**Figure 5. Conversion/Acquisition Cycle with the PF Bit Asserted**

<b>.MSB</b>								<b>High-Byte</b>							
D19	D18	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4
<b>Mid-Byte</b>															
D3	D2	D1	D0	0	0	OD	OF								

**A) 20-Bit Conversion Data Word**

<b>MSB</b>								<b>High-Byte</b>							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
<b>Mid-Byte</b>															
<b>Low-Byte</b>															
1	1	1	1	0	0	OD	OF								

**B) 16-Bit Conversion Data Word**

**0**- always zero, **1**- always one,  
**OD** - Oscillation Detect, **OF** - Overflow

**Figure 6. Bit Representation/Storage in 68HC05**

**MAXIMUM SCLK RATE**

A machine cycle in the 68HC05 consists 2 oscillator periods or 500 ns if the microcontroller’s oscillator frequency is 4 MHz. Since the CS5525/6/9’s maximum SCLK rate is 2MHz, additional no operation (NOP) delays may be necessary to reduce the

transfer rate if the microcontroller system requires higher rate oscillators.

**CONCLUSION**

This application note presents an example of how to interface the CS5525/6/9 to the 68HC05. It is divided into two main sections: hardware and software. The hardware section illustrates both a three-wire and a four-wire interface. The three-wire interface is *SPI™* and *MICROWIRE™* compatible. The software section illustrates how to initialize the converter and microcontroller, calibrate the converters offset, write to and read from the ADC’s internal register, and acquire a conversion. The software is modularized and illustrates important subroutines, e.g. *write\_register* and *read\_register*. The software described in this note is included in the **Appendix** at the end of this document.

*SPI™* is a trademark of Motorola.

*MICROWIRE™* is a trademark of National Semiconductor.

**APPENDIX****68HC05 Microcode to Interface to the CS5525/6/9**

\*\*\*\*\*

\*

\* File: 55266805.asm  
 \* Date: November 1, 1996  
 \* Programmer: Keith Coffey  
 \* Revision: 0

\*

\* Processor: 68HC05

\*

\* Program entry point at routine "main". The entry point is address \$100

\*\*\*\*\*

\*

\* This program is designed as an example of interfacing a 68HC05 to a CS5525/6/9  
 \* ADC. The program interfaces via SPI (i.e. port D) which controls the  
 \* serial communications, calibration, and conversion signals. Other ADC's  
 \* (16-bit and 20-bit) in the product family can be used.

\*\*\*\*\*

\*\*\*\*\* Memory Map Equates

PORTA	EQU	\$00	; General Purpose I/O Port
DDRA	EQU	\$04	; Data Direction Control For Port A
SPCR	EQU	\$0A	; Serial Peripheral Control Register
SPSR	EQU	\$0B	; Serial Peripheral Status Register
SPDR	EQU	\$0C	; Serial Peripheral Data I/O Register
SPIF	EQU	7	; Serial Peripheral Data Transfer Flag

\*\*\*\*\* RAM Values

ORG \$50

\*\*\*\*\* Ram Memory Equates

HIGHBYTE	RMB	1	; Upper 8 bits of Conversion Register
MIDBYTE	RMB	1	; Middle 8 bits of Conversion Register
LOWBYTE	RMB	1	; Lowest 8 Bits of Conversion Register
COMMANDBYTE	RMB	1	; One byte RAM storage location

```
*****
*   Program Code
*****
          ORG          $0100
*****
* Routine - Main
* Input   - none
* Output  - none
* This is the entry point to the program.
*****
MAIN      EQU          *                ; Start from Reset Vector
*****   Initialize System and Perform SELF OFFSET Calibration
          JSR          initialize        ; Initialize the system
          JSR          calibrate         ; Calibrate the ADC Offset
*****   Write to the GAIN Register
          LDA          #$82              ; Prepare COMMANDBYTE
          STA          COMMANDBYTE
          LDA          #$80              ; Prepare HIGHBYTE
          STA          HIGHBYTE
          CLR          MIDBYTE           ; Prepare MIDBYTE
          CLR          LOWBYTE           ; Prepare LOWBYTE
          JSR          write_register     ; Write to Gain Register
*****   Read from the GAIN Register
          LDA          #$92              ; Prepare COMMANDBYTE
          STA          COMMANDBYTE
          JSR          read_register      ; Read the Gain Register
*****   Perform Single Conversions
LOOP      JSR          convert           ; Convert Analog input
          JMP          LOOP              ; Repeat Loop
*****   End MAIN
```

\*\*\*\*\*

\* Subroutines

\*\*\*\*\*

\*\*\*\*\*

\* Routine - initialize

\* Input - none

\* Output - none

\* This subroutine initializes port D for interfacing to the CS5525/6/9 ADC.

\* It provides a time delay for oscillator start-up/wake-up period.

\* A typical start-up time for a 32768 Hz crystal, due to high Q, is 500 ms.

\* Also 1003 XIN clock cycles are allotted for the ADC's power on reset.

\*\*\*\*\*

```

initialize    LDA        #%01010000        ; Load ACCA with for SPSC
              STA        SPCR          ; Setup SPI
              LDA        #40           ; Load ACCA with delay count
              JSR        delay         ; Delay, Power on Reset 1003 XIN
              LDA        #220          ; Load ACCA with delay count
              JSR        delay         ; Delay, Oscillator start-up 170 mS
              JSR        delay         ; Delay, Oscillator start-up 170 mS
              JSR        delay         ; Delay, Oscillator start-up 170 mS
              LDX        #$0F          ; Reset Serial Port on ADC
              LDA        #$FF          ; Load ACCA with $FF
loop          JSR        send_spi       ; Move $FF to SPDR
              DECX         ; Decrement the counter
              BNE        loop          ; Repeat loop is counter not zero
              LDA        #%11111110   ; Load ACCA with last byte
              JSR        send_spi       ; Move $FE to SPDR
              RTS          ; Exit subroutine
  
```

\*\*\*\*\*

\* Routine - calibrate

\* Input - none

\* Output - none

\* This subroutine instructs the CS5525/6/9 to perform self-calibration.

\*\*\*\*\*

```

calibrate    LDA        #$84           ; set command byte for config write
              STA        COMMANDBYTE   ; set COMMAND BYTE
              CLR        HIGHBYTE       ; clear HIGHBYTE
              CLR        MIDBYTE        ; clear MIDBYTE
              LDA        #$01           ; get ready for self offset cal
              STA        LOWBYTE        ; set LOWBYTE
              JSR        write_register  ; Write to Config Register

              LDA        #$94           ; set command byte for config read
              STA        COMMANDBYTE   ; set COMMAND BYTE
poll_done:   JSR        read_register    ; Poll done flag until cal complete
              BRCLR     3,LOWBYTE,poll_done; repeat if flag not set
              RTS          ; Exit subroutine
  
```

\*\*\*\*\*

```
* Routine - convert
* Input   - none
* Output  - Conversion results in memory locations HIGHBYTE, MIDBYTE and
*          LOWBYTE. This algorithm performs only single conversions. If
*          continuous conversions are needed the routine needs to be
*          modified. Port flag is zero.
```

```
*          HIGHBYTE    MIDBYTE    LOWBYTE
*          7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
* 16-bit results MSB                                LSB 1 1 1 1 0 0 OD OF
* 20-bit results MSB                                LSB 0 0 OD OF
```

```
* This subroutine initiates a single conversion.
```

\*\*\*\*\*

```
convert    LDA  #$C0                                ; Set COMMANDBYTE for single CONV
           STA  COMMANDBYTE
           JSR  send_spi                            ; Transmit command out SPI
           LDA  #$94                                ; Set command byte for config read
           STA  COMMANDBYTE                        ; Send COMMAND BYTE
done1      JSR  read_register                        ; Poll done flag until CONV complete
           BRCLR 3,LOWBYTE,done1                   ; Repeat if Done Flag not Set

           LDA  #$96                                ; Set Byte to Read Conversion Reg.
           STA  COMMANDBYTE                        ; Store COMMAND BYTE
           JSR  read_register                        ; Acquire the Conversion
           RTS                                     ; Exit subroutine
```

\*\*\*\*\*

```
* Routine - write_register
* Input   - COMMANDBYTE, HIGHBYTE, MIDBYTE, LOWBYTE
* Output  - none
* This subroutine instructs the CS5525/6/9 to write to an internal register.
```

\*\*\*\*\*

```
write_register LDA  COMMANDBYTE                    ; Load ACCA with COMMANDBYTE
              JSR  send_spi                        ; transfer byte
              LDA  HIGHBYTE                        ; Load ACCA with HIGHBYTE
              JSR  send_spi                        ; transfer byte
              LDA  MIDBYTE                         ; Load ACCA with MIDBYTE
              JSR  send_spi                        ; transfer byte
              LDA  LOWBYTE                         ; Load ACCA with LOWBYTE
              JSR  send_spi                        ; transfer byte
              RTS                                  ; Exit Subroutine
```

\*\*\*\*\*

\* Routine - read\_register  
\* Input - COMMANDBYTE  
\* Output - HIGHBYTE, MIDBYTE, LOWBYTE  
\*

\* This subroutine reads an internal register of the ADC.

\*\*\*\*\*

```
read_register  LDA      COMMANDBYTE      ; Load ACCA with COMMANDBYTE
               JSR      send_spi         ; transfer byte
               JSR      receive_spi      ; receive byte
               STA      HIGHBYTE        ; Move ACCA with HIGHBYTE
               JSR      receive_spi      ; receive byte
               STA      MIDBYTE         ; Move ACCA with MIDBYTE
               JSR      receive_spi      ; receive byte
               STA      LOWBYTE         ; Move ACCA with LOWBYTE
               RTS                       ; Exit Subroutine
```

\*\*\*\*\*

\* Routine - send\_spi  
\* Input - Byte to be transmitted is placed in ACCA  
\* Output - none  
\*

\* This subroutine sends a byte to the ADC.

\*\*\*\*\*

```
send_spi:     STA      SPDR              ; Move ACCA to SPDR
wait0         BRCLR   SPIF,SPSR,wait0    ; Loop until byte is transmitted
               RTS                       ; Exit Subroutine
```

\*\*\*\*\*

\* Routine - receive\_spi  
\* Input - none  
\* Output - Byte received is placed in ACCA  
\* This subroutine receives a byte from the ADC.

\*\*\*\*\*

```
receive_spi:  CLRA                      ; Load ACCA register with Zero
               STA      SPDR              ; Initiate a transfer of all Zero's
wait1         BRCLR   SPIF,SPSR,wait1    ; Reset Flag SPIF bit
               LDA      SPDR              ; Move SPDR to ACCA
               RTS                       ; Exit Subroutine
```

\*\*\*\*\*

\* Routine - delay  
 \* Input - Count in register A  
 \* Output - none  
 \*

\* This subroutine delays by using count from register A. The 68HC05  
 \* development board uses a 4.0MHz clock (E = 2.0 MHz), thus each cycle is  
 \* 500 nS. This delay is equivalent to (500ns)\*(1545)\*(count value),  
 \* (a count of 720 provides a 556ms delay).

\*\*\*\*\*

```

delay
outlp      CLRX                ; X used as inner loop count
innlp      DECX                ; 0-FF, FF-FE, FE-FD, ....1-0 256 loops
           NOP                 ; 2 cycles
           NOP                 ; 2 cycles
           BNE innlp           ; 10 cycles*256*500ns=1.28 ms
           DECA                ; Countdown the accumulator
           BNE outlp           ; 2569 cycles*500ns*A
           RTS                 ; Exit subroutine
  
```

\*\*\*\*\*

\* Interrupt Vectors

\*\*\*\*\*

```

NOT_USED  RTI
           ORG $1FF4           ; Interrupt Vectors
           FDB NOT_USED       ; SPI Interrupt
           FDB NOT_USED       ; SCI Interrupt
           FDB NOT_USED       ; Timer Interrupt
           FDB NOT_USED       ; IRQ Interrupt
           FDB NOT_USED       ; SWI Interrupt
           FDB MAIN           ; Reset interrupt- power on reset
  
```

• Notes •

---

SMART  
Analog™