بسم تعالی

دانشکده مهندسی برق

VHDL_Report2

نویسنده:

امیرحسین احمدی_401611328

نام استاد:

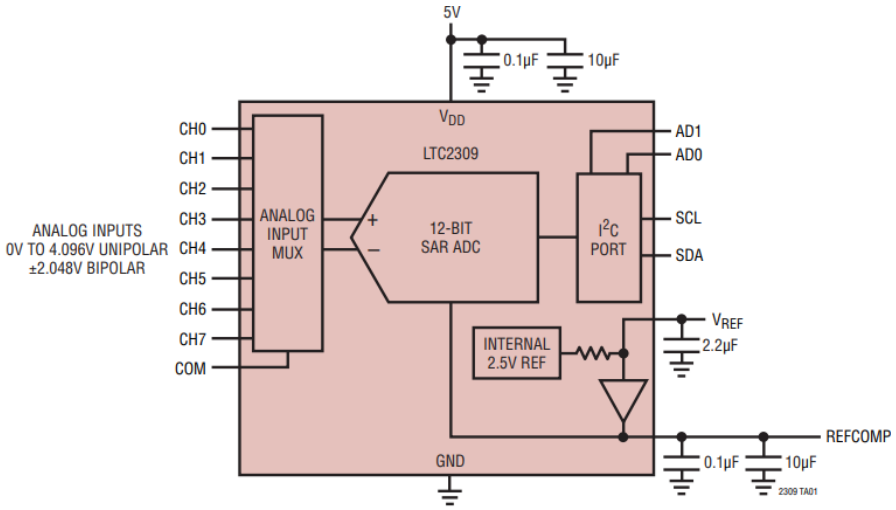
دکتر ستار میرزا کوچکی

تاریخ:

1401/11/04

LTC2309 a 8-channels,12-Bit SAR ADC with I2C Interface:

LTC2309 یک ADC توان پایین با نویز کم و هشت کاناله 12بیتی با یک واسط سریال دوسیمه I2C سازگار با 1+9 آدرس برای سنکرون‌کردن است. دارای یک مرجع داخلی و یک مدار تمام تفاضلی برای کاهش نویز مشترک(Common) هست. و از کلاک داخلی برای رسیدن به فرکانس 769kHz (1.3us) استفاده می‌کند. دارای مالتی‌پلکسر(Mux) هشت کاناله می‌باشد. همچنین در کنترل پردازش صنعتی، کنترل موتور، مانیتورینگ منبع تغذیه و شتاب‌سنج و دسترسی به داده‌های ایزوله و از راه دور کاربرد دارد.



بلوک دیاگرام LTC2309

پین‌های ورودی:

SYMBOL	PARAMETER
V_{IN}^+	Absolute Input Range (CH0 to CH7)
V_{IN}^-	Absolute Input Range (CH0 to CH7, COM)
$V_{IN}^+ - V_{IN}^-$	Input Differential Voltage Range
I_{IN}	Analog Input Leakage Current
C_{IN}	Analog Input Capacitance
CMRR	Input Common Mode Rejection Ratio

ورودی و خروجی‌های دیجیتالی I2C:

SYMBOL	PARAMETER
V_{IH}	High Level Input Voltage
V_{IL}	Low Level Input Voltage
V_{IHA}	High Level Input Voltage for Address Pins A1, A0
V_{ILA}	Low Level Input Voltage for Address Pins A1, A0
R_{INH}	Resistance from A1, A0, to V_{DD} to Set Chip Address Bit to 1
R_{INL}	Resistance from A1, A0 to GND to Set Chip Address Bit to 0
R_{INF}	Resistance from A1, A0 to GND or V_{DD} to Set Chip Address Bit to Float
I_I	Digital Input Current
V_{HYS}	Hysteresis of Schmitt Trigger Inputs
V_{OL}	Low Level Output Voltage (SDA)
t_{OF}	Output Fall Time V_H to $V_{IL(MAX)}$
t_{SP}	Input Spike Suppression
C_{CAX}	External Capacitance Load On-Chip Address Pins (A1, A0) for Valid Float

ویژگی‌های زمانی I2C:

SYMBOL	PARAMETER
f_{SCL}	SCL Clock Frequency
$t_{HD(SDA)}$	Hold Time (Repeated) START Condition
t_{LOW}	LOW Period of the SCL Pin
t_{HIGH}	HIGH Period of the SCL Pin
$t_{SU(STA)}$	Set-Up Time for a Repeated START Condition
$t_{HD(DAT)}$	Data Hold Time
$t_{SU(DAT)}$	Data Set-Up Time
t_r	Rise Time for SDA/SCL Signals
t_f	Fall Time for SDA/SCL Signals
$t_{SU(STO)}$	Set-Up Time for STOP Condition
t_{BUF}	Bus Free Time Between a STOP and START Condition

ویژگی‌های زمانی ADC:

SYMBOL	PARAMETER
f_{SMPL}	Throughput Rate (Successive Reads)
t_{CONV}	Conversion Time
t_{ACQ}	Acquisition Time
$t_{REFWAKE}$	REFCOMP Wake-Up Time (Note 13)

در ارتباط دو سیمه I2C تبدیل‌ها با سیگنال‌کردن یک شرط توقف(Stop) پس‌از این‌که بخش با موفقیت برای یک عملیات خواندن/ نوشتن آدرس‌دهی شد، آغاز می‌شوند. دستگاه تا زمانی که تبدیل به‌پایان نرسد، درخواست خارجی را تایید(NACK) نخواهد کرد. وقتی برای حالت خواندن آدرس‌دهی شود، دستگاه نتیجه تغییر را تحت کنترل کلاک سریال(SCL) روی خروجی قرار می‌دهد که 12 بیت داده خروجی و پس از آن 4 صفر متوالی قرار دارد. با لبه پایین رونده SCL دیتا به‌روز می‌شود که به کاربر اجازه می‌دهد تا داده‌ها را به‌طور قابل اطمینانی روی لبه بالارونده SCL قفل کند. عملیات نوشتن ممکن است با استفاده‌از یک START تکراری یا یک شرط STOP برای شروع یک تبدیل جدید انجام شود. با انتخاب حالت نوشتن، ADC از طریق 6-Bits D_{in} پروگرم می‌شود. (مالتی‌پلکسر از طریق D_{in} پیکربندی می‌شود.)

دیتای ورودی SDI در لبه بالارونده کلاک در حین عمل خواندن بارگیری می‌شود که S/D بیت اول آن و SLP بیت آخر آن در شش لبه بالارونده بعدی بارگیری می‌شوند.

S/D	O/S	S1	S0	UNI	SLP

S/D = SINGLE-ENDED/$\overline{\text{DIFFERENTIAL}}$ BIT

O/S = ODD/$\overline{\text{SIGN}}$ BIT

S1 = CHANNEL SELECT BIT 1

S0 = CHANNEL SELECT BIT 0

UNI = UNIPOLAR/$\overline{\text{BIPOLAR}}$ BIT

SLP = SLEEP MODE BIT

ورودی آنالوگ Mux:

ورودی آنالوگ از طریق بیت‌های O/S، S1 و S0 از D_{in} پروگرم می‌شود. که مطابق جدول زیر است.

S/D	O/S	S1	S0	0	1	2	3	4	5	6	7	COM
0	0	0	0	+	−							
0	0	0	1			+	−					
0	0	1	0					+	−			
0	0	1	1							+	−	
0	1	0	0	−	+							
0	1	0	1			−	+					
0	1	1	0					−	+			
0	1	1	1							−	+	
1	0	0	0	+								−
1	0	0	1			+						−
1	0	1	0					+				−
1	0	1	1							+		−
1	1	0	0		+							−
1	1	0	1				+					−
1	1	1	0						+			−
1	1	1	1								+	−

کلاک تبدیل داخلی:

یک کلاک پیش‌فرض کارخانه‌ای است که به زمان تبدیل t_{CONV} به میزان 1.3us تا ماکزیمم زمان 1.8us دست یابیم.

واسط I2C:

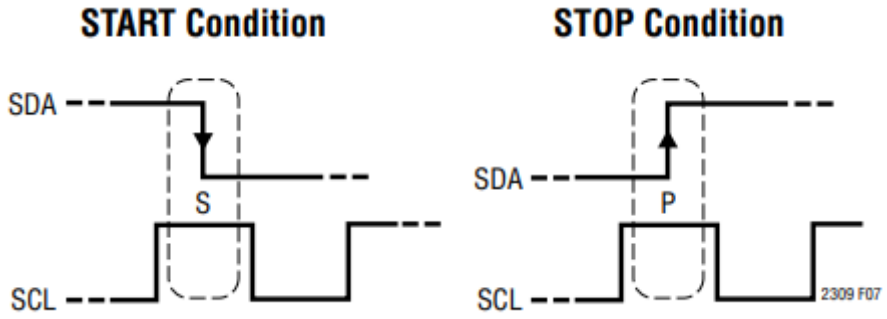
LTC2309 با واسط I2C ارتباط برقرار می‌کند که یک واسط دو سیمه با اتصال Open-Drain است که می‌تواند چند دستگاه و چند Master را روی یک باس پشتیبانی کند. دستگاه متصل فقط می‌تواند SDA را در حال LOW قرار دهد و هیچوقت HIGH نمی‌شود و SDA از طریق مقاومت Pull-Up به تغذیه متصل می‌شود.

دیتا از روی باس I2C از نرخ استاندارد 100kBits/sec تا 400kBits/sec در حالت سریع منتقل می‌شود.

LTC2309 فقط می‌تواند به عنوان Slave آدرس‌دهی شود و می‌تواند D_{IN} را دریافت کند یا نتیجه آخرین تغییرات را انتقال دهد. SCL به‌عنوان ورودی به LTC2309 داده شده و SDA دوطرفه است.

شرایط Stop/Start:

حالت Start وقتی اتفاق می‌افتد که SDA در لبه پایین‌رونده باشد وقتی SCL در حالت HIGH هست و در این حالت باس در وضعیت BUSY قرار می‌گیرد. بعد از انتقال داده حالت Stop وقتی اتفاق می‌افتد که SDA در لبه بالارونده باشد و SCL مجددا در حالت HIGH باشد و در این حالت باس به وضعیت FREE می‌رود. (وضعیت Stop/Start توسط Master تغییر می‌کند.) اگر باس در حال استفاده باشد و وضعیت شروع مکرر (SR) به‌جای وضعیت توقف اتفاق بیافتد، باس همچنان در وضعیت BUSY باقی می‌ماند. زمان شروع مکرر تبعا برابر وضعیت شروع است و برای خواندن/نوشتن از دیوایس قبل از شروع رخداد جدید است.



انتقال دیتا:

بعد از وضعیت Start، باس I2C در حالت Busy بوده و انتقال دیتا بین Master و Slave مذکور انجام می‌شود. دیتا از طریق گروه های 9-Bits منتقل می‌شود که بیت نهم بیت AKC می‌باشد و Master خط SDA را در نه کلاک بعدی آزاد می‌کند و Slave می‌تواند به ACK با قرار دادن SDA در وضعیت LOW پاسخ دهد یا درغیر این صورت به NACK با قرار دادن SDA در وضعیت High-Impedance پاسخ دهد. (مقاومت خارجی خط را در وضعیت HIGH نگه می‌دارد.) انتقال دیتا درصورتی اتفاق می‌افتد که SCL در وضیعت LOW باشد.

قالب داده:

بعد از وضعیت Start، Master یک آدرس 7-Bits به همراه بیت(R/W) ارسال می‌کند (خواندن در وضعیت 1 و نوشتن در وضعیت 0). حال اگر این آدرس 7بیتی به یکی از 9 پایه انتخابی LTC2309 مطابقت داشته باشد، ADC انتخاب می‌شود. اگر ADC حین تغییر آدرس‌دهی شود، درخواست R/W را پاسخ نمی‌دهد و NACK را با قراردادن SDA در وضعیت HIGH صادر می‌کند. وقتی تبدیل انجام شد، LTC2309 با قراردادن SDA در وضعیت LOW، ACK را صادر می‌کند.

LTC2309 دارای دو رجیستر است؛ رجیستر خروجی 12-Bits شامل آخرین تبدیل است و رجیستر ورودی 6-Bits وضعیت عملکرد ADC و ورودی مالتی‌پلکسر را پیکربندی می‌کند.

قالب داده خروجی:

رجیستر خروجی شامل آخرین تبدیل است و بعد از هر تبدیل، دستگاه به‌صورت خودکار به وضعیت Nap یا Sleep Mode بسته‌به وضعیت بیت SLP می‌رود.

وقتی LTC2309 برای حالت خواندن آدرس‌دهی می‌شود، SDA را در وضعیت LOW قرارداده و به‌عنوان فرستنده عمل می‌کند و Master به‌عنوان گیرنده می‌تواند تا 2-Bytes از LTC2309 را بخواند. پس از عمل خواندن باید وضعیت Stop صورت بگیرد تا تبدیل جدید شروع شود. درحالی‌که تبدیل انجام می‌شود، دستگاه عملیات خواندن بعدی را لغو خواهد کرد.

جریان داده خروجی 16-Bits طول داشته و در لبه پایین‌رونده SCL جابه‌جا می‌شود. بیت اول MSB و بیت دوازدهم LSB نتیجه تبدیل خواهدبود و چهار بیت باقی‌مانده صفر هستند؛ در دو حالت دوقطبی و تک‌قطبی هم استفاده می‌شود. داده خروجی در وضعیت مکمل 2 برای خواندن دوقطبی و در حالت مستقیم برای خواندن تک‌قطبی روی SDA قرار می‌گیرد.

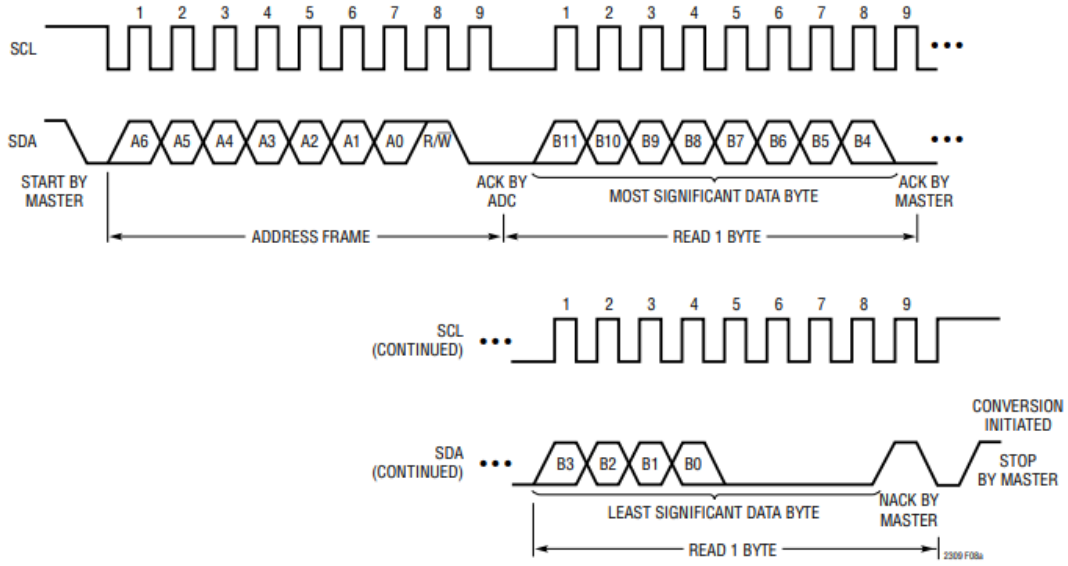


Figure 8a. Timing Diagram for Reading from the LTC2309

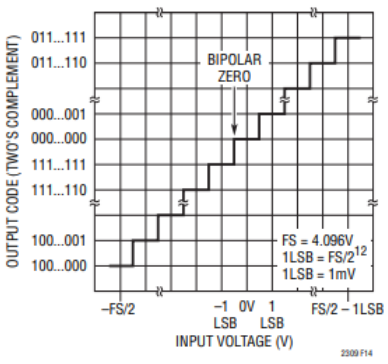


Figure 14. Bipolar Transfer Characteristics (2's Complement)

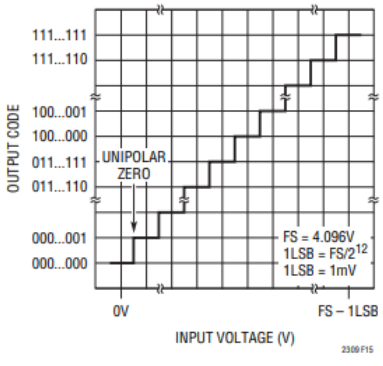


Figure 15. Unipolar Transfer Characteristics (Straight Binary)

قالب داده ورودی:

وقتی LTC2309 برای حالت نوشتن آدرس‌دهی می‌شود، SDA را در وضعیت LOW در زمان پریود LOW قبل از نهمین سیکل قرارداده و به‌عنوان گیرنده عمل می‌کند و Master به‌عنوان فرستنده 1-Byte را برای پروگرم‌کردن ارسال می‌کند که 6-Bits آن همان D_{IN} و دو بیت دیگر Don't care(X) هستند. بیت‌های ورودی در لبه بالارونده SCL در زمان عمل نوشتن قفل می‌شوند

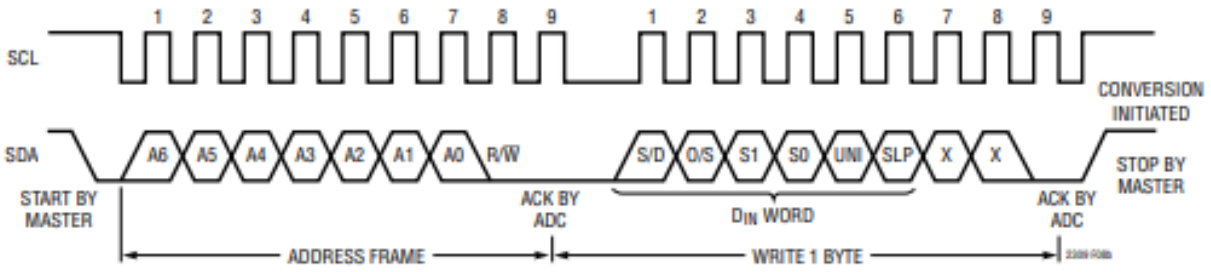


Figure 8b. Timing Diagram for Writing to the LTC2309

بعد از اعمال، ADC یک سیکل ریست داخلی را شروع می‌کند که تمام بیت‌های D_{IN} را صفر می‌کند. اگر وضعیت پیش‌فرض پیکربندی ADC's مطلوب نباشد، عملیات نوشتن ممکن است انجام شود؛ درغیراین‌صورت ADC باید به‌درستی آدرس‌دهی شود و به‌دنبال آن وضعیت Stop تا تبدیل جدید شروع شود.

شروع تبدیل جدید:

LTC2309 از وضعیت Nap/Sleep موقعی تغییر وضعیت می‌دهد که به حالت خواندن یا نوشتن آدرس‌دهی شود. یک فرمان توقف ممکن است پس‌از اجرای عملیات خواندن/نوشتن برای تحریک تبدیل جدید صادر شود. صدور فرمان توقف بعد از هشتمین کلاک SCL از قالب آدرس و قبل از کامل شدن عمل خواندن/نوشتن می‌تواند منجربه شروع تبدیل جدید شود؛ اما خروجی به‌دلیل فقدان زمان کافی معتبر نخواهد بود.

آدرس LTC2309:

LTC2309 دارای دو پایه آدرس AD0 و AD1 می‌باشد که می‌تواند به حالت LOW/HIGH یا شناور(Float) محدود شود تا یکی از 9 آدرس ممکن را فعال کند.

علاوه‌بر این پیکربندی، LTC2309 یک آدرس کلی (1101011) هم دارد که برای سنکرون‌کردن چندین LTC2309s یا بقیه خانواده آن مثل I2C LTC230X SAR ADCs باشد.

Table 2. Address Assignment

AD1	AD0	ADDRESS
LOW	LOW	0001000
LOW	Float	0001001
LOW	HIGH	0001010
Float	HIGH	0001011
Float	Float	0011000
Float	LOW	0011001
HIGH	LOW	0011010
HIGH	Float	0011011
HIGH	HIGH	0101000

خواندن متوالی:

در کاربردهایی که کانال ورودی مشابه هر سیکل نمونه‌برداری می‌شود، تبدیل می‌تواند به‌طور حتم اجرا شود و بدون سیکل نوشتن، بخواند. کلمه D_{IN} از مقدار قبلی بدون تغییر باقی می‌ماند. اگر از ابتدا مقداری برای D_{IN} تعیین نشده‌باشد، مقدار پیش‌فرض 0 برای همه بیت‌های آن درنظر گرفته‌می‌شود. در پایان فرآیند خواندن، باید شرایط توقف برقرار شود تا تبدیل بعدی شروع شود. در نتیجه سیکل تبدیل، نتیجه بعدی را می‌توان با استفاده از روش شرح‌داده‌شده خواند. اگر چرخه تبدیل به پایان نرسد و یک آدرس معتبر دستگاه را انتخاب کند، LTC2309 یک سیگنال لغو NACK ارسال می‌کند که نشان دهد که سیکل تبدیل درحال انجام است.

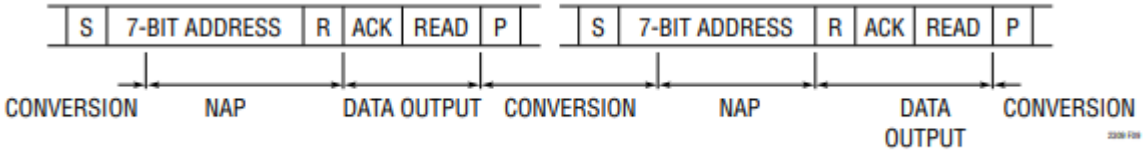


Figure 9. Consecutive Reading with the Same Configuration

خواندن/نوشتن متوالی:

زمانی‌که سیکل تبدیل کامل شد، LTC2309 می‌تواند با استفاده‌از فرمان شروع مکرر(Sr) نوشته و سپس خوانده‌شود. شکل زیر یک سیکل را نشان می‌دهد که با نوشتن دیتا شروع می‌شود، یک شروع مجدد، پس از آن خواندن با فرمان توقف به پایان رسید. پس از آن‌که همه 16 بیت خوانده‌شد، یک تبدیل می‌تواند با صادرکردن یک فرمان توقف شروع شود. تبدیل بعدی با استفاده از داده‌های جدید برنامه‌ریزی‌شده انجام خواهد شد.

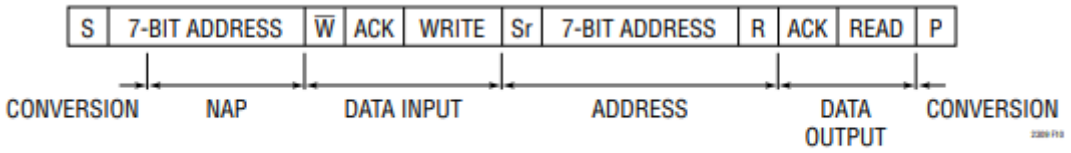


Figure 10. Write, Read, START Conversion

سنکرون‌کردن چندین LTC2309 با آدرس کلی:

در کاربردهایی که چندین LTC2309 یا I2C SAR ADCs از شرکت فن‌آوری خطی در یک باس مشترک I2C استفاده می‌شوند، تمام مبدل‌ها می‌توانند با هم از طریق آدرس کلی(Global address) هم‌زمان شوند(البته تمام مبدل‌ها باید سیکل تبدیل کامل‌شده داشته باشند). Master فرمان شروع را صادر می‌کند، که به دنبال آن آدرس کلی 1101011 هست و پس از آن درخواست نوشتن. تمام مبدل ها انتخاب‌شده می‌شوند؛ سپس Master یک بایت نوشتن همراه با فرمان توقف ارسال می‌کند که این کار انتخاب کانال را به روز می‌کند و به طور همزمان تبدیل برای همه ADC ها روی باس آغاز می‌شود.

برای همزمان کردن چندین مبدل بدون تغییر کانال‌ها، فرمان توقف باید پس از تایید فرمان نوشتن کلی صادر شود. فرمان خواندن کلی مجاز نیست و مبدل‌ها یک فرمان خواندن کلی را لغو (NACK) می‌کنند.

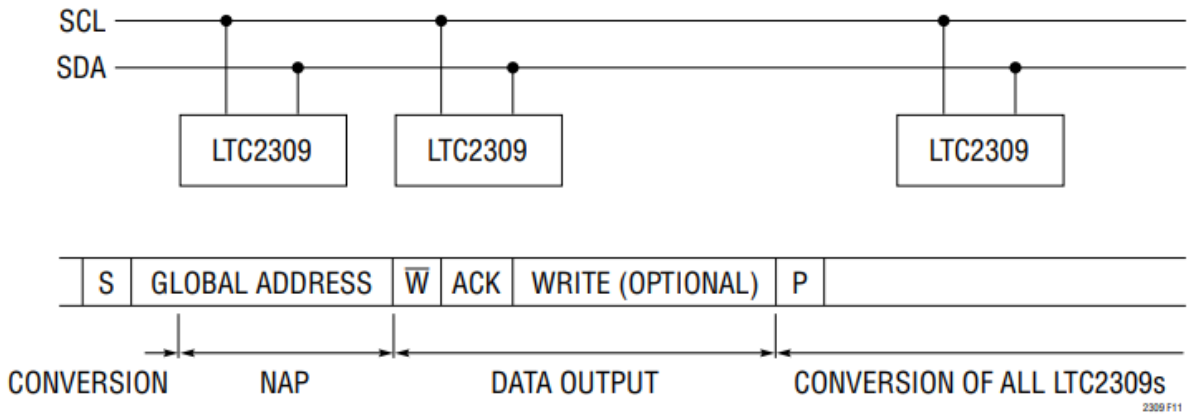


Figure 11. Synchronous Multiple LTC2309s with a Global Address Call

حالت NAP:

ADC پس از آن که یک تبدیل کامل شد(t_{CONV})، به حالت Nap می‌روند به‌شرطی‌که بیت SLP صفر باشد. همچنین میزان جریان تغذیه کاهش می‌یابد(210uA).

حالت Sleep:

ADC پس از آن که یک تبدیل کامل شد(t_{CONV})، به حالت Sleep می‌روند به‌شرطی‌که بیت SLP یک باشد. همچنین میزان جریان تغذیه به شرطی که هیچ یک از ورودی‌های دیجیتال سوئیچ نشوند، کاهش می‌یابد(7uA). وقتی که LTC2309 آدرس‌دهی شد، ADC از حالت Sleep خارج شده و مدت زمان 200ms($t_{REFWAKE}$) طول می‌کشد تا به اصطلاح بیدار شود و قبل از این زمان تبدیلی صورت نمی‌گیرد.

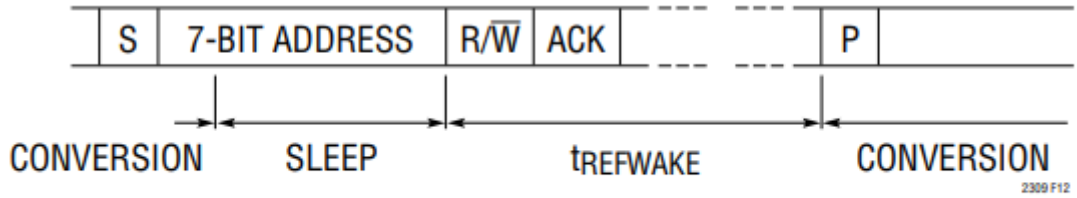


Figure 12. Exiting Sleep Mode and Starting a New Conversion

Acquistion:

LTC2309 شروع به دریافت سیگنال ورودی در موارد مختلف بسته به این که آیا عملیات خواندن یا نوشتن باشد، می‌کند. اگر عمل خواندن اجرا شود، دریافت سیگنال ورودی از لبه بالارونده کلاک نهم به دنبال قالب آدرس شروع می‌شود.

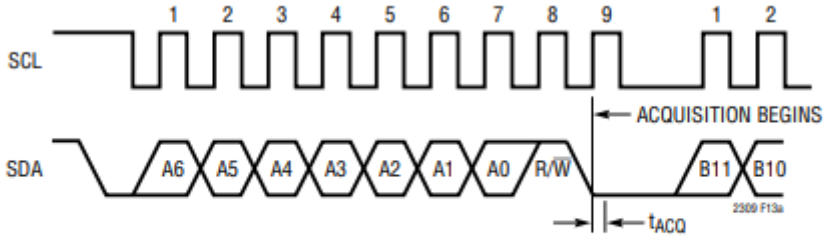


Figure 13a. Timing Diagram Showing Acquisition During a Read Operation

همچنین اگر عمل نوشتن اجرا شود، دریافت سیگنال ورودی از لبه پایین‌رونده کلاک ششم پس از آن که کلمه D_{IN} جابه‌جا شد، شروع می‌شود.

LTC2309 سیگنال را از کانال ورودی‌ای دریافت می‌کند که اخیرا از طریق D_{IN} برنامه‌ریزی‌شده است.

حداقل زمان 240ns نیاز است تا قبل از این که تبدیل شروع شود، سیگنال ورودی دریافت شود.

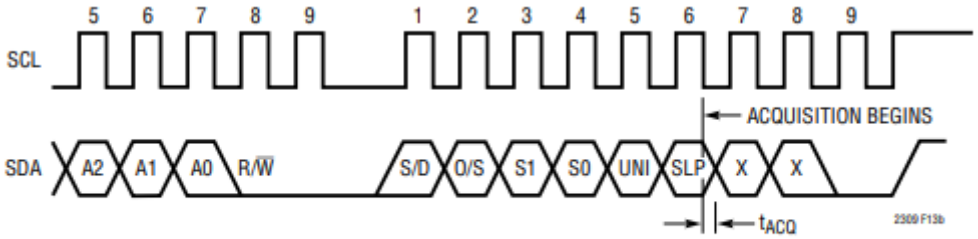
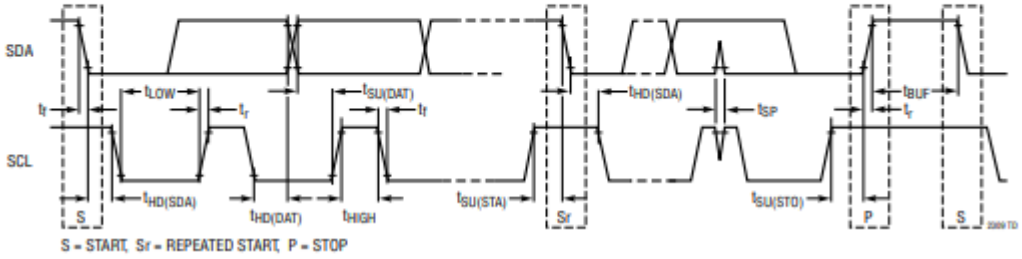


Figure 13b. Timing Diagram Showing Acquisition During a Write Operation

TIMING DIAGRAM

Definition of Timing for Fast/Standard Mode Devices on the I²C Bus



تا اینجا شکل کلی دیتاشیت LTC2309 بررسی شد. حال باید بخش‌های مختلف را شبیه‌سازی کنیم.

در ادامه شکل کد همراه با توضیحات آورده شده است.

کد LTC2309 بخش دیجیتال:

```
--AmirHosein Ahmadi_401611328
--VHDL PROJECT:
--LTC2309
------------------------------------------------------------------   -- LIBRARY
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.NUMERIC_STD.ALL;
LIBRARY UNISIM;
USE UNISIM.VCOMPONENTS.ALL;
------------------------------------------------------------------   -- ENTITY
ENTITY LTC2309 IS
PORT(
     CLK : IN    STD_LOGIC:='0';
     RW  : IN    STD_LOGIC:='1';
     AD0 : IN    STD_LOGIC:='X';
     AD1 : IN    STD_LOGIC:='X';
     SDA : BUFFER STD_LOGIC:='1'
     );
     CONSTANT TCONV    : TIME := 1300 NS;                    -- FOR CONVERSION
     CONSTANT TACQ     : TIME := 240  NS;                    -- FOR ACQUISTION
     CONSTANT TREFWAKE : TIME := 200  mS;                    -- FOR SLEEP MODE
END LTC2309;
```

خب در تصویر بالا پین‌های ورودی CLk,R/W,AD0,AD1 و SDA تعریف می‌شوند. LTC2309 همواره یک SLAVE بوده و باید کلاک از MASTER به آن وارد شود؛ پورت SDA دوطرفه بوده و ما آن را یک BUFFER در نظر می‌گیریم؛ وضعیت خواندن/نوشتن با R/W مشخص می‌شود و آدرس‌دهی از طریق AD0,AD1 انجام می‌شود. در ادامه ثابت‌های زمانی مد نظر تعریف شده است.

کد زیر بخش مربوط به ARCHITECTURE با سیگنال‌های لازمه که توضیحات COMMAND شده است.

```
------------------------------------------------------------------   -- ARCHITECTURE
ARCHITECTURE Behavioral OF LTC2309 IS
SIGNAL SCL_CLK     : STD_LOGIC := '1';                              -- GENERATE INTERNAL SCL CLOCK
SIGNAL SDA_CLK     : STD_LOGIC := '0';                              -- GENERATE INTERNAL SDA CLOCK
SIGNAL SDA_CLK_PRV : STD_LOGIC := '0';                              -- GENERATE INTERNAL SDA CLOCK
SIGNAL SCL         : STD_LOGIC := '1';
SIGNAL AD          : STD_LOGIC_VECTOR(1 DOWNTO 0);                  -- USED TO COMBINE AD1&AD0
SIGNAL ENABLE      : STD_LOGIC := '0';                              -- WHEN LTC2309 CALLED BY MASTER THIS BECOMES 1
--
SIGNAL ADDRESS     : STD_LOGIC_VECTOR(7  DOWNTO 0) := (OTHERS => 'Z'); -- I2C ADDRESS
SIGNAL DATA        : STD_LOGIC_VECTOR(11 DOWNTO 0) := (OTHERS => 'X'); -- ADC DATA FOR READ MODE
SIGNAL DAT         : STD_LOGIC_VECTOR(11 DOWNTO 0) := (OTHERS => 'X'); -- ADC DATA FOR WRITE MODE
SIGNAL DIN         : STD_LOGIC_VECTOR(5  DOWNTO 0) := (OTHERS => '0'); -- DIN 6BIT FOR MUX CONFIG OF ADC IN WRITE MODE
                                                                    -- DIN(S/D,O/S,S1,S0,UNI,,SLP)
                                                                    -- S/D : SINGLE-ENDED/DIFFERENTIAL BIT
                                                                    -- O/S : ODD/SIGN BIT
                                                                    -- S1  : CHANNEL SELECT BIT 1
                                                                    -- S0  : CHANNEL SELECT BIT 0
                                                                    -- UNI : UNIPOLAR/BIPOLAR BIT
                                                                    -- SLP : SLEEP/NAP MODE BIT
--
SIGNAL TEMP        : STD_LOGIC_VECTOR(3  DOWNTO 0) := DIN(5 DOWNTO 2); -- USED FOR DATA SELECTION FROM ADC
SIGNAL TEMP1       : STD_LOGIC_VECTOR(3  DOWNTO 0) := DIN(5 DOWNTO 2); -- USED FOR DATA SELECTION FROM ADC
SIGNAL BUSS          : STD_LOGIC :='0';
--
SIGNAL SLV_ACK       : STD_LOGIC :='0';                             -- 8BITS ADDRESS&RW BIT SLAVE ACK
SIGNAL DIN_REC       : STD_LOGIC :='0';                             -- 8BITS ADDRESS&RW BIT SLAVE ACK
SIGNAL MAS_ACK       : STD_LOGIC :='0';                             -- 8BITS MSB DATA MAS ACK
SIGNAL MAS_NACK      : STD_LOGIC :='0';                             -- 8BITS LBS DATA WITH 4 NIBBLE 0 MAS ACK
SIGNAL DATA_READY    : STD_LOGIC :='0';
SIGNAL NIBBLE_READY  : STD_LOGIC :='0';
SIGNAL STOP          : STD_LOGIC :='0';
--
SIGNAL ONCE_SLV_ACK  : STD_LOGIC :='0';
SIGNAL ONCE_MAS_ACK  : STD_LOGIC :='0';
SIGNAL ONCE_MAS_NACK : STD_LOGIC :='0';
BEGIN
```

کد زیر برای بدست آوردن آدرس از روی AD0,AD1 می‌باشد.(در طول ارتباط فرض می‌شود آدرسی که میدهیم حتما یکی از این آدرس‌ها است و دیگر شرط تطبیق آدرس‌ها را چک نمی‌کنیم.(فرض است ولی به‌طور کلی باید چک شود.)

```
61  ------------------------------------------------------------------  -- SET I2C ADDRESS DEPPEND ON AD1 & AD0 INPUT
62  I2C_ADDRESS: PROCESS(CLK)
63     BEGIN
64        AD <= AD1 & AD0;
65        CASE AD IS
66           WHEN "00"   => ADDRESS <= "0001000" & RW; ENABLE <= '1';
67           WHEN "0Z"   => ADDRESS <= "0001001" & RW; ENABLE <= '1';
68           WHEN "01"   => ADDRESS <= "0001010" & RW; ENABLE <= '1';
69           WHEN "Z1"   => ADDRESS <= "0001011" & RW; ENABLE <= '1';
70           WHEN "ZZ"   => ADDRESS <= "0011000" & RW; ENABLE <= '1';
71           WHEN "Z0"   => ADDRESS <= "0011010" & RW; ENABLE <= '1';
72           WHEN "10"   => ADDRESS <= "0011010" & RW; ENABLE <= '1';
73           WHEN "1Z"   => ADDRESS <= "0011011" & RW; ENABLE <= '1';
74           WHEN "11"   => ADDRESS <= "0101000" & RW; ENABLE <= '1';
75           WHEN OTHERS => ADDRESS <= "ZZZZZZZZ"    ; ENABLE <= '0';
76        END CASE;
77  END PROCESS I2C_ADDRESS;
```

بخش زیر برای چک کردن دیتای ADC با توجه به وضعیت چهار پین DIN MSB که در حالت Write ورودی LTC2309 است مشخص می‌شود. (در اینجا چون نمی‌شد ورودی آنالوگ به شبیه‌ساز داد، مقادیر پیش‌فرض برای هر حالت در نظر گرفته شده است.

```
78  ------------------------------------------------------------------  -- ADC DATA IS SELECTIVE BY DIN (S/D,O/S,S1,S0)
79  --NOTE: ADC DIDN'T SIMULATE IN THIS PROJECT AND JUST SET DEFAULT DATA FOR EVERY DIN SELECTION
80  ADC_DATA: PROCESS(CLK)
81     BEGIN
82        TEMP <= DIN(5 DOWNTO 2);
83        CASE TEMP IS                                    -- S/D O/S S1 S0 CH0 CH1 CH2 CH3 CH4 CH5 CH6 CH7 COM
84           WHEN "0000" => DATA <= "010000001111";       --  0   0  0  0   +   -
85           WHEN "0001" => DATA <= "010000101111";       --  0   0  0  1           +   -
86           WHEN "0010" => DATA <= "010001001111";       --  0   0  1  0                   +   -
87           WHEN "0011" => DATA <= "010001101111";       --  0   0  1  1                           +   -
88           WHEN "0100" => DATA <= "010010001111";       --  0   1  0  0   -   +
89           WHEN "0101" => DATA <= "010010101111";       --  0   1  0  1           -   +
90           WHEN "0110" => DATA <= "010011001111";       --  0   1  1  0                   -   +
91           WHEN "0111" => DATA <= "010011101111";       --  0   1  1  1                           -   +
92           WHEN "1000" => DATA <= "010100001111";       --  1   0  0  0   +                               -
93           WHEN "1001" => DATA <= "010100101111";       --  1   0  0  1       +                           -
94           WHEN "1010" => DATA <= "010101001111";       --  1   0  1  0           +                       -
95           WHEN "1011" => DATA <= "010101101111";       --  1   0  1  1               +                   -
96           WHEN "1100" => DATA <= "010110001111";       --  1   1  0  0                   +               -
97           WHEN "1101" => DATA <= "010110101111";       --  1   1  0  1                       +           -
98           WHEN "1110" => DATA <= "010111001111";       --  1   1  1  0                           +       -
99           WHEN "1111" => DATA <= "010111101111";       --  1   1  1  1                               +   -
100          WHEN OTHERS => NULL;
101       END CASE;
102 END PROCESS ADC_DATA;
```

در بخش بعدی فرض بر این است فرکانس کلاک سیستم و کلاک کاری ما متفاوت است؛ لذا از یک کانتر برای تولید SCLو SDA که اختلاف فاز 90 درجه دارند استفاده می‌شود. در اینجا کلاک اصلی 100MHz بوده و با یک کانتر 1000تایی آن را به مود استاندارد 100kHz می‌رسانیم. در اینجا بخش استاندارد بررسی شده اما می‌شد بخش fastmode را هم با یک کانتر دیگر درست کرد.

```
103 ------------------------------------------------------------------  -- GENERATE SDA/SCL_CLOCK FOR STANDARD MODE 100kHZ
104 SDA_CLOCK: PROCESS(CLK)
105    VARIABLE Count: INTEGER RANGE 0 TO 1000 :=0;
106    BEGIN
107       IF(RISING_EDGE(CLK)) THEN
108          SDA_CLK_PRV <= SDA_CLK;
109          IF(Count=999) THEN
110             Count :=0;
111          Else
112             Count := Count+1;
113          END IF;
114          CASE Count IS
115             WHEN 0   TO 249 => SCL_CLK <= '0' ; SDA_CLK<= '0';
116             WHEN 250 TO 499 => SCL_CLK <= '0' ; SDA_CLK<= '1';
117             WHEN 500 TO 749 => SCL_CLK <= '1' ; SDA_CLK<= '1';
118             WHEN OTHERS     => SCL_CLK <= '1' ; SDA_CLK<= '0';
119          END CASE;
120       END IF;
121 END PROCESS SDA_CLOCK;
122
```

بخش زیر برای حالت READ نوشته شده است. دراین کد برای آدرس،دیتا ، نیبل صفر و ورودی DIN کانترهای مختلف تعریف شده است. در اینجا با درست بودن آدرس مقدار ENABLE=1 شده و وارد بخش START می‌شود، با وارد شدن به بخش START، باس به حالت BUSY می‌رود.

```
123  ------------------------------------------------------------------  -- MAIN PROCESS OF I2C
124  MAIN: PROCESS(CLK)
125  VARIABLE ADDRESS_CNT : INTEGER  RANGE -1 TO 7  := 7;                        -- ADDRESS COUNTER
126  VARIABLE DATA_CNT    : INTEGER  RANGE -1 TO 11 := 11;                       -- DATA    COUNTER
127  VARIABLE ZERO_CNT    : INTEGER  RANGE -1 TO 3  := 3;                        -- ZERO    COUNTER
128  VARIABLE DIN_CNT     : INTEGER  RANGE -1 TO 7  := 7;                        -- DIN    | COUNTER
129
130  BEGIN
131  IF(RW='1') THEN
132     IF(ENABLE='0') THEN
133        SCL <='1';
134     ELSIF(FALLING_EDGE(CLK)) THEN                                            -- WHEN LTC2309 CALLED BY MASTER
135        SCL <= SCL_CLK;
136        --
137        IF(BUSS='0' AND ENABLE='1') THEN
138           SCL<='1';
139        END IF;
140        --
141        --
142        IF(((SDA_CLK='0' AND SDA_CLK_PRV='1') AND SCL_CLK='1') AND BUSS='0') THEN-- START CONDITION AND BUS GETS BUSY
143           SDA         <= '0';
144           ADDRESS_CNT :=  7 ;
145           --
146           DATA_READY  <= '0';                                                -- THESE ARE FLAGS MUST TO GET RESET VALUE
147           SLV_ACK     <= '0';                                                -- THIS BRANCH APPLYS ON BEGINING OF START
148           MAS_ACK     <= '0';
149           MAS_NACK    <= '0';
150           STOP        <= '0';
151           BUSS        <= '1';
152        END IF;
153        --
```

در اینجا آدرس بر روی SDA قرار می‌گیرد و فرض بر این است که آدرس‌ها در نهایت مساوی است و ارتباط برقرار می‌شود، پس از برقراری ارتباط پرچم DATA_READY فعال شده و آماده خواندن دیتا از LTC می‌باشد(12بیت دیتای ADC به همراه نیبل صفر که در دو بایت منتقل می‌شود.)

```
154        IF((SDA_CLK='1' AND SDA_CLK_PRV='0') AND SCL_CLK='0' AND BUSS='1') THEN  -- ADDRESS SET ON RISING EDGE OF SDA_CLK AND SCL='0'
155           IF(ADDRESS_CNT >= 0) THEN                                           -- ADDRESS SET IN SDA IN 7 CYCLES
156              SDA <= ADDRESS(ADDRESS_CNT);
157              ADDRESS_CNT := ADDRESS_CNT-1;
158           ELSIF(ADDRESS_CNT < 0 AND SLV_ACK='0') THEN                         -- SLAVE_ACK (BYTE RECEIVED)
159              SDA     <= '0';
160              SLV_ACK <= '1';
161           END IF;
162           --
163           IF(SLV_ACK = '1' AND DATA_READY = '0') THEN                         -- SLAVE IS READY TO TRANSFER DATA
164              DATA_READY <= '1';
165              --                                                               -- THIS BRANCH APPLYS BEFORE DATA TRANSFER
166              DATA_CNT   := 11 ;                                               -- THIS COUNTER MUST GET RESET VALUE
167           END IF;
168           -----------------
```

بخش زیر برای دریافت 12 بیت دیتا از ADC به مستر است، در اینجا باید در دوبایت دیتا منتقل شود، لذا

```
169          IF(DATA_READY='1' AND RW='1') THEN                                  -- READ OPERATION
170             ONCE_SLV_ACK <='1';
171             --t conv
172             --
173             IF(DATA_CNT >=4) THEN                                            -- 8BITS MSB DATA READING
174                SDA      <= DATA(DATA_CNT);
175                DATA_CNT := DATA_CNT -1;
176             ELSIF((DATA_CNT <4 AND DATA_CNT>=0) AND MAS_ACK='0') THEN         -- MAS_ACK (MSB BYTE RECEIVED)
177                SDA      <= '0';
178                MAS_ACK  <= '1';
179             END IF;
180             IF(MAS_ACK='1' AND NIBBLE_READY='0') THEN
181                ZERO_CNT     :=  3 ;
182                NIBBLE_READY <= '1';
183             END IF;
184             --
185             IF(DATA_CNT >=0 AND NIBBLE_READY='1') THEN                       -- 4BITS LBS DATA READING
186                ONCE_MAS_ACK <= '1';
187                --
188                SDA      <= DATA(DATA_CNT);
189                DATA_CNT := DATA_CNT -1;
190             --
191             ELSIF(DATA_CNT < 0 AND ZERO_CNT >= 0) THEN                       -- 4BITS ZEROZS DATA READING
192                SDA      <= '0';                                             -- DON'T CARE NIBBLE FOR ADC
193                ZERO_CNT := ZERO_CNT -1;
194             ELSIF(DATA_CNT < 0 AND ZERO_CNT <0 AND MAS_NACK='0') THEN        -- MAS_NACK (LSB BYTE RECEIVED
195                SDA      <= '1';
196                MAS_NACK <= '1';
197             END IF;
198             --
199             IF(MAS_NACK='1' AND STOP='0') THEN                               -- TRANSFER COMPLETE
200                STOP     <= '1';
201                SDA      <= '0';
202             END IF;
```

برای دریافت 8بیت باارزش از نه کلاک به همراه پرچم MAS_ACK استفاده شده و در دریافت 8 بیت بعدی، ابتدا چهار بیت کم ارزش منتقل شده و سپس چهار بیت نیبل صفر به همراه پرچم MAS_NACK منتقل می‌شود. در این حالت برای اثر گذاشتن روی SCL، پرچم‌های دیگری تعریف شده اند که باعث عوض شدن SCL شود،(هر 9 سیکل باید LOW SCL، شود تا بفهمیم دیتا منقل شده است.)

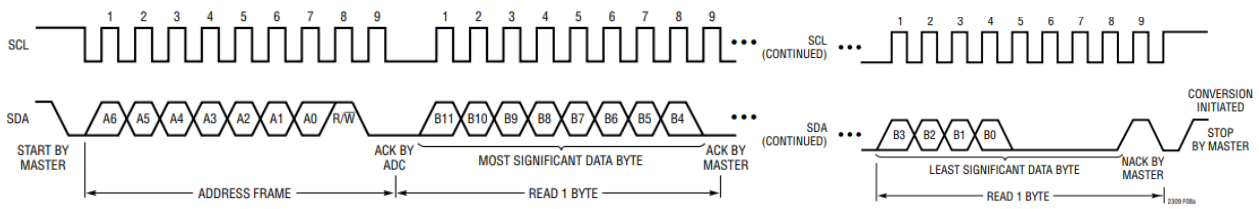
```
203             --
204             IF((SDA_CLK='1' AND SDA_CLK_PRV='0') AND SCL_CLK='0' AND STOP='1') THEN
205                SCL  <= '1';
206                ONCE_MAS_NACK <='1';
207             END IF;
208          --
209          END IF;
210          --
211       END IF;
212          IF((SLV_ACK='1'    AND DATA_READY='1'  ) AND ONCE_SLV_ACK='0' ) THEN SCL <= '0'; END IF;
213          IF((MAS_ACK='1'    AND NIBBLE_READY='1') AND ONCE_MAS_ACK='0' ) THEN SCL <= '0'; END IF;
214          IF(((MAS_NACK='1') AND STOP = '1'      ) AND ONCE_MAS_NACK='0') THEN SCL <= '1'; END IF;
215    END IF;
216    IF ONCE_MAS_NACK='1' THEN BUSS <= '0'; SDA<='1'; END IF;
217 END IF;
218 ---------------------------------------------------------------------------------
```

در ادامه برای بررسی این کد یک تست بنچ تعریف شده که به صورت کد صفحه بعد است.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
--------------------------------------------------------------------
ENTITY LTC2309_TB IS END LTC2309_TB;
--------------------------------------------------------------------
ARCHITECTURE behavior OF LTC2309_TB IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT LTC2309
    PORT(CLK,RW,AD0,AD1 : IN  std_logic;
         SDA        : BUFFER  std_logic);
    END COMPONENT;
   signal CLK : std_logic := '0';   --Inputs
   signal RW  : std_logic := '0';
   signal AD0 : std_logic := 'X';
   signal AD1 : std_logic := 'X';
   signal SDA : std_logic := '1';   --BiDirs
   constant CLK_period : time := 10 ns; -- Clock period definitions
BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: LTC2309 PORT MAP (CLK => CLK,RW  => RW,AD0 => AD0,AD1 => AD1,SDA => SDA);
   -- Clock process definitions
   CLK_process :process
   begin
      CLK <= '0';wait for CLK_period/2;
      CLK <= '1';wait for CLK_period/2;
   end process;
    -- Stimulus process
   stim_proc: process
   begin
      AD0 <= 'X';AD1 <= 'X';wait for CLK_period*2000;
      AD0 <= '0';AD1 <= 'Z';RW  <= '1';wait;
   end process;
END;
```

برای این تست بنچ خروجی به صورت زیر در می‌آید.

همانطور که مشاهده می‌شود، شکل موج زرد SCL و شکل موج بنفش SDA شکل‌موج دیتاشیت است.



بخش زیر برای حالت WRITE نوشته شده است. بخش اول دریافت آدرس مشابه قسمت خواندن است

(البته ما فرض کردیم که آدرس درست فراخوانی می‌شود وگرنه باید شرط تطابق آدرس فراخوانی با آدرس یکی باشد. تا مستر و اسیلو کانکت شوند.)

```
218  ---------------------------------------------------------------------------
219  IF(RW='0') THEN
220     IF(ENABLE='0') THEN
221        SCL <='1';
222     ELSIF(FALLING_EDGE(CLK)) THEN                                    -- WHEN LTC2309 CALLED BY MASTER
223        SCL <= SCL_CLK;
224        --
225        IF(BUSS='0' AND ENABLE='1') THEN
226           SCL<='1';
227        END IF;
228        --
229        --
230        IF(((SDA_CLK='0' AND SDA_CLK_PRV='1') AND SCL_CLK='1') AND BUSS='0') THEN-- START CONDITION AND BUS GETS BUSY
231           SDA         <= '0';
232           ADDRESS_CNT :=  6;
233           --
234           DATA_READY  <= '0';                                        -- THESE ARE FLAGS MUST TO GET RESET VALUE
235           SLV_ACK     <= '0';                                        -- THIS BRANCH APPLYS ON BEGINING OF START
236           MAS_ACK     <= '0';
237           MAS_NACK    <= '0';
238           DIN_REC     <= '0';
239           STOP        <= '0';
240           BUSS        <= '1';
241        END IF;
242        --
243        IF((SDA_CLK='1' AND SDA_CLK_PRV='0') AND SCL_CLK='0' AND BUSS='1') THEN  -- ADDRESS SET ON RISING EDGE OF SDA_CLK AND SCL='0'
244           IF(ADDRESS_CNT >= 0) THEN                                  -- ADDRESS SET IN SDA IN 7 CYCLES
245              IF(SDA ='X' OR SDA='1') THEN
246              ADDRESS(ADDRESS_CNT)<= '1';
247              ELSIF(SDA ='0') THEN
248              ADDRESS(ADDRESS_CNT)<= '0';
249              END IF;
250              ADDRESS_CNT := ADDRESS_CNT-1;
251           ELSIF(ADDRESS_CNT < 0 AND SLV_ACK='0') THEN                -- SLAVE_ACK (BYTE RECEIVED)
252              SDA     <= '0';
253              SLV_ACK <= '1';
254           END IF;
255           --
256           IF(SLV_ACK = '1' AND DATA_READY = '0') THEN                -- SLAVE IS READY TO TRANSFER DATA
257              DATA_READY <= '1';
258              --                                                        -- THIS BRANCH APPLYS BEFORE DATA TRANSFER
259              DATA_CNT   := 11 ;                                        -- THIS COUNTER MUST GET RESET VALUE
260           END IF;
261           ------------
```

در بخش خواندن باید DIN شیش بیتی به همراه دو DON’T CARE در بخش داده روی SDA قرار بگیرد. با اعمال این بیت‌ها به DIN می‌توان داده ی ADC را تغییر داد. پس از دریافت داده پرچم DIN_REC فعال می‌شود ونشان از این است که داده دریافت شد.

```
261           ------------
262           IF(DATA_READY='1' AND RW='0') THEN                        -- WRITE OPERATION
263              ONCE_SLV_ACK <= '1';
264              DIN_CNT      :=  7 ;
265              IF(DIN_CNT>=0) THEN
266                 IF(DIN_CNT>=2) THEN
267                    IF(SDA ='X' OR SDA='1') THEN
268                       DIN(DIN_CNT-2)<= '1';
269                    ELSIF(SDA ='0') THEN
270                       DIN(DIN_CNT-2)<= '0';
271                    END IF;
272                 END IF;
273                 DIN_CNT := DIN_CNT-1;
274              ELSIF(DIN_CNT<0 AND SLV_ACK='1' AND DIN_REC='0') THEN
275                 DIN_REC <= '1';
276                 SDA <= '0';
277              END IF;
```

در قسمت بعدی با توجه به اینکه دیتای ADC باید تغییر کند، پس باید یک CASE بنوسیم که بتوان

```
278          ------------------------------------------------------- -- ADC DATA IS SELECTIVE BY DIN (S/D,O/S,S1,S0)
279          --NOTE: ADC DIDN'T SIMULATE IN THIS PROJECT AND JUST SET DEFAULT DATA FOR EVERY DIN SELECTION
280          IF (DIN_CNT <0) THEN
281             TEMP1 <= DIN(5 DOWNTO 2);
282             CASE TEMP1 IS                                          -- S/D O/S S1 S0 CH0 CH1 CH2 CH3 CH4 CH5 CH6 CH7 COM
283                WHEN "0000" => DAT <= "010000001111";               --  0   0  0  0   +   -
284                WHEN "0001" => DAT <= "010000101111";               --  0   0  0  1           +   -
285                WHEN "0010" => DAT <= "010001001111";               --  0   0  1  0                   +   -
286                WHEN "0011" => DAT <= "010001101111";               --  0   0  1  1                           +   -
287                WHEN "0100" => DAT <= "010010001111";               --  0   1  0  0   -   +
288                WHEN "0101" => DAT <= "010010101111";               --  0   1  0  1           -   +
289                WHEN "0110" => DAT <= "010011001111";               --  0   1  1  0                   -   +
290                WHEN "0111" => DAT <= "010011101111";               --  0   1  1  1                           -   +
291                WHEN "1000" => DAT <= "010100001111";               --  1   0  0  0   +                                   -
292                WHEN "1001" => DAT <= "010100101111";               --  1   0  0  1       +                               -
293                WHEN "1010" => DAT <= "010101001111";               --  1   0  1  0           +                           -
294                WHEN "1011" => DAT <= "010101101111";               --  1   0  1  1               +                       -
295                WHEN "1100" => DAT <= "010110001111";               --  1   1  0  0                   +                   -
296                WHEN "1101" => DAT <= "010110101111";               --  1   1  0  1                       +               -
297                WHEN "1110" => DAT <= "010111001111";               --  1   1  1  0                           +           -
298                WHEN "1111" => DAT <= "010111101111";               --  1   1  1  1                               +   -
299                WHEN OTHERS => NULL;
300             END CASE;
301          END IF;
302          --
303          IF(DIN(1)='0') THEN                                       --BIPOLAR
304          DATA <= NOT(DAT(11)) & DAT(10 DOWNTO 0);
305          ELSE DATA<= DAT;
306          END IF;
307          --
308             END IF;
309          END IF;
310       END IF;
311    IF((SLV_ACK='1'     AND DATA_READY='1'   ) AND ONCE_SLV_ACK='0' ) THEN SCL <= '0'; END IF;
312    IF((SLV_ACK='1'     AND DIN_REC    ='1'  ) AND ONCE_SLV_ACK='1' ) THEN SCL <= '1'; END IF;
313    END IF;
314
315    --
316    END PROCESS MAIN;
317    --
318    END Behavioral;
```

مقدار دیتای LTC را تغییر داد. پس از مشخص شدن مقدار دیتا به کمک چهار بیت MSB باید دو بیت باقی را هم حساب کنیم، اگر داده قرار باشد به صورت BIPOLAR به شرط UNI=0 باشد، باید مقدار 2'COMPLEMENT دیتا را نگه داریم درغیر این صورت دیتا تغییر نمی‌کند.

همچنین برای بیت SLP=NAP/SLEEP اگر LTC به حالت استراحت برود تا زمان $T_{REFWAKE}$ نمی‌توان از آن استفاده کرد و باید منتظر ماند تا این زمان تمام شود.

خروجی این بخش بنا به دلایل شبیه‌سازی بدست نیامد اما الگوریتم مشابه دیتاشیت می‌باشد.

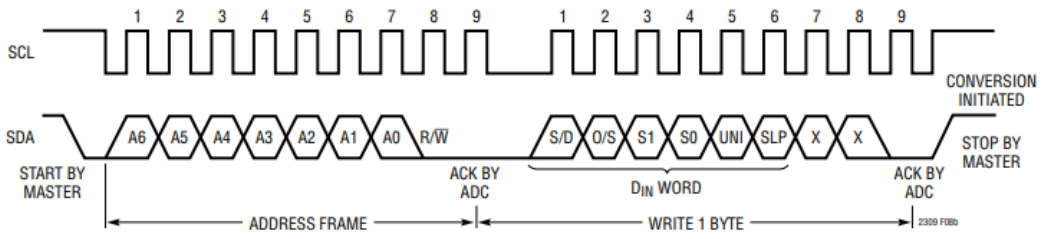


Figure 8b. Timing Diagram for Writing to the LTC2309