Ernest Gonzales
Associate Applications Engineer,
Signal Conditioning Products
Phone: (408) 432-1900 Ext: 3477
Email: egonzales@linear.com

Getting Started:
LTC6804-2 Programming Guide.

The purpose of this guide is to provide example working microcontroller C code to communicate to a 3 stack of LTC6804-2 Parallel-Addressable Devices. To develop this code a DC1941B LTC6820 isoSPI Serial Interface demoboard and a set of 3 DC1942B LTC6804-2 Addressable isoSPI Battery-Stack Monitor demoboards are used as a representative battery monitoring system.

Outline:

1) Connecting 3 boards in a Parallel-Addressable isoSPI Bus and jumper settings for the boards.

2) Setting the correct SPI Modes.

3) LTC6804 Core Operational States vs isoSPI Port Operational States.
   - 3a) Introduction.
   - 3b) Details and Terminology.
   - 3c) Sequence Tables.
     - Table 5: CORE==SLEEP Sequence Tables.
     - Table 6: CORE==STANDBY Sequence Tables.
     - Table 7: CORE==REFUP Sequence Tables.

4) Function to create isoSPI Wake-Up byte and delay before each command.

5) SPI command usage with scope photos:
   - 5a) Address Read Configuration (RDCFG) to confirm everything is connected.
   - 5b) Address Write Configuration (WRCFG) to individually set each device flag thresholds and reduce ADC measurement time.
   - 5c) Broadcast Write Configuration (WRCFG) simultaneously set all devices to indentical flag thresholds and reduce ADC measurement time.
   - 5d) Address Start Cell Voltage ADC Conversion and Poll Status (ADCV) to have one device measure cells.
   - 5e) Broadcast Start Cell Voltage ADC Conversion (ADCV) to have all devices simultaneously measure cells.
   - 5f) Address Poll ADC Status (PLADC) to perform other tasks during an ADC Conversion of one device then return to confirm ADC Conversions have completed.
   - 5g) Address Read Cell Voltage Register Group A (RDCVA) to individually read Cells 1 to 3 results from each device.
   - 5h) Typical ADCV, PLADC, and RDCVA command usage together.
   - 5i) Address Clear Cell Voltage Registers (CLRCELL) to clear one device Cell Voltage Registers to ones.
   - 5j) Broadcast Clear Cell Voltage Registers (CLRCELL) to simultaneously clear the Cell Voltage Registers of all devices to ones.

1) Connecting 3 boards in a Parallel-Addressable isoSPI Bus and jumper settings for the boards.

The following examples use a configuration of 3 stacked LTC6804-2 devices in full isoSPI communication mode: bottom (B), middle (M) and top (T).

DC1942B SETTINGS:	BOTTOM BRD#1	MIDDLE BRD#2	TOP BRD#3
JUMPERS	2-WIRE isoSPI	2-WIRE isoSPI	2-WIRE isoSPI
JP1 ,2 ,3 ,4, 5 : ISOMD	1 (VREG)	1 (VREG)	1 (VREG)
JP6 : SWTEN	0 (GND)	0 (GND)	0 (GND)
JP7 : A0	1 (VREG)	0 (GND)	1 (VREG)
JP8 : A1	0 (GND)	1 (VREG)	1 (VREG)
JP9 : A2	0 (GND)	0 (GND)	0 (GND)
JP10 : A3	0 (GND)	0 (GND)	0 (GND)
JP11, 12 : POWER	BAT (CELL12)	BAT (CELL12)	BAT (CELL12)
JP13 : TERM	0 (NO TERM)	0 (NO TERM)	1 (R59 + R51)

DC1941B SETTINGS:	isoSPI MASTER
JUMPERS	2-WIRE isoSPI
JP2 : VCC	EXT (VCC)
JP3 : ENABLE	EN (VCCS)
JP4 : SLOW	0 (GND)
JP5 : MODE	MASTER (VCC)
JP6 : VCCS	VCC (VCC)
JP7 : PHASE	1 (VCC)
JP8 : POLARITY	1 (VCC)
JP9, 10 : VTHRESHOLD	VTH2 (R6 + R17)

DC1941B 4-WIRE SPI CONNECTIONS:	SPI MASTER CONNECTIONS:
JP1 PINOUT	PINOUT
1 : MISO	MISO/SDI
2 : VCC	VDD
3 : SCK	SCK
4 : MOSI	MOSI/SDO
5 : CS	CS/SS
6 : GND	VSS

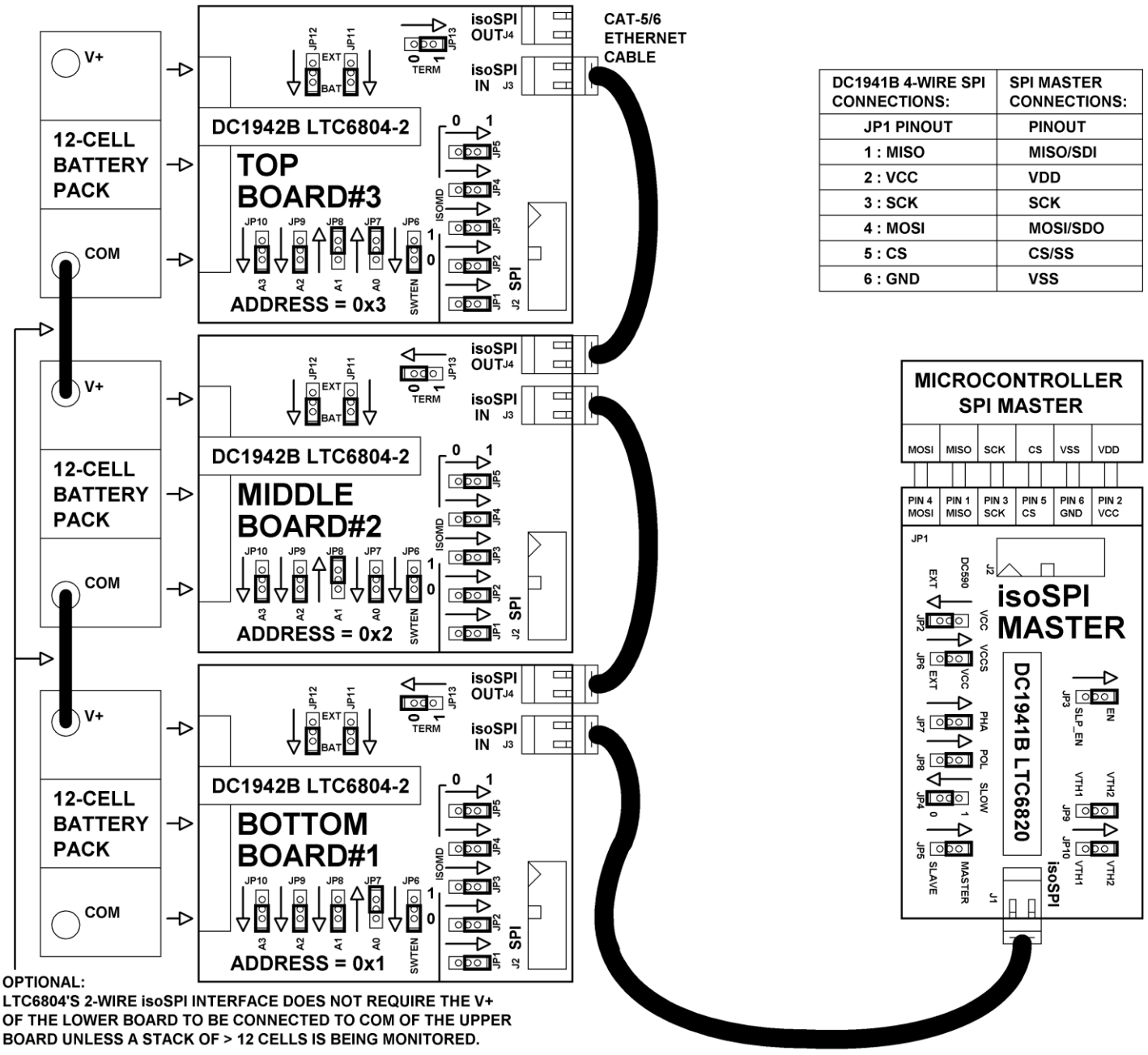


Figure 1: Full isoSPI Communications setup with 3 x DC1942B LTC6804-2, 1 x DC1941B LTC6820, and 1 x Microcontroller

2) Setting the correct SPI Modes.

The Clock Polarity (CPOL) can be set to either idle normally low or idle normally high based on preference. However the Clock Phase must be correctly set to latch onto the rising edge data only.

Incorrectly setting the Clock Phase (CPHA) will latch onto the falling edge data and have the following issues:
1) Valid SPI reads require a slow SPI Clock speed such as 100kHz. Otherwise SPI Clock speeds greater than 100kHz will always result with invalid SPI reads due to the data being latched onto the falling edge instead of the rising edge.
2) No response or SPI reads of ones or 0xF's when SPI Clock speed approaches 1MHz or the SCK Period (tCLK) of 1uS minimum.

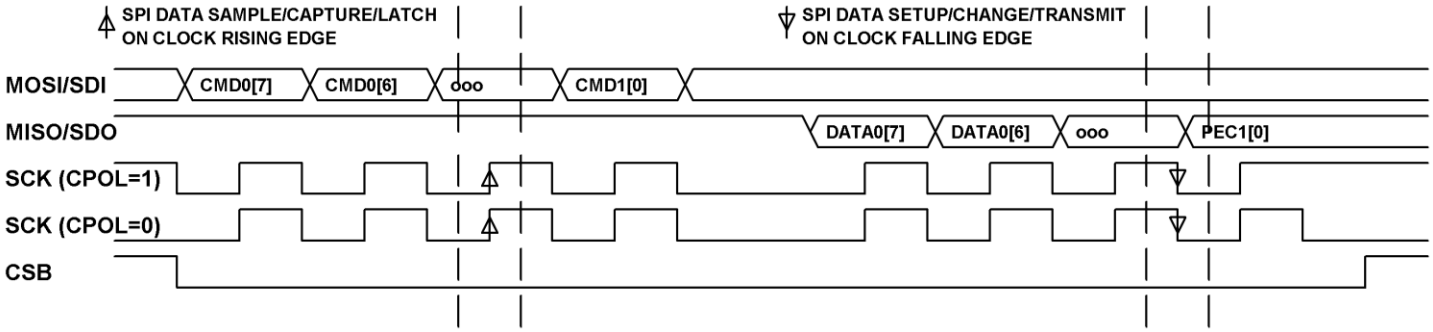


Figure 2: Clock Phase (CPHA) rising edge data latch vs Clock Polarity (CPOL)

Table 1: Compatible Atmel, Freescale, NXP, Silicon Labs, and ST Microcontroller SPI Mode Settings

CPOL	CPHA	Clock Polarity	Leading Edge	Trailing Edge	SPI Mode
0	0	Idles Low	Data Sample/ Capture/Latch(Rising)	Data Setup/Change/ Transmit (Falling)	0
1	1	Idles High	Data Setup/Change/ Transmit (Falling)	Data Sample/ Capture/Latch(Rising)	3

Table 2: Compatible Microchip Microcontroller SPI Mode Settings

CKP	CKE	Clock Polarity (CKP)	Clock Edge Select (CKE)	CPOL	CPHA	SPI Mode
0	1	Idles Low	Data transmitted on falling edge of SCK	0	0	0
1	0	Idles High	Data transmitted on falling edge of SCK	1	1	3

Table 3: Compatible Renesas Microcontroller Clocked Serial Interface (CSI) to SPI Mode Settings

CKPmn	DAPmn	Clock Phase (CKP)	Data Phase (DAP)	CPOL	CPHA	SPI Mode
1	1	Idles Low	Data latched on rising edge of SCK	0	0	0
0	0	Idles High	Data latched on rising edge of SCK	1	1	3

3) LTC6804 Core Operational States vs isoSPI Port Operational States.

3a) Introduction.

Consistently successful communication and control requires knowledge of the LTC6804 Operational States.

The operation of the LTC6804 is divided into two separate sections: the Core circuit (CORE) and the isoSPI circuit (ISOSPI). The Core and isoSPI circuits manage the LTC6804 device power consumption based on the operational states required. The isoSPI circuit manages only the isoSPI Port communications while the Core circuit is the main controller of the ADCs, MUXes, VREG, VREF, etc. Both sections have an independent set of operating states, start-up/wake-up times and shutdown timeouts.

When using 2-wire isoSPI communication, any differential signal activity above the Threshold-Setting Voltage on ICMP Pin (VICMP) prevents the LTC6804 device from remaining in the lowest power states (ISOSPI==IDLE and CORE==SLEEP).

When using only 4-wire SPI communication, the isoSPI Port Operational States and additional power consumption are ignored. The Core can only transistion from STANDBY to REFUP State with a valid START ADC COMMAND or a valid WRCFG COMMAND setting REFON=1.

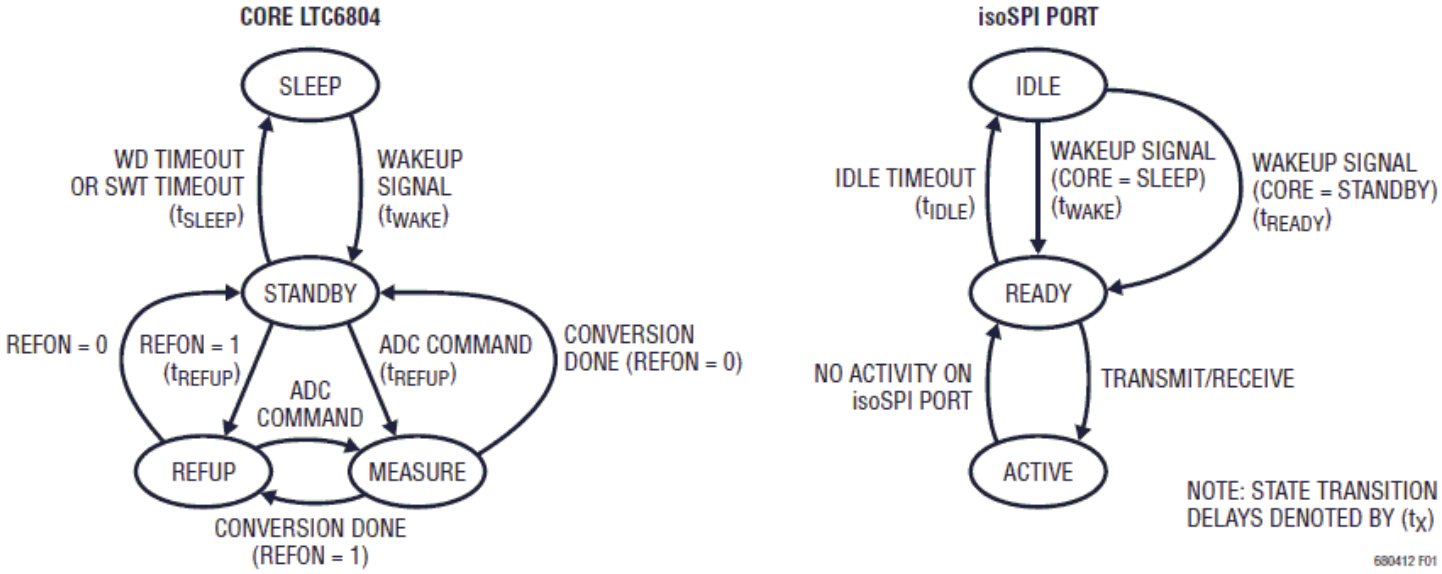


Figure 3: LTC6804 Operational State Diagram

Table 4: Core and isoSPI Port Timing Specifications

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS
tWAKE	Regulator Start-Up Time		100	300	uS
tREFUP	Reference Wake-Up Time State: Core = STANDBY State: Core = REFUP	 2.7	 3.5	 4.4 0	 mS mS
tSLEEP	Watchdog Timer SWTEN Pin = 0 or DCTO[3:0] = 0000	 1.8	 2.0	 2.2	 sec
tREADY	Start-Up Time After Wake Detection			10	uS
tIDLE	Idle Timeout Duration	4.3	5.5	6.7	mS

3b) Details and Terminology.

CORE AND isoSPI PORT START-UP/WAKE-UP TIME:

tWAKE = Voltage Regulator Start-Up Time

Time after a wake-up signal that the Core transitions from SLEEP to STANDBY State.

DRIVE pin powers up VREG input pin through external transistor or external supply.
Watchdog Timer starts running.
VREF 1 and 2 output pins are off.

Time after a wake-up signal isoSPI transitions from IDLE to READY State when CORE==SLEEP.

isoSPI Port powers up through VREG input pin.
isoSPI IDLE Timer has started running.
isoSPI Port communication take the longest to start.

CORE START-UP/WAKE-UP TIME:

tREFUP = Voltage Reference Wake-Up Time.

Time VREF (VREF1 and VREF2) output pins to settle.

If CORE==SLEEP and a START ADC Command is received,

ADC Conversions take the longest time to start.
Core power consumption is the lowest with VREG and VREF normally powered off.
Conversion Time (CORE==SLEEP) = tWAKE + tREFUP + tCYCLE

If CORE==STANDBY and a START ADC Command is received,

ADC Conversions take a short time to start.
Core power consumption is low with VREG normally on and VREF normally powered off.
Conversion Time (CORE==STANDBY) = tREFUP + tCYCLE

If CORE==REFON and a START ADC Command is received,

ADC Conversions take the shortest time to start.
Core power consumption is the highest with VREG and VREF normally powered on.
Conversion Time (CORE==REFUP) = tCYCLE

isoSPI PORT START-UP TIMES:

tREADY = Start-Up Time After Wake Detection

Time after a wake-up signal isoSPI transitions from IDLE to READY State when CORE==STANDBY or REFUP.

VREG input pin is still supplying power to isoSPI Port.
isoSPI IDLE Timer has started running.
isoSPI Port communication takes a short time to start.

CORE SHUTDOWN TIMEOUT:

tSLEEP = Watchdog Timeout

Time Core powers down to the lowest power consumption.

Watchdog Timer expired.
DRIVE pin powers down, no external transistor or external supply 5V to VREG input pin.
VREF 1 and 2 output pins are powered down.
isoSPI IDLE Timer expired.
isoSPI Port is powered down.

isoSPI Port SHUTDOWN TIMEOUT:

tIDLE = Idle Timeout Duration

Time isoSPI Port powers down to the lower power consumption.

isoSPI IDLE Timer expired.
isoSPI Port is powered down.

3c) Sequence Tables.

When writing commands to the LTC6804 a sequence of events in various sections of the circuitry occur. The Tables that follow illustrate what happens when the device is awakened, accepts a command, then returns to a lower power state. The sections of circuitry are shown in Figure 4.

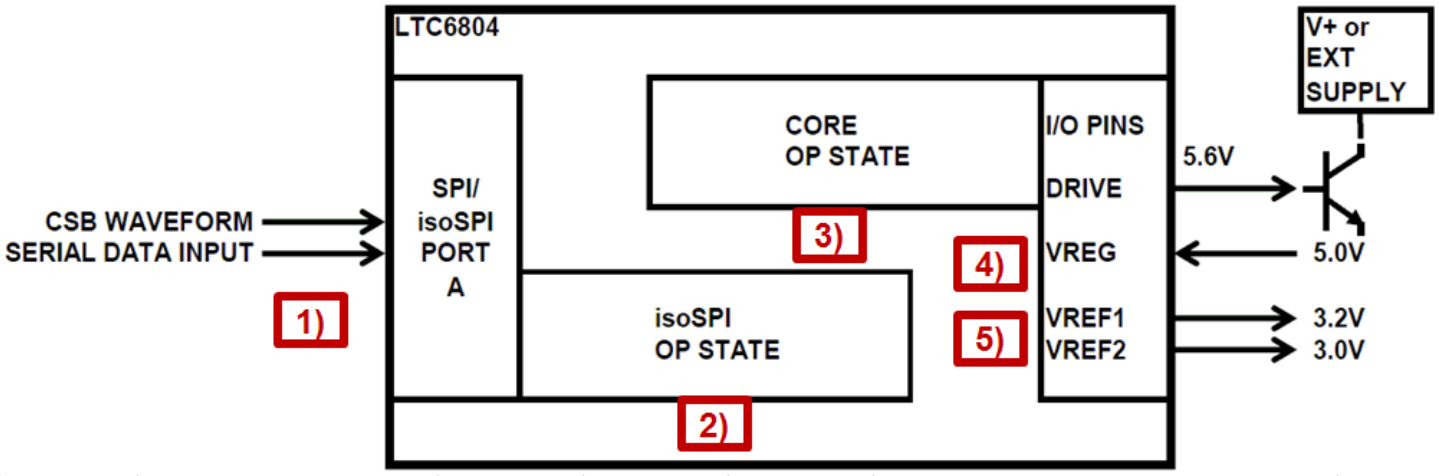


Figure 4. Circuit Sections of Interest for Command Processing. Refer to Figure 3 for the Core (3), and isoSPI port (2) State Diagrams.

Before sending any command three conditions must be known to create the appropriate **Wake-up byte and Delay time intervals.** These conditions are:

1. Current CORE Operational State: SLEEP, STANDBY or REFUP.
2. Current isoSPI PORT Operational State: IDLE, READY or ACTIVE.
3. Type of command to be sent: NON- ADC command, WRCFG (to set REFON=1) or START ADC command

The following summarizes these categories of commands.

NON-START ADC COMMANDS or NON-ADC COMMANDS:
WRCFG, RDCFG, RDCVA, RDCVB, RDCVC, RDCVD, RDAUXA, RDAUXB, RDSTATA, RDSTATB, CLRCELL, CLRAUX, CLRSTAT, PLADC, DIAGN, WRI2C, RDI2C, and STI2C.

START ADC COMMANDS or ADC COMMANDS:
ADCV, ADOW, CVST, ADCVAX, ADAX, AXST, ADSTAT, and STATST.

Depending on the type of command issued the state of circuit sections will vary. Tables 5, 6 and 7 show the sequence in time from left to right of events in each circuitry section with each of the three possible current CORE states and each of the three command types.

Table 5: CORE==SLEEP Sequence Tables.

CORE IS CURRENTLY IN SLEEP STATE (CORE==SLEEP):

Table 5a: WILL SEND NON-START ADC COMMAND:

	SERIAL DATA TYPE	WAKE-UP		VALID COMMAND				
1)	CSB WAVEFORM	HIGH	LOW	HIGH	LOW		HIGH	
	SERIAL DATA INPUT	NONE	0xXX	NONE	NON-START ADC		NONE	
2)	isoSPI OP STATE	IDLE		IDLE >>> tWAKE*	READY***	READY >>> tIDLE		IDLE
3)	CORE OP STATE	SLEEP REFON==0		SLEEP >>> tWAKE	STANDBY		STANDBY >>> tSLEEP	
4)	VREG INPUT	OFF		DRIVE PIN POWER-UP		ON		SLEEP
5)	VREF1 & 2 OUTPUT				OFF			OFF

Table 5b: WILL SEND WRCFG COMMAND TO SET REFON=1:

	SERIAL DATA TYPE	WAKE-UP		VALID COMMAND				
1)	CSB WAVEFORM	HIGH	LOW	HIGH	LOW		HIGH	
	SERIAL DATA INPUT	NONE	0xXX	NONE	WRCFG REFON=1		NONE	
2)	isoSPI OP STATE	IDLE		IDLE >>> tWAKE*	READY***	READY >>> tIDLE		IDLE
3)	CORE OP STATE	SLEEP REFON==0		SLEEP >>> tWAKE	STANDBY	tREFUP	REFUP >>> tSLEEP	
4)	VREG INPUT	OFF		DRIVE PIN POWER-UP		ON		SLEEP
5)	VREF1 & 2 OUTPUT			OFF		POWER-UP	ON	OFF

Table 5c: WILL SEND START ADC COMMAND:

	SERIAL DATA TYPE	WAKE-UP		VALID COMMAND						
1)	CSB WAVEFORM	HIGH	LOW	HIGH	LOW			HIGH		
	SERIAL DATA INPUT	NONE	0xXX	NONE	START ADC			NONE		
2)	isoSPI OP STATE	IDLE		IDLE >>> tWAKE*	READY***	READY >>> tIDLE			IDLE	
3)	CORE OP STATE	SLEEP REFON==0		SLEEP >>> tWAKE	STANDBY	tREFUP	REFUP	MEASURE	STANDBY >>> tSLEEP	
4)	VREG INPUT	OFF		DRIVE PIN POWER-UP		ON		ON		SLEEP
5)	VREF1 & 2 OUTPUT			OFF		POWER-UP	ON		OFF	OFF

* WHEN CORE==SLEEP AND isoSPI==IDLE, USE THE MAXIMUM tWAKE TIME AS A DELAY BETWEEN THE WAKE-UP BYTE AND VALID COMMAND.
*** WHEN COREI=SLEEP AND isoSPI==READY, NO WAKE-UP BYTE IS NEEDED BEFORE A VALID COMMAND.

Table 6: CORE==STANDBY Sequence Tables.

CORE IS CURRENTLY IN STANDBY STATE (CORE==STANDBY):

Table 6a: WILL SEND NON-START ADC COMMAND:

	SERIAL DATA TYPE	WAKE-UP		VALID COMMAND		
1)	CSB WAVEFORM	HIGH	LOW	HIGH	LOW	HIGH
	SERIAL DATA INPUT	NONE	0xXX	NONE	NON-START ADC	NONE
2)	isoSPI OP STATE	IDLE	IDLE >>> tREADY**	READY***	ACTIVE	READY >>> tIDLE	IDLE
3)	CORE OP STATE	STANDBY REFON==0			STANDBY	STANDBY >>> tSLEEP	SLEEP
4)	VREG INPUT	ON					OFF
5)	VREF1 & 2 OUTPUT	OFF					

Table 6b: WILL SEND WRCFG COMMAND TO SET REFON=1:

	SERIAL DATA TYPE	WAKE-UP		VALID COMMAND			
1)	CSB WAVEFORM	HIGH	LOW	HIGH	LOW	HIGH	
	SERIAL DATA INPUT	NONE	0xXX	NONE	WRCFG REFON=1	NONE	
2)	isoSPI OP STATE	IDLE	IDLE >>> tREADY**	READY***	ACTIVE	READY >>> tIDLE	IDLE
3)	CORE OP STATE	STANDBY REFON==0			STANDBY	tREFUP	REFUP >>> tSLEEP	SLEEP
4)	VREG INPUT	ON					OFF
5)	VREF1 & 2 OUTPUT	OFF			POWER-UP	ON	OFF

Table 6c: WILL SEND START ADC COMMAND:

	SERIAL DATA TYPE	WAKE-UP			VALID COMMAND				
1)	CSB WAVEFORM	HIGH	LOW	HIGH	LOW	HIGH			
	SERIAL DATA INPUT	NONE	0xXX	NONE	START ADC	NONE			
2)	isoSPI OP STATE	IDLE	IDLE >>> tREADY**	READY***	ACTIVE	READY >>> tIDLE		IDLE	
3)	CORE OP STATE	STANDBY REFON==0			STANDBY	tREFUP	REFUP	MEASURE	STANDBY >>> tSLEEP	SLEEP
4)	VREG INPUT	ON							OFF
5)	VREF1 & 2 OUTPUT	OFF			POWER-UP	ON		OFF	

** WHEN COREI=SLEEP AND isoSPI==IDLE, USE THE MAXIMUM tREADY TIME AS A DELAY BETWEEN THE WAKE-UP BYTE AND VALID COMMAND.
*** WHEN COREI=SLEEP AND isoSPI==READY, NO WAKE-UP BYTE IS NEEDED BEFORE A VALID COMMAND.

Table 7: CORE==REFUP Sequence Tables.

CORE IS CURRENTLY IN REFUP STATE (CORE==REFUP):

Table 7a: WILL SEND NON-START ADC COMMAND:

	SERIAL DATA TYPE	WAKE-UP		VALID COMMAND		
1)	CSB WAVEFORM	HIGH	LOW	HIGH	LOW	HIGH
	SERIAL DATA INPUT	NONE	0xXX	NONE	NON-START ADC	NONE
2)	isoSPI OP STATE	IDLE	IDLE >>> tREADY**	READY***	READY >>> tIDLE	IDLE
3)	CORE OP STATE	REFUP REFON==1			REFUP >>> tSLEEP	SLEEP
4)	VREG INPUT				ON	OFF
5)	VREF1 & 2 OUTPUT				ON	OFF

Table 7b: WILL SEND WRCFG COMMAND TO SET REFON=0:

	SERIAL DATA TYPE	WAKE-UP		VALID COMMAND		
1)	CSB WAVEFORM	HIGH	LOW	HIGH	LOW	HIGH
	SERIAL DATA INPUT	NONE	0xXX	NONE	WRCFG REFON=0	NONE
2)	isoSPI OP STATE	IDLE	IDLE >>> tREADY**	READY***	READY >>> tIDLE	IDLE
3)	CORE OP STATE	REFUP REFON==1			REFUP >>> tSLEEP	SLEEP
4)	VREG INPUT				ON	OFF
5)	VREF1 & 2 OUTPUT	ON			OFF	

Table 7c: WILL SEND START ADC COMMAND:

	SERIAL DATA TYPE	WAKE-UP		VALID COMMAND			
1)	CSB WAVEFORM	HIGH	LOW	HIGH	LOW	HIGH	
	SERIAL DATA INPUT	NONE	0xXX	NONE	START ADC	NONE	
2)	isoSPI OP STATE	IDLE	IDLE >>> tREADY**	READY***	READY >>> tIDLE	IDLE	
3)	CORE OP STATE	REFUP REFON==1			REFUP >>> tSLEEP		SLEEP
4)	VREG INPUT				ON		OFF
5)	VREF1 & 2 OUTPUT				ON		OFF

** WHEN CORE!=SLEEP AND isoSPI==IDLE, USE THE MAXIMUM tREADY TIME AS A DELAY BETWEEN THE WAKE-UP BYTE AND VALID COMMAND.
*** WHEN CORE!=SLEEP AND isoSPI==READY, NO WAKE-UP BYTE IS NEEDED BEFORE A VALID COMMAND.

4) Function to create isoSPI Wake-Up byte and delay before each command.

isoSPI Wake-Up Function

```
// The following isoSPI Wake-Up Function will used by all commands.
// This function will create the necessary Wake-Up Byte and delay
// before a valid command based on LTC6804 Core State vs isoSPI State.
// Parameters:
// int1_t  isoSPIMode   :  0 = 4-wire SPI used.
//                         1 = 2-wire isoSPI used.
// int1_t  isoSPI_IDLE  :  0 = isoSPI in READY State.
//                         1 = isoSPI in IDLE State.
// int1_t  core_SLEEP   :  0 = LTC6804 Core in STANDBY/REFUP State.
//                         1 = LTC6804 Core in SLEEP State.
// int1_t  daisyChain   :  0 = Addressable Parallel-Bus Communication.
//                         1 = Daisy Chain Bus Communication.
// uint8_t totalDevices : XX = Total devices connected.

void isoSPI_WakeUp_Write(int1_t isoSPIMode, int1_t isoSPI_IDLE, int1_t core_SLEEP,
                         int1_t daisyChain, uint8_t totalDevices)
   {
   /*****************************************************************************
    *    Core State (LTC6804 Core Operational States):
    *    SLEEP -> STANDBY (REFON==0) -> REFUP (REFON==1) ->
    *    MEASURE (ADC Conversion running)
    *
    *    isoSPI State (isoSPI Port Operational States):
    *    IDLE -> READY -> ACTIVE
   *****************************************************************************/

   // Trigger to output a Wake-Up Byte.
    int1_t sendWakeUp           = 0;

   // Variables to set delay time ensuring isoSPI State is READY (ISOSPI==READY).
    uint16_t isoWakeUpTime      = 0;
    uint8_t devicesConnected    = 0;
    uint16_t totalisoWakeUpTime = 0;

   // Decide if Wake-Up Byte is needed. If needed, calculate total isoSPI Wake-Up Time.
   if (isoSPIMode)
      {
      // 2-wire isoSPI used.
      // Decide if Wake-Up Byte is necessary.
      if (isoSPI_IDLE)
         {
         // isoSPI State is IDLE (ISOSPI==IDLE).
         // WakeUp Byte is necessary.
         sendWakeUp = 1;

         // Set the correct isoSPI Wake-Up Time (uS) based on Core State.
         if (core_SLEEP)
            {
            // Core State is SLEEP (CORE==SLEEP).
            // VREG takes 300uS (tWAKE) to settle and power up the isoSPI Port to be
            // in READY State.
            isoWakeUpTime = 300;
            }
         else if (! core_SLEEP)
            {
            // Core State is STANDBY or REFUP (CORE==STANDBY/REFUP).
            // VREG is already on or powered-up but isoSPI Port needs 10uS (tREADY)
            // to be in READY State.
            isoWakeUpTime = 10;
            }
```

```
      // Set devicesConnected multiplier.
      if (daisyChain)
         {
         // Daisy Chain Bus must Wake-Up each isoSPI Port one-by-one.
         devicesConnected = totalDevices;
         }
      else if (! daisyChain)
         {
         // Parallel-Addressable isoSPI Bus must Wake-Up only one isoSPI port.
         devicesConnected = 1;
         }
      // Calculate total isoSPI Port Wake-Up Time.
      totalisoWakeUpTime = (isoWakeUpTime * devicesConnected);
      }

   else if (! isoSPI_IDLE)
      {
      // isoSPI Operational State is READY (ISOSPI==READY).
      // Wake-Up Byte is unnecessary.
      sendWakeUp = 0;
      }
   }
else if (! isoSPIMode)
   {
   // 4-wire SPI used.
   // Wake-Up Byte is unnecessary.
   sendWakeUp = 0;
   }

// Finally, if necessary send the Wake-Up Byte!
if (sendWakeUp)
   {
   // Send a Wake-Up byte.

   // Pull CSB low.
   output_low(CSB);

   // Wake-Up Byte can be any value.
   spi_write(0x00);

   // Pull CSB high.
   output_high(CSB);

   // Give enough Wake-Up time for LTC6804 Cores to accept a valid command.
   delay_us(totalisoWakeUpTime);
   }

} // end of void isoSPI_WakeUp_Write()
```

5) SPI command usage with scope photos:

5a) Address Read Configuration Registers (RDCFG) to confirm everything is connected.

The Address RDCFG command is the first command used to confirm all the device connections between the microcontroller and SPI communication code are done correctly. The Address RDCFG command reads the configuration of a single device only. Multiple Address RDCFG commands are necessary to read multiple addressable LTC6804-2 devices.

The Address RDCFG command is also commonly used after a Broadcast or Address Write Configuration Registers (WRCFG) command to confirm changes in the Configuration Registers (CFGR).

Total LTC6804-2 Address RDCFG communication bytes:
Single Device = 4 write bytes (2 x command + 2 x PEC) + 8 read bytes (6 x CFGR + 2 x PEC)
Multiple Devices = Number of Devices * (4 write bytes + 8 read bytes)

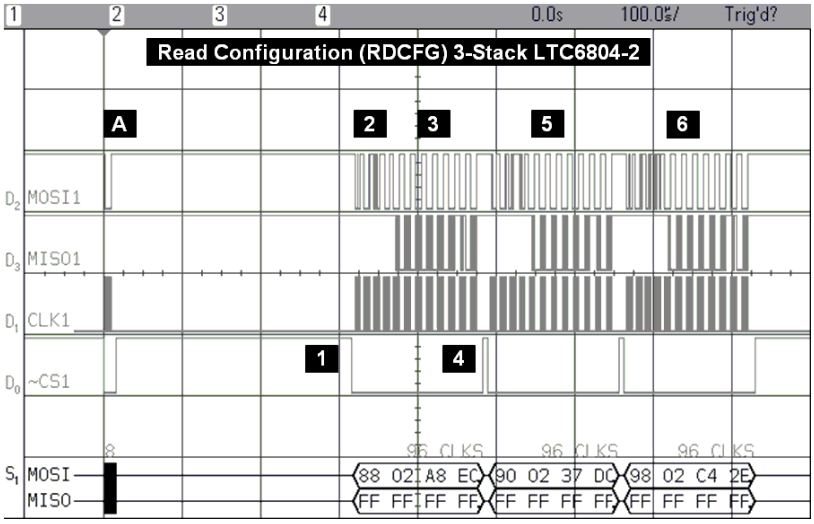
Read Configuration Registers

(Parallel-Addressable configuration, 3 stacked devices)

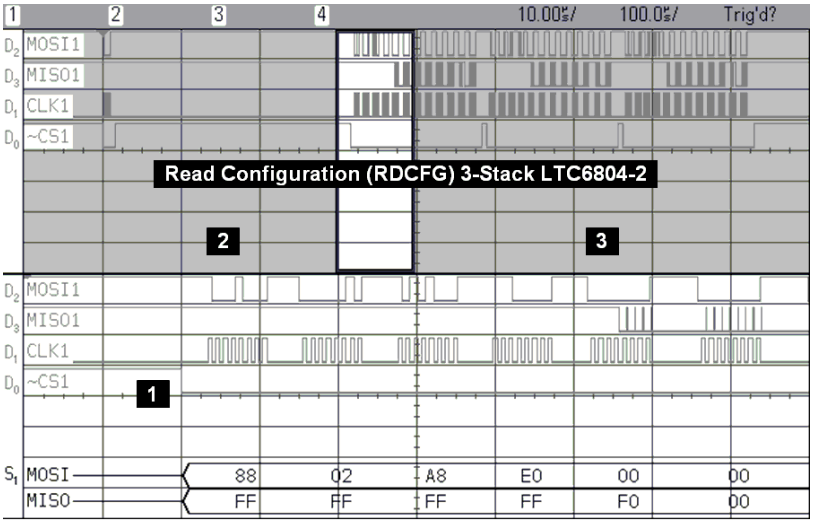
```
// isoSPI_WakeUp_Write() FUNCTION VARIABLE DECLARATION:
    int1_t isoSPIMode  = 1;             // 2-wire isoSPI used.
    int1_t isoSPI_IDLE = 1;             // isoSPI in IDLE State (ISOSPI==IDLE).
    int1_t core_SLEEP  = 1;             // LTC6804 Core in SLEEP State (CORE==SLEEP).
    int1_t daisyChain  = 0;             // Addressable Bus Communication.

// RDCFG VARIABLE DECLARATION:
//  3 Total Devices = (Bottom (B) Index=0) + (Middle (M) Index=1) + (Top (T) Index=2).
    const uint8_t TOTALDEVICES      = 3;
//  4 Total Command Bytes  = 2 bytes (CMD0 to CMD1) + 2 bytes (PECcmd0 to PECcmd1).
    const uint8_t TOTALCOMMANDBYTES = 4;
//  8 Total Data Bytes  = 6 bytes (CFGR0 to CFGR5) + 2 bytes (PECcfgr0 to PECcfgr1).
    const uint8_t TOTALDATABYTES    = 8;
// "for loop" variables.
    int8_t deviceIndex = 0;
    uint8_t dataIndex  = 0;

// LTC6804-2 A0 to A3 Pin Addresses: Bottom = 0x1, Middle = 0x2, Top = 0x3.
// Each addressable device has an individual command.
// Create uint8_t command arrays for spi_write().
    uint8_t ReadConfig_CommandArray[TOTALDEVICES][TOTALCOMMANDBYTES] =
        {
            { 0x88, 0x02, 0xA8, 0xE0 },     // BOTTOM DEVICE.
            { 0x90, 0x02, 0x37, 0xD0 },     // MIDDLE DEVICE.
            { 0x98, 0x02, 0xC4, 0x2E }      // TOP DEVICE.
        };

// Create uint8_t data read arrays for spi_read().
    uint8_t ReadConfig_DataArray[TOTALDEVICES][TOTALDATABYTES] =
        {
            { 0, 0, 0, 0, 0, 0, 0, 0 },     // BOTTOM DEVICE.
            { 0, 0, 0, 0, 0, 0, 0, 0 },     // MIDDLE DEVICE.
            { 0, 0, 0, 0, 0, 0, 0, 0 }      // TOP DEVICE.
        };
```

```
// A) Function will create a Wake-Up Byte based on LTC6804 Core State vs isoSPI State.
// void = isoSPI_WakeUp_Write(int1_t isoSPIMode, int1_t isoSPI_IDLE, int1_t core_SLEEP,
//                            int1_t daisyChain, uint8_t totalDevices)
    isoSPI_WakeUp_Write(isoSPIMode, isoSPI_IDLE, core_SLEEP, daisyChain, TOTALDEVICES);


// START OF ADDRESS RDCFG SPI WRITE THEN SPI READ:

// ON AN ADDRESSABLE STACK, EACH DEVICE IS SENT A COMMAND THEN DATA IS READ INDIVIDUALLY.
// RDCFG device order: 1) Bottom 2) Middle 3) Top.
    for (deviceIndex = 0; deviceIndex < TOTALDEVICES; deviceIndex++)
        {
        // | deviceIndex | devicePosition | Step |
        // |      0      | Bottom (B)     |  1-4 |
        // |      1      | Middle (M)     |  5   |
        // |      2      | Top    (T)     |  6   |

        // 1) Pull CSB low.
        output_low(CSB);

        // 2) Send Address RDCFG command and PEC Bytes to current LTC6804-2.
        spi_write(ReadConfig_CommandArray[deviceIndex][0]);
        spi_write(ReadConfig_CommandArray[deviceIndex][1]);
        spi_write(ReadConfig_CommandArray[deviceIndex][2]);
        spi_write(ReadConfig_CommandArray[deviceIndex][3]);

        // 3) Read the 8 CFGR bytes from current LTC6804-2.
        for (dataIndex = 0; dataIndex < TOTALDATABYTES; dataIndex++)
            {
            // | dataIndex | byteReadBack | dataIndex | byteReadBack |
            // |      0    | CFGR0        |      4    | CFGR4        |
            // |      1    | CFGR1        |      5    | CFGR5        |
            // |      2    | CFGR2        |      6    | PECCcfgr0    |
            // |      3    | CFGR3        |      7    | PECCcfgr1    |
            ReadConfig_DataArray[deviceIndex][dataIndex] = spi_read(0);
            }

        // 4) Pull CSB high.
        output_high(CSB);

        // 5) Repeat steps 1) to 4) to Address RDCFG Middle LTC6804-2.
        // 6) Repeat steps 1) to 4) to Address RDCFG Top LTC6804-2.
        }

// END OF ADDRESS RDCFG SPI WRITE THEN SPI READ:
```

Example Address Read Configuration (RDCFG) Pictures of 3 Stacked LTC6804-2 Devices with Wake-Up Byte:
A) Wake-Up Byte. First Address RDCFG device (B): 1) CSB low. 2) Send Address RDCFG + PEC. 3) Read CFGR. 4) CSB high.
5) Second Address RDCFG device (M) steps 1) - 4). 6) Third Address RDCFG device (T) steps 1) - 4).



Picture 1: Overall View.



Picture 2: Bottom Device Command + PEC Bytes Zoom.

Note 1: DC1942B LTC6804-2 GPIO1 is normally low and CFGR0 reads 0xF0 instead of 0xF8.

Note 2: When using a LTC6820 isoSPI Interface device as an isoSPI Master, SPI Reads will have narrow high pulses on MISO/SDO when SCK cycles low and MISO/SDO data is low. These 50nS narrow pulses are the isoSPI receive pulses or isoSPI Data Half-Pulse Widths (t1/2PW(D)). Direct 4-wire SPI Interface to a LTC6804 without a LTC6820, will not have narrow MISO/SDO pulses during SPI Reads.

5b) Address Write Configuration (WRCFG) to individually set each device flag thresholds and reduce ADC measurement time.

The Address WRCFG command sets the configuration of a single device only. Multiple Address WRCFG commands are necessary to set different configurations to multiple addressable LTC6804-2 devices.

After a Broadcast or Address WRCFG, use the Address Read Configuration (RDCFG) command to confirm changes in the Configuration Registers (CFGR).

Total LTC6804-2 Address WRCFG communication bytes:
Single Device = 4 write bytes (2 x command + 2 x PEC) + 8 data write bytes (6 x CFGR + 2 x PEC)
Multiple Devices = Number of Devices * (4 write bytes + 8 data write bytes)

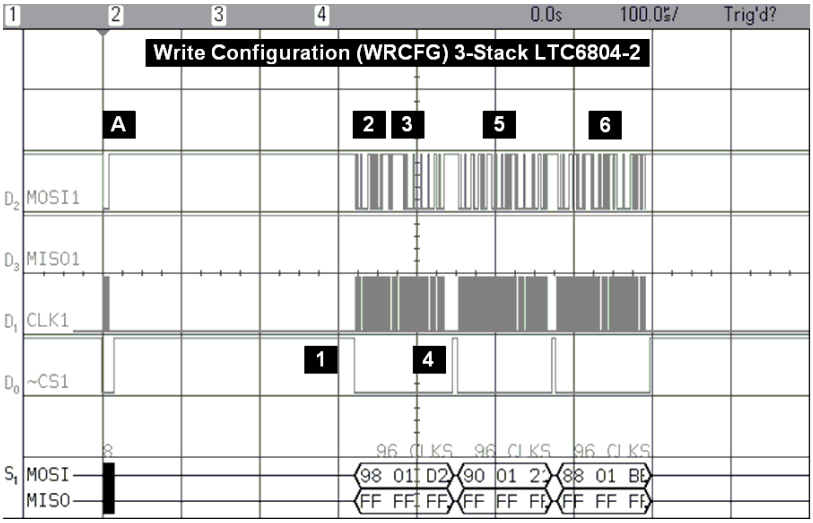
Write Configuration Registers

(Parallel-Addressable configuration, 3 stacked devices)

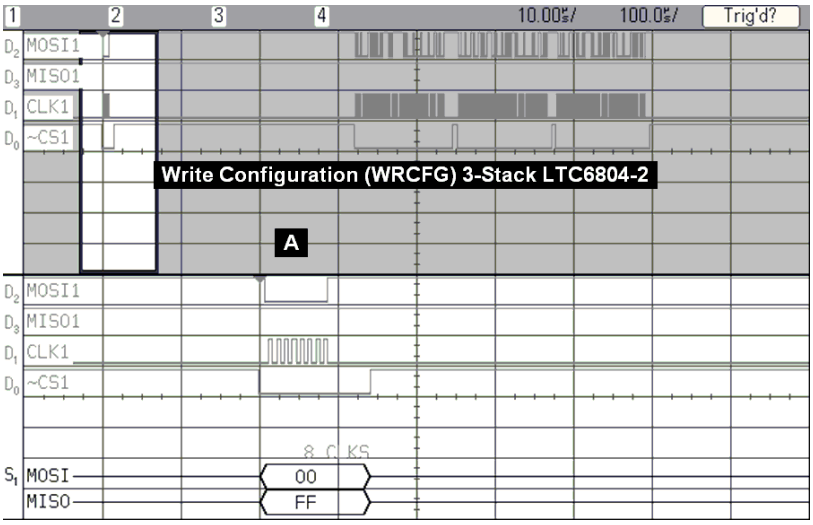
```
// isoSPI_WakeUp_Write() FUNCTION VARIABLE DECLARATION:
    int1_t isoSPIMode  = 1;             // 2-wire isoSPI used.
    int1_t isoSPI_IDLE = 1;             // isoSPI in IDLE State (ISOSPI==IDLE).
    int1_t core_SLEEP  = 1;             // LTC6804 Core in SLEEP State (CORE==SLEEP).
    int1_t daisyChain  = 0;             // Addressable Bus Communication.

// WRCFG VARIABLE DECLARATION:
//  3 Total Devices = (Bottom (B) Index=0) + (Middle (M) Index=1) + (Top (T) Index=2).
    const uint8_t TOTALDEVICES      = 3;
//  4 Total Command Bytes  = 2 bytes (CMD0 to CMD1) + 2 bytes (PECcmd0 to PECcmd1).
    const uint8_t TOTALCOMMANDBYTES = 4;
//  8 Total Data Bytes  = 6 bytes (CFGR0 to CFGR5) + 2 bytes (PECcfgr0 to PECcfgr1).
    const uint8_t TOTALDATABYTES    = 8;
// "for loop" variables.
    int8_t deviceIndex = 0;
    uint8_t dataIndex  = 0;

// LTC6804-2 A0 to A3 Pin Addresses: Bottom = 0x1, Middle = 0x2, Top = 0x3.
// Each addressable device has an individual command.
// Create uint8_t command arrays for spi_write().
    uint8_t WriteConfig_CommandArray[TOTALDEVICES][TOTALCOMMANDBYTES] =
        {
            { 0x88, 0x01, 0xBE, 0x84 },     // BOTTOM DEVICE.
            { 0x90, 0x01, 0x21, 0xB4 },     // MIDDLE DEVICE.
            { 0x98, 0x01, 0xD2, 0x4A }      // TOP DEVICE.
        };

// Configure all 3 devices on the Parallel-Addressable LTC6804-2 stack to have:
// 1. GPIO1 to GPIO5    = 1 (input mode).
// 2. REFON             = 1 (VREF1 and VREF2 normally on)
//                        With REFON==1, there is no Reference Wake-Up Time (tREFUP).
// NOTE: Flags are given arbitrary values to help differentiate between the devices.
//                       | DEVICE#1 | DEVICE#2 | DEVICE#3 |
// 3. UnderVoltageFlag  = |  3.1008V |  3.2000V |  2.3008V |
// 4. OverVoltageFlag   = |  4.1008V |  4.2000V |  4.3008V |
//
// | Register | BIT 7  | BIT 6  | BIT 5  | BIT 4  | BIT 3  | BIT 2  | BIT 1  | BIT 0  |
// | CFGR0    | GPIO5  | GPIO4  | GPIO3  | GPIO2  | GPIO1  | REFON  | SWTEN  | ADCOPT |
// | CFGR1    | VUV[07]| VUV[06]| VUV[05]| VUV[04]| VUV[03]| VUV[02]| VUV[01]| VUV[00]|
// | CFGR2    | VOV[03]| VOV[02]| VOV[01]| VOV[00]| VUV[11]| VUV[10]| VUV[09]| VUV[08]|
// | CFGR3    | VOV[11]| VOV[10]| VOV[09]| VOV[08]| VOV[07]| VOV[06]| VOV[05]| VOV[04]|
```

```
// Create uint8_t data write arrays for spi_write().
    uint8_t WriteConfig_DataArray[TOTALDEVICES][TOTALDATABYTES] =
        {
            { 0xFC, 0x02, 0x2A, 0x79, 0x00, 0x00, 0xD5, 0xAE },    // BOTTOM DEVICE.
            { 0xFC, 0x40, 0x0A, 0x7D, 0x00, 0x00, 0x41, 0x14 },    // MIDDLE DEVICE.
            { 0xFC, 0x7F, 0xFA, 0x80, 0x00, 0x00, 0x03, 0x10 }     // TOP DEVICE.
        };

// A) Function will create a Wake-Up Byte based on LTC6804 Core State vs isoSPI State.
// void = isoSPI_WakeUp_Write(int1_t isoSPIMode, int1_t isoSPI_IDLE, int1_t core_SLEEP,
//                            int1_t daisyChain, uint8_t totalDevices)
    isoSPI_WakeUp_Write(isoSPIMode, isoSPI_IDLE, core_SLEEP, daisyChain, TOTALDEVICES);

// START OF ADDRESS WRCFG SPI WRITE:

// ON AN ADDRESSABLE STACK, EACH DEVICE IS SENT A COMMAND AND DATA INDIVIDUALLY.
// WRCFG device order: 1) Top 2) Middle 3) Bottom.
    for (deviceIndex = 2; deviceIndex > -1; deviceIndex--)
        {
        // | deviceIndex | devicePosition | Step |
        // |      2      | Top    (T)     |  1-4 |
        // |      1      | Middle (M)     |  5   |
        // |      0      | Bottom (B)     |  6   |

        // 1) Pull CSB low.
        output_low(CSB);

        // 2) Send Address WRCFG command and PEC Bytes to current LTC6804-2.
        spi_write(WriteConfig_CommandArray[deviceIndex][0]);
        spi_write(WriteConfig_CommandArray[deviceIndex][1]);
        spi_write(WriteConfig_CommandArray[deviceIndex][2]);
        spi_write(WriteConfig_CommandArray[deviceIndex][3]);

        // 3) Write the 8 CFGR bytes to current LTC6804-2.
        for (dataIndex = 0; dataIndex < TOTALDATABYTES; dataIndex++)
            {
            // | dataIndex | byteWrite    | dataIndex | byteWrite    |
            // |      0    | CFGR0        |      4    | CFGR4        |
            // |      1    | CFGR1        |      5    | CFGR5        |
            // |      2    | CFGR2        |      6    | PECCcfgr0    |
            // |      3    | CFGR3        |      7    | PECCcfgr1    |
            spi_write(WriteConfig_DataArray[deviceIndex][dataIndex]);
            }

        // 4) Pull CSB high.
        output_high(CSB);

        // 5) Repeat steps 1) to 4) to Address WRCFG Middle LTC6804-2.
        // 6) Repeat steps 1) to 4) to Address WRCFG Bottom LTC6804-2.
        }

// END OF ADDRESS WRCFG SPI WRITE:
```

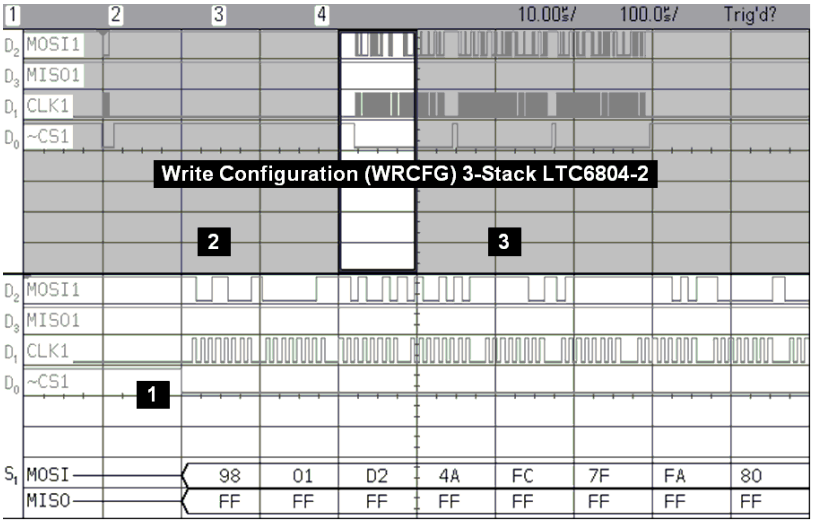
Example Address Write Configuration (WRCFG) Pictures of 3 Stacked LTC6804-2 Devices with Wake-Up Byte.
A) Wake-Up Byte. First Address WRCFG device (T): 1) CSB low. 2) Send Address WRCFG + PEC. 3) Write CFGR. 4) CSB high.
5) Second Address WRCFG device (M): steps 1) - 4). 6) Third Address WRCFG device (B): Steps 1) - 4).



Picture 3: Overall View.



Picture 4: Wake-Up Byte Zoom.



Picture 5: Top Device Command + PEC Bytes Zoom.

5c) Broadcast Write Configuration (WRCFG) simultaneously set all devices to identical flag thresholds and reduce ADC measurement time.

The LTC6804-2 Broadcast WRCFG command sets all devices to identical configurations. If cells are discharged by setting CFGR0.DCC[x] bits to 1, all devices on Parallel-Addressable isoSPI Bus will discharge identical cells. Example: Broadcast WRCFG DCC12 = 1, all LTC6804-2 devices on bus will discharge Cell12.

After a Broadcast or Address WRCFG, use the Address Read Configuration (RDCFG) command to confirm changes in the Configuration Registers (CFGR).

Total LTC6804-2 Broadcast WRCFG communication bytes:
Single Device = 4 write bytes (2 x command + 2 x PEC) + 8 data write bytes (6 x CFGR + 2 x PEC)
Multiple Devices = 4 write bytes + 8 data write bytes

Write Configuration Registers
(Parallel-Addressable configuration, 3 stacked devices)

```
// isoSPI_WakeUp_Write() FUNCTION VARIABLE DECLARATION:
    int1_t isoSPIMode  = 1;             // 2-wire isoSPI used.
    int1_t isoSPI_IDLE = 1;             // isoSPI in IDLE State (ISOSPI==IDLE).
    int1_t core_SLEEP  = 1;             // LTC6804 Core in SLEEP State (CORE==SLEEP).
    int1_t daisyChain  = 0;             // Addressable Bus Communication.

// WRCFG VARIABLE DECLARATION:
//  3 Total Devices = (Bottom (B) Index=0) + (Middle (M) Index=1) + (Top (T) Index=2).
    const uint8_t TOTALDEVICES      = 3;
//  8 Total Data Bytes  = 6 bytes (CFGR0 to CFGR5) + 2 bytes (PECcfgr0 to PECcfgr1).
    const uint8_t TOTALDATABYTES    = 8;
// "for loop" variables.
    uint8_t dataIndex  = 0;
// Configure all 3 devices on the Parallel-Addressable LTC6804-2 stack to have:
// 1. GPIO1 to GPIO5    = 1 (input mode).
// 2. REFON             = 1 (VREF1 and VREF2 normally on)
//                        With REFON==1, there is no Reference Wake-Up Time (tREFUP).
// 3. UnderVoltageFlag  = 2.7008V
// 4. OverVoltageFlag   = 4.200V
// Create uint8_t data write arrays for spi_write().
    uint8_t WriteConfig_DataArray[TOTALDATABYTES] =
        { 0xFC, 0x97, 0x16, 0XA4, 0x00, 0x00, 0xCD, 0x9E }; //ALL DEVICES.
```

```
// A) Function will create a Wake-Up Byte based on LTC6804 Core State vs isoSPI State.
// void = isoSPI_WakeUp_Write(int1_t isoSPIMode, int1_t isoSPI_IDLE, int1_t core_SLEEP,
//                            int1_t daisyChain, uint8_t totalDevices)
    isoSPI_WakeUp_Write(isoSPIMode, isoSPI_IDLE, core_SLEEP, daisyChain, TOTALDEVICES);


// START OF BROADCAST WRCFG SPI WRITE:

// 1) Pull CSB low.
    output_low(CSB);

// 2) Send Broadcast WRCFG command (0x00 0x01) and PEC Bytes (0x3D 0x6E).
    spi_write(0x00);
    spi_write(0x01);
    spi_write(0x3D);
    spi_write(0x6E);

// 3) Write the 8 CFGR bytes to all LTC6804-2 devices.
    for (dataIndex = 0; dataIndex < TOTALDATABYTES; dataIndex++)
        {
        // | dataIndex | byteWrite    | dataIndex | byteWrite    |
        // |      0    | CFGR0        |      4    | CFGR4        |
        // |      1    | CFGR1        |      5    | CFGR5        |
        // |      2    | CFGR2        |      6    | PECCcfgr0    |
        // |      3    | CFGR3        |      7    | PECCcfgr1    |
        spi_write(WriteConfig_DataArray[dataIndex]);
        }

// 4) Pull CSB high.
    output_high(CSB);

// END OF BROADCAST WRCFG SPI WRITE:
```

5d) Address Start Cell Voltage ADC Conversion and Poll Status (ADCV) to have one device measure cells.

The Address ADCV command is very useful to concentrate on a single device monitoring a specific group of cells without having all devices on the bus unnecessarily measuring cells.

Polling immediately after the Address ADCV command is an normal way to determine when an addressable device ADC conversion has completed. However, an issue with this polling method is the microcontroller is not free to do other serial communication while waiting for ADC conversions to complete.

After ADC Conversions have completed, use the 4 Address Read Cell Voltage Register commands (RDCVA (1-3), RDCVB (4-6), RDCVC (7-9), and RDCVD (10-12)) to read the cell voltage results.

Total LTC6804-2 Address ADCV communication bytes:
Single Device with polling = 4 write bytes (2 x command + 2 x PEC) + n* polling read bytes
Single Device without polling = 4 write bytes

n*= The number of polling read bytes is dependent on the Measurement + Calibration Cycle Time (tCYCLE).

Start Cell Voltage ADC Conversion

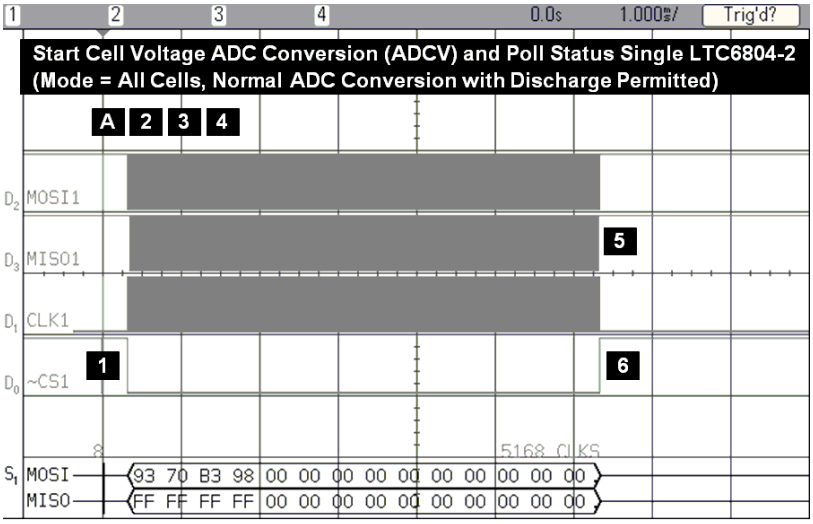
(Parallel-Addressable configuration, 3 stacked devices)
(All cells, normal mode with discharge permitted) and poll status.

```
// isoSPI_WakeUp_Write() FUNCTION VARIABLE DECLARATION:
    int1_t isoSPIMode  = 1;             // 2-wire isoSPI used.
    int1_t isoSPI_IDLE = 1;             // isoSPI in IDLE State (ISOSPI==IDLE).
    int1_t core_SLEEP  = 1;             // LTC6804 Core in SLEEP State (CORE==SLEEP).
    int1_t daisyChain  = 0;             // Addressable Bus Communication.

// ADCV VARIABLE DECLARATION:
//  3 Total Devices = (Bottom (B) Index=0) + (Middle (M) Index=1) + (Top (T) Index=2).
    const uint8_t TOTALDEVICES      = 3;
//  4 Total Command Bytes  = 2 bytes (CMD0 to CMD1) + 2 bytes (PECcmd0 to PECcmd1).
    const uint8_t TOTALCOMMANDBYTES = 4;
// Stores results while reading MISO/SDO to confirm ADC Conversions have completed.
    uint8_t sDORead = 0x00;

// Send ADCV command to the middle device only.
// | deviceIndex | devicePosition |
// |      0      | Bottom (B)     |
// |      1      | Middle (M)     |
// |      2      | Top    (T)     |
    int8_t deviceIndex = 1;

// LTC6804-2 A0 to A3 Pin Addresses: Bottom = 0x1, Middle = 0x2, Top = 0x3.
// Each addressable device has an individual command.
//    1. ADC Mode            = Normal    (MD[1:0] =  10).
//    2. Discharge Permitted = True      (DCP     =   1).
//    3. Cell Selection      = All Cells (CH[2:0] = 000).
// Create uint8_t command arrays for spi_write().
    uint8_t StartCellConversion_CommandArray[TOTALDEVICES][TOTALCOMMANDBYTES] =
    {
        { 0x8B, 0x70, 0x2C, 0xA8 },     // BOTTOM DEVICE.
        { 0x93, 0x70, 0xB3, 0x98 },     // MIDDLE DEVICE.
        { 0x9B, 0x70, 0x40, 0x66 }      // TOP DEVICE.
    };
```

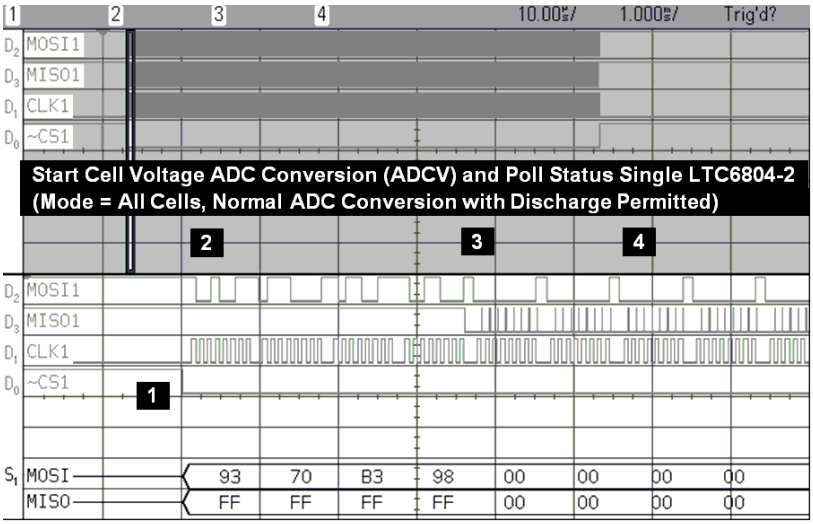
```
// A) Function will create a Wake-Up Byte based on LTC6804 Core State vs isoSPI State.
// void = isoSPI_WakeUp_Write(int1_t isoSPIMode, int1_t isoSPI_IDLE, int1_t core_SLEEP,
//                            int1_t daisyChain, uint8_t totalDevices)
    isoSPI_WakeUp_Write(isoSPIMode, isoSPI_IDLE, core_SLEEP, daisyChain, TOTALDEVICES);


// START OF ADDRESS ADCV AND POLL SPI WRITE THEN SPI READ:

// 1) Pull CSB low.
    output_low(CSB);

// 2) Send Address ADCV command and PEC Bytes to current LTC6804-2.
    spi_write(StartCellConversion_CommandArray[deviceIndex][0]);
    spi_write(StartCellConversion_CommandArray[deviceIndex][1]);
    spi_write(StartCellConversion_CommandArray[deviceIndex][2]);
    spi_write(StartCellConversion_CommandArray[deviceIndex][3]);

// 3) MISO/SDO output of the current device is pulled low for the duration of the
//    conversion (~ XX ms).

// 4) Continue to send clock pulses on SCK to simultaneously prevent isoSPI
//    Idle Timeout (tIDLE) to IDLE state and monitor/read the updated MISO/SDO
//    output after every pulse.
    while (sDORead != 0xFF)
        {
        // 5) MISO/SDO output goes high indicating conversions are complete for
        // the current device.

        // Keep on reading MISO/SDO until sDORead==0xFF; signifying the ADC
        // Conversion has completed!
        sDORead = spi_read(0);
        }

// 6) Pull CSB high to exit polling.
    output_high(CSB);

// END OF ADDRESS ADCV AND POLL SPI WRITE THEN SPI READ:
```

Example Address Start Cell Voltage ADC and Poll Status (ADCV) Pictures of a Single LTC6804-2 Device with Wake-Up Byte.

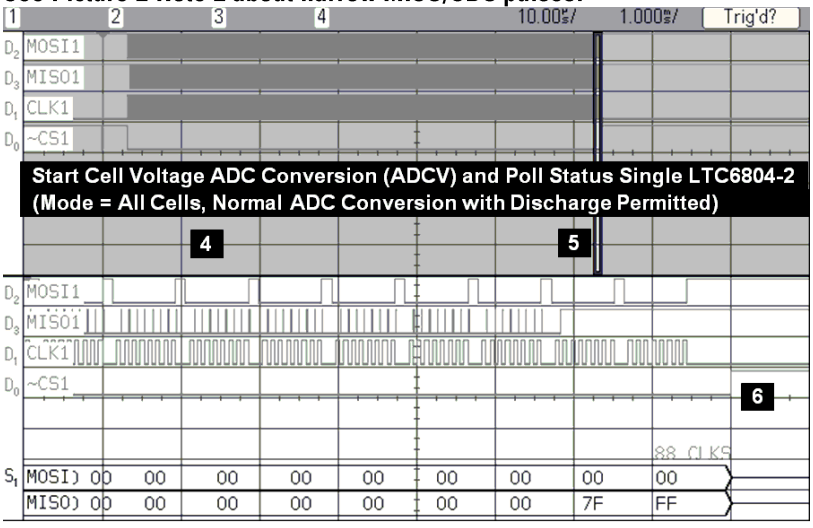
A) Wake-Up Byte. Address ADCV device (M): 1) CSB low. 2) Send Address ADCV + PEC. 3) ADC Conversion started; MISO/SDO output is low. 4) Poll ADC by reading MISO/SDO output. 5) ADC Conversion completed; MISO/SDO output is high. 6) CSB high.



Picture 6: Overall View.



Picture 7: Command + PEC Bytes and Device Busy with Conversion Zoom. See Picture 2 Note 2 about narrow MISO/SDO pulses.



Picture 8: Device Finished with Conversion Zoom. See Picture 2 Note 2 about narrow MISO/SDO pulses.

5e) Broadcast Start Cell Voltage ADC Conversion (ADCV) to have all devices simultaneously measure cells.

The Broadcast ADCV command saves time by having all devices on the bus simultaneously measure cells; very useful for getting all cell voltages at a point in time.

Use the Address Poll ADC (PLADC) command to confirm when the ADC Conversions have completed for one of the devices.

After ADC Conversions have completed, use the 4 Address Read Cell Voltage Register commands (RDCVA (1-3), RDCVB (4-6), RDCVC (7-9), and RDCVD (10-12)) to read the cell voltage results.

Total LTC6804-2 Broadcast ADCV communication bytes:
Single Device without polling = 4 write bytes (2 x command + 2 x PEC)
Multiple Devices without polling = 4 write bytes

Start Cell Voltage ADC Conversion

(Parallel-Addressable configuration, 3 stacked devices)
(All cells, normal mode with discharge permitted)

```
// isoSPI_WakeUp_Write() FUNCTION VARIABLE DECLARATION:
    int1_t isoSPIMode  = 1;             // 2-wire isoSPI used.
    int1_t isoSPI_IDLE = 1;             // isoSPI in IDLE State (ISOSPI==IDLE).
    int1_t core_SLEEP  = 1;             // LTC6804 Core in SLEEP State (CORE==SLEEP).
    int1_t daisyChain  = 0;             // Addressable Bus Communication.

// ADCV VARIABLE DECLARATION:
//  3 Total Devices = (Bottom (B) Index=0) + (Middle (M) Index=1) + (Top (T) Index=2).
   const int8 TOTALDEVICES    = 3;

// A) Function will create a Wake-Up Byte based on LTC6804 Core State vs isoSPI State.
// void = isoSPI_WakeUp_Write(int1 isoSPIMode, int1 isoSPI_IDLE, int1 core_SLEEP,
//                            int1 daisyChain, int8 totalDevices)
   isoSPI_WakeUp_Write(isoSPIMode, isoSPI_IDLE, core_SLEEP, daisyChain, TOTALDEVICES);

// START OF BROADCAST ADCV SPI WRITE:

// 1) Pull CSB low.
    output_low(CSB);

// 2) Send ADCV command (0x03 0x70) and PEC Bytes (0xAF 0x42).
//    1) ADC Mode            = Normal    (MD[1:0] =  10).
//    2) Discharge Permitted = True      (DCP     =   1).
//    3) Cell Selection      = All Cells (CH[2:0] = 000).
    spi_write(0x03);
    spi_write(0x70);
    spi_write(0xAF);
    spi_write(0x42);

// 3) Pull CSB high.
    output_high(CSB);

// END OF BROADCAST ADCV SPI WRITE:
```

5f) Address Poll ADC Status (PLADC) to perform other tasks during an ADC Conversion of one device then return to confirm ADC Conversions have completed.

The Address PLADC command is another normal way to determine when an addressable device ADC conversion has completed but allows the microcontroller to perform other tasks after Start ADC Command such as ADCV.

Example of using the Address PLADC command instead of polling immediately after an ADCV command:
1) A Broadcast or Address ADCV command is sent without polling. 2) Other tasks are performed.
3) An Address PLADC command polls an addressable device ADC until the conversions have completed.

After ADC Conversions have completed, use the 4 Address Read Cell Voltage Register commands (RDCVA (1-3), RDCVB (4-6), RDCVC (7-9), and RDCVD (10-12)) to read the cell voltage results.

Total LTC6804-2 Address PLADC communication bytes:
Single Device with polling = 4 write bytes (2 x command + 2 x PEC) + n* polling read bytes
Single Device without polling = 4 write bytes

n*= The number of polling read bytes is dependent on the Measurement + Calibration Cycle Time (tCYCLE).

Poll ADC status

(Parallel-Addressable configuration, 3 stacked devices)

```
// isoSPI_WakeUp_Write() FUNCTION VARIABLE DECLARATION:
    int1_t isoSPIMode  = 1;             // 2-wire isoSPI used.
    int1_t isoSPI_IDLE = 1;             // isoSPI in IDLE State (ISOSPI==IDLE).
    int1_t core_SLEEP  = 1;             // LTC6804 Core in SLEEP State (CORE==SLEEP).
    int1_t daisyChain  = 0;             // Addressable Bus Communication.

// PLADC VARIABLE DECLARATION:
//  3 Total Devices = (Bottom (B) Index=0) + (Middle (M) Index=1) + (Top (T) Index=2).
    const uint8_t TOTALDEVICES      = 3;
//  4 Total Command Bytes  = 2 bytes (CMD0 to CMD1) + 2 bytes (PECcmd0 to PECcmd1).
    const uint8_t TOTALCOMMANDBYTES = 4;
// Stores results while reading MISO/SDO to confirm ADC Conversions have completed.
    uint8_t sDORead = 0x00;

// Send Address PLADC command to the middle device only.
// | deviceIndex | devicePosition |
// |      0      | Bottom (B)     |
// |      1      | Middle (M)     |
// |      2      | Top    (T)     |
    int8_t deviceIndex = 1;

// LTC6804-2 A0 to A3 Pin Addresses: Bottom = 0x1, Middle = 0x2, Top = 0x3.
// Each addressable device has an individual command.
// Create uint8_t command arrays for spi_write().
    uint8_t PollADC_CommandArray[TOTALDEVICES][TOTALCOMMANDBYTES] =
        {
            { 0x8F, 0x14, 0x70, 0x86 },     // BOTTOM DEVICE.
            { 0x97, 0x14, 0xEF, 0xB6 },     // MIDDLE DEVICE.
            { 0x9F, 0x14, 0x1C, 0x48 }      // TOP DEVICE.
        };
```

```
// A) Function will create a Wake-Up Byte based on LTC6804 Core State vs isoSPI State.
// void = isoSPI_WakeUp_Write(int1 isoSPIMode, int1 isoSPI_IDLE, int1 core_SLEEP,
//                            int1 daisyChain, int8 totalDevices)
   isoSPI_WakeUp_Write(isoSPIMode, isoSPI_IDLE, core_SLEEP, daisyChain, TOTALDEVICES);

// START OF ADDRESS PLADC SPI WRITE THEN SPI READ:

// 1) Pull CSB low.
    output_low(CSB);

// 2) Send Address PLADC command and PEC Bytes to current LTC6804-2.
    spi_write(PollADC_CommandArray[deviceIndex][0]);
    spi_write(PollADC_CommandArray[deviceIndex][1]);
    spi_write(PollADC_CommandArray[deviceIndex][2]);
    spi_write(PollADC_CommandArray[deviceIndex][3]);

// 3) MISO/SDO output of the bottom device is pulled low for the duration of the
//    conversion (~ XX ms).

// 4) Continue to send clock pulses on SCK to simultaneously prevent isoSPI
//    Idle Timeout (tIDLE) to IDLE state and monitor/read the updated MISO/SDO
//    output after every pulse.
    while (sDORead != 0xFF)
        {
        // 5) MISO/SDO output goes high indicating conversions are complete for
        // the current device.

        // Keep on reading MISO/SDO until sDORead==0xFF; signifying the ADC
        // Conversion has completed!
        sDORead = spi_read(0);
        }

// 6) Pull CSB high to exit polling.
    output_high(CSB);

// END OF ADDRESS PLADC SPI WRITE THEN SPI READ:
```

5g) Address Read Cell Voltage Register Group A (RDCVA) to individually read Cells 1 to 3 results from each device.

The LTC6804 only allows readback of 3 cell voltage results at one time. To read all 12 cell voltage results require sending 4 Read Cell Voltage Register commands (RDCVA (1-3), RDCVB (4-6), RDCVC (7-9), and RDCVD (10-12)).

The Address RDCVA command reads the 3 cell voltage results of a single device only. Multiple Address RDCVA commands are necessary to read multiple addressable LTC6804-2 devices.

The Address RDCVA and other Address RDCVx commands are used after ADC Conversions have completed to read the cell voltage results with the following Broadcast or Address Start ADC commands:
1) Start Cell Voltage ADC Conversion (ADCV) 2) Start Open Wire ADC Conversion (ADOW)
3) Start Self-Test Cell Voltage Conversion (CVST) 4) Start Combined Cell Voltage and GPIO1, GPIO2 Conversion (ADCVAX) and 5) Clear Cell Voltage Register Group (CLRCELL).

Total LTC6804-2 Address RDCVA communication bytes:
Single Device = 4 write bytes (2 x command + 2 x PEC) + 8 read bytes (6 x CVAR + 2 x PEC)
Multiple Devices = Number of Devices * (4 write bytes + 8 read bytes)

Read Cell Voltage Register Group A

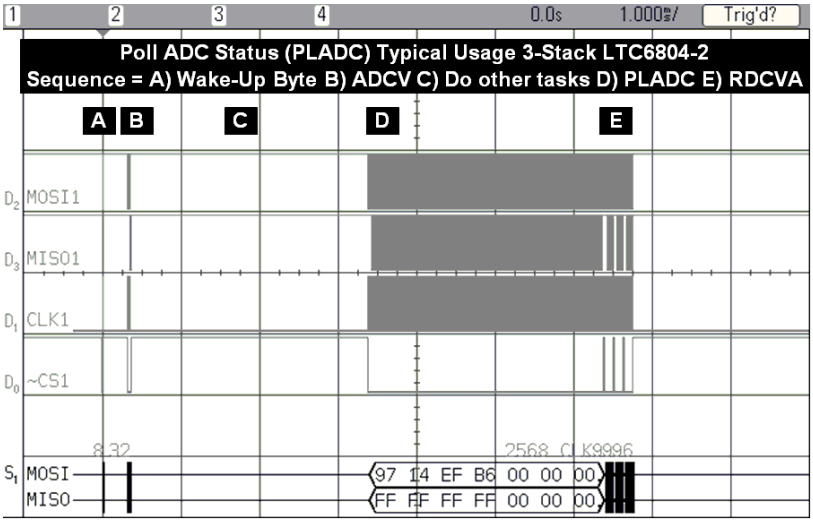
(Parallel-Addressable configuration, 3 stacked devices)

```
// isoSPI_WakeUp_Write() FUNCTION VARIABLE DECLARATION:
    int1_t isoSPIMode  = 1;             // 2-wire isoSPI used.
    int1_t isoSPI_IDLE = 1;             // isoSPI in IDLE State (ISOSPI==IDLE).
    int1_t core_SLEEP  = 1;             // LTC6804 Core in SLEEP State (CORE==SLEEP).
    int1_t daisyChain  = 0;             // Addressable Bus Communication.

// RDCVA VARIABLE DECLARATION:
//  3 Total Devices = (Bottom (B) Index=0) + (Middle (M) Index=1) + (Top (T) Index=2).
    const uint8_t TOTALDEVICES      = 3;
//  4 Total Command Bytes  = 2 bytes (CMD0 to CMD1) + 2 bytes (PECcmd0 to PECcmd1).
    const uint8_t TOTALCOMMANDBYTES = 4;
//  8 Total Data Bytes  = 6 bytes (CFGR0 to CFGR5) + 2 bytes (PECcvar0 to PECcvar1).
    const uint8_t TOTALDATABYTES    = 8;
// "for loop" variables.
    int8_t deviceIndex = 0;
    uint8_t dataIndex = 0;

// LTC6804-2 A0 to A3 Pin Addresses: Bottom = 0x1, Middle = 0x2, Top = 0x3.
// Each addressable device has an individual command.
// Create uint8_t command arrays for spi_write().
    uint8_t RDCVA_CommandArray[TOTALDEVICES][TOTALCOMMANDBYTES] =
    {
        { 0x88, 0x04, 0x84, 0x28 },     // BOTTOM DEVICE.
        { 0x90, 0x04, 0x1B, 0x18 },     // MIDDLE DEVICE.
        { 0x98, 0x04, 0xE8, 0xE6 }      // TOP DEVICE.
    };

// Create uint8_t data read arrays for spi_read().
    uint8_t RDCVA_DataArray[TOTALDEVICES][TOTALDATABYTES] =
    {
        { 0, 0, 0, 0, 0, 0, 0, 0 },     // BOTTOM DEVICE.
        { 0, 0, 0, 0, 0, 0, 0, 0 },     // MIDDLE DEVICE.
        { 0, 0, 0, 0, 0, 0, 0, 0 }      // TOP DEVICE.
    };
```

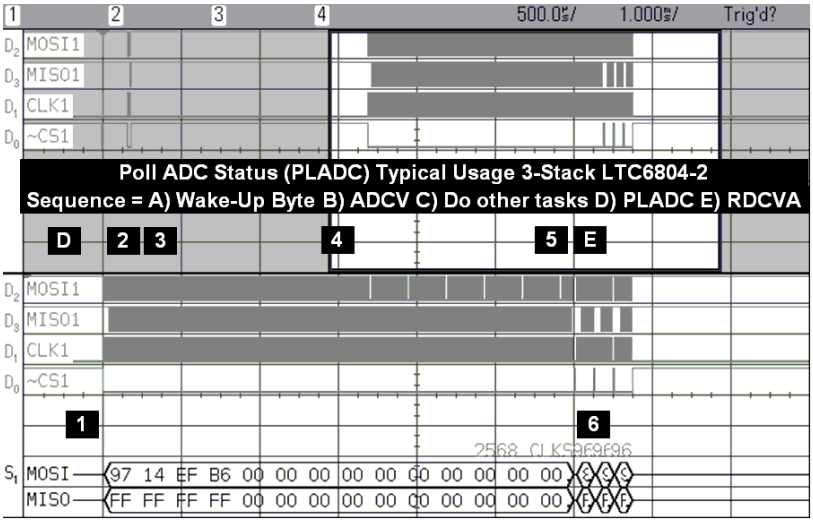
```
// A) Function will create a Wake-Up Byte based on LTC6804 Core State vs isoSPI State.
// void = isoSPI_WakeUp_Write(int1 isoSPIMode, int1 isoSPI_IDLE, int1 core_SLEEP,
//                            int1 daisyChain, int8 totalDevices)
    isoSPI_WakeUp_Write(isoSPIMode, isoSPI_IDLE, core_SLEEP, daisyChain, TOTALDEVICES);

// START OF ADDRESS RDCVA SPI WRITE THEN SPI READ:

// ON AN ADDRESSABLE STACK, EACH DEVICE IS SENT A COMMAND THEN DATA IS READ INDIVIDUALLY.
// RDCVA device order: 1) Bottom 2) Middle 3) Top.
    for (deviceIndex = 0; deviceIndex < TOTALDEVICES; deviceIndex++)
        {
        // | deviceIndex | devicePosition | Step |
        // |      0      | Bottom (B)     |  1-4 |
        // |      1      | Middle (M)     |  5   |
        // |      2      | Top    (T)     |  6   |

        // 1) Pull CSB low.
        output_low(CSB);

        // 2) Send Address RDCVA command and PEC Bytes to current LTC6804-2.
        spi_write(RDCVA_CommandArray[deviceIndex][0]);
        spi_write(RDCVA_CommandArray[deviceIndex][1]);
        spi_write(RDCVA_CommandArray[deviceIndex][2]);
        spi_write(RDCVA_CommandArray[deviceIndex][3]);

        // 3) Read the 8 CVAR bytes from current LTC6804-2.
        for (dataIndex = 0; dataIndex < TOTALDATABYTES; dataIndex++)
            {
            // | dataIndex | byteReadBack | dataIndex | byteReadBack |
            // |      0    | CVAR0        |      4    | CVAR4        |
            // |      1    | CVAR1        |      5    | CVAR5        |
            // |      2    | CVAR2        |      6    | PECcvar0     |
            // |      3    | CVAR3        |      7    | PECcvar1     |
            RDCVA_DataArray[deviceIndex][dataIndex] = spi_read(0);
            }

        // 4) Pull CSB high.
        output_high(CSB);

        // 5) Repeat steps 1) to 4) to Address RDCVA Middle LTC6804-2.
        // 6) Repeat steps 1) to 4) to Address RDCVA Top LTC6804-2.
        }

// END OF ADDRESS RDCVA SPI WRITE THEN SPI READ:
```

5h) Typical ADCV, PLADC, and RDCVA command usage together.

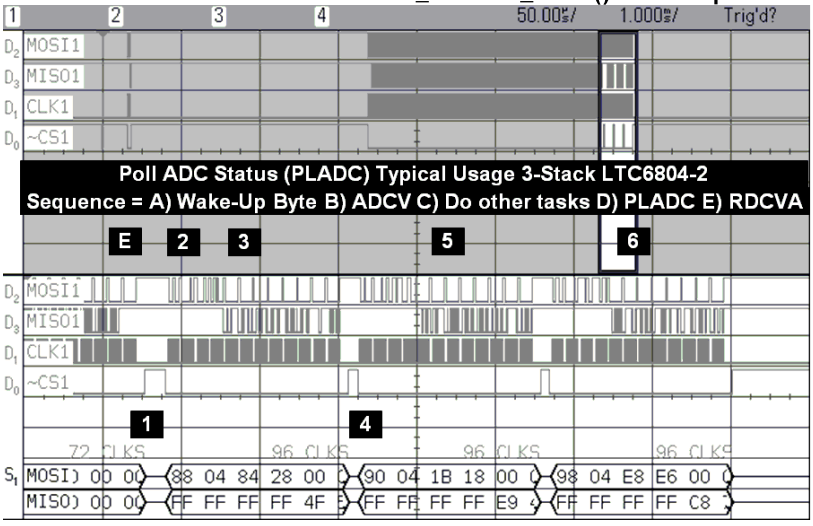
Example Command Sequence Pictures of 3 Stacked LTC6804-2 Devices:
A) Wake-Up Byte. B) Broadcast Start Cell Voltage ADC (ADCV). C) Perform Other Tasks for 3mS. D) Address Poll ADC (PLADC) until Conversion is done. E) Address Read Cell Voltage Group Register A (RDCVA).



Picture 9: Overall View.



Picture 10: Address PLADC and Address RDCVA Zoom. See Pictures 7 and 8 for detailed polling.
Note 3: The PLADC and RDCVA isoSPI_WakeUP_Write() function parameters isoSPI_IDLE=0 and core_SLEEP=0.



Picture 11: Address RDCVA Zoom. See Picture 2 Note 2 about narrow MISO/SDO pulses.

5i) Address Clear Cell Voltage Registers (CLRCELL) to clear one device Cell Voltage Registers to ones.

One type of register diagnostics is to clear previous results by setting all bytes to ones or 0xFF's. Clearing previous results can also be used for determining if any register bits are stuck high or low. The Address Clear Cell Voltage Registers (CLRCELL) command clears a single device Cell Voltage Register Groups A, B, C and D (CVAR, CVBR, CVCR, and CVDR).

After a Broadcast or Address CLRCELL command, use the 4 Address Read Cell Voltage Register commands (RDCVA (1-3), RDCVB (4-6), RDCVC (7-9), and RDCVD (10-12)) to read the cell voltage results.

Cell voltage results can be read immediately with the 4 Address RDCVx commands because the Broadcast or Address CLRCELL command does not have a Measurement + Calibration Cycle Time (tCYCLE).

Total LTC6804-2 Address CLRCELL communication bytes:
Single Device = 4 write bytes (2 x command + 2 x PEC)

Clear Cell Voltage Registers

(Parallel-Addressable configuration, 3 stacked devices)

```
// isoSPI_WakeUp_Write() FUNCTION VARIABLE DECLARATION:
    int1_t isoSPIMode  = 1;             // 2-wire isoSPI used.
    int1_t isoSPI_IDLE = 1;             // isoSPI in IDLE State (ISOSPI==IDLE).
    int1_t core_SLEEP  = 1;             // LTC6804 Core in SLEEP State (CORE==SLEEP).
    int1_t daisyChain  = 0;             // Addressable Bus Communication.

// CLRCELL VARIABLE DECLARATION:
//  3 Total Devices = (Bottom (B) Index=0) + (Middle (M) Index=1) + (Top (T) Index=2).
    const uint8_t TOTALDEVICES      = 3;
//  4 Total Command Bytes  = 2 bytes (CMD0 to CMD1) + 2 bytes (PECcmd0 to PECcmd1).
    const uint8_t TOTALCOMMANDBYTES = 4;

// Send Address CLRCELL command to the middle device only.
// | deviceIndex | devicePosition |
// |      0      | Bottom (B)     |
// |      1      | Middle (M)     |
// |      2      | Top    (T)     |
    int8_t deviceIndex = 1;

// LTC6804-2 A0 to A3 Pin Addresses: Bottom = 0x1, Middle = 0x2, Top = 0x3.
// Each addressable device has an individual command.
// Create uint8_t command arrays for spi_write().
    uint8_t ClearCell_CommandArray[TOTALDEVICES][TOTALCOMMANDBYTES] =
        {
            { 0x8F, 0x11, 0x4A, 0x2A },     // BOTTOM DEVICE.
            { 0x97, 0x11, 0xD5, 0x1A },     // MIDDLE DEVICE.
            { 0x9F, 0x11, 0x26, 0xE4 }      // TOP DEVICE.
        };
```

```
// A) Function will create a Wake-Up Byte based on LTC6804 Core State vs isoSPI State.
// void = isoSPI_WakeUp_Write(int1 isoSPIMode, int1 isoSPI_IDLE, int1 core_SLEEP,
//                            int1 daisyChain, int8 totalDevices)
    isoSPI_WakeUp_Write(isoSPIMode, isoSPI_IDLE, core_SLEEP, daisyChain, TOTALDEVICES);

// START OF ADDRESS CLRCELL SPI WRITE:
// 1) Pull CSB low.
    output_low(CSB);

// 2) Send CLRCELL command and PEC Bytes to current LTC6804-2.
    spi_write(ClearCell_CommandArray[deviceIndex][0]);
    spi_write(ClearCell_CommandArray[deviceIndex][1]);
    spi_write(ClearCell_CommandArray[deviceIndex][2]);
    spi_write(ClearCell_CommandArray[deviceIndex][3]);

// 3) Pull CSB high.
    output_high(CSB);

// END OF ADDRESS CLRCELL SPI WRITE:
```

5j) Broadcast Clear Cell Voltage Registers (CLRCELL) to simultaneously clear the Cell Voltage Registers of all devices to ones.

One type of register diagnostics is to clear previous results by setting all bytes to ones or 0xFF's. Clearing previous results can also be used for determining if any register bits are stuck high or low. The Broadcast Clear Cell Voltage Registers (CLRCELL) command saves time by having all devices on the bus simultaneously clear Cell Voltage Register Groups A, B, C and D (CVAR, CVBR, CVCR, and CVDR).

After a Broadcast or Address CLRCELL command, use the 4 Address Read Cell Voltage Register commands (RDCVA (1-3), RDCVB (4-6), RDCVC (7-9), and RDCVD (10-12)) to read the cell voltage results.

Cell voltage results can be read immediately with the 4 Address RDCVx commands because the Broadcast or Address CLRCELL command does not have a Measurement + Calibration Cycle Time (tCYCLE).

Total LTC6804-2 Broadcast CLRCELL communication bytes:
Single Device = 4 write bytes (2 x command + 2 x PEC)
Multiple Devices = 4 write bytes

Clear Cell Voltage Registers

(Parallel-Addressable configuration, 3 stacked devices)

```
// isoSPI_WakeUp_Write() FUNCTION VARIABLE DECLARATION:
    int1_t isoSPIMode  = 1;             // 2-wire isoSPI used.
    int1_t isoSPI_IDLE = 1;             // isoSPI in IDLE State (ISOSPI==IDLE).
    int1_t core_SLEEP  = 1;             // LTC6804 Core in SLEEP State (CORE==SLEEP).
    int1_t daisyChain  = 0;             // Addressable Bus Communication.

// CLRCELL VARIABLE DECLARATION:
//  3 Total Devices = (Bottom (B) Index=0) + (Middle (M) Index=1) + (Top (T) Index=2).
   const int8 TOTALDEVICES    = 3;

// A) Function will create a Wake-Up Byte based on LTC6804 Core State vs isoSPI State.
// void = isoSPI_WakeUp_Write(int1 isoSPIMode, int1 isoSPI_IDLE, int1 core_SLEEP,
//                            int1 daisyChain, int8 totalDevices)
   isoSPI_WakeUp_Write(isoSPIMode, isoSPI_IDLE, core_SLEEP,
                       daisyChain, TOTALDEVICES);

// START OF BROADCAST CLRCELL SPI WRITE:

// 1) Pull CSB low.
   output_low(CSB);

// 2) Send CLRCELL command (0x07 0x11) and PEC Bytes (0xC9 0xC0).
   spi_write(0x07);
   spi_write(0x11);
   spi_write(0xC9);
   spi_write(0xC0);

// 3) Pull CSB high.
   output_high(CSB);

// END OF BROADCAST CLRCELL SPI WRITE:
```