**به نام خدا**

**فایل گزارش پروژه**

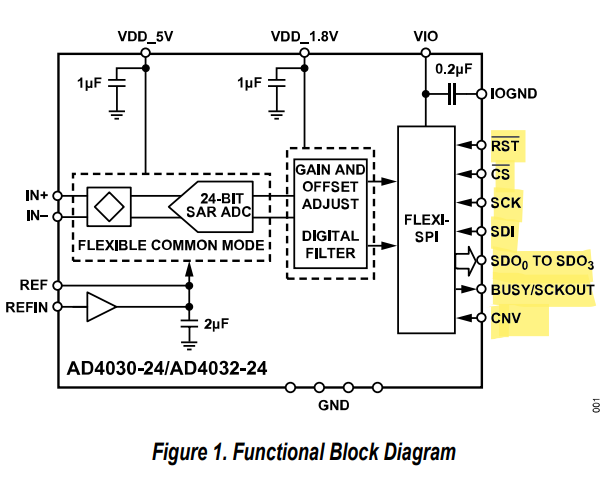
**درس VHDL**

**Designing Hardware Block to handle AD4030 ADC with VHDL**

**رامین اصیلی**

**401611022**

**شمای المان های داخل ADC :**

****

ارتباط با این ADC 24 بیتی توسط پروتکل SPI است. این آیسی قابلیت ارسال دیتای spi به صورت 1-line , 2-line و 4-line

(Quad SPI) را دارا میباشد.

**شرح پایه های مورد نیاز برای طراحی بلوک سخت افزاری:**

* **SDO0 SDO1 SDO2 SDO3**

خروجی دیتای سریال . نتایج تبدیل روی این پایه ها قرار میگیرد . این پایه ها با SCK سنکرون شده است

* **SDI**

ورودی دیتای سریال .

* **SCK**

ورودی کلاک دیتای سریال

* **CS**

ورودی chip select

* **RST**

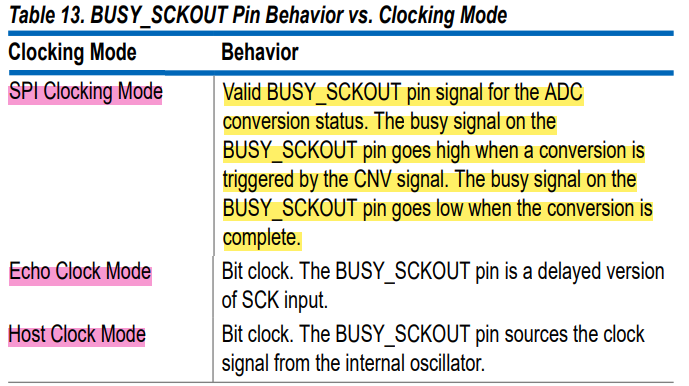
ورودی Active Low ریست

* **CNV**

ورودی convert . لبه ی بالا رونده روی این پایه ، device را روشن میکند و device شروع به convert میکند.

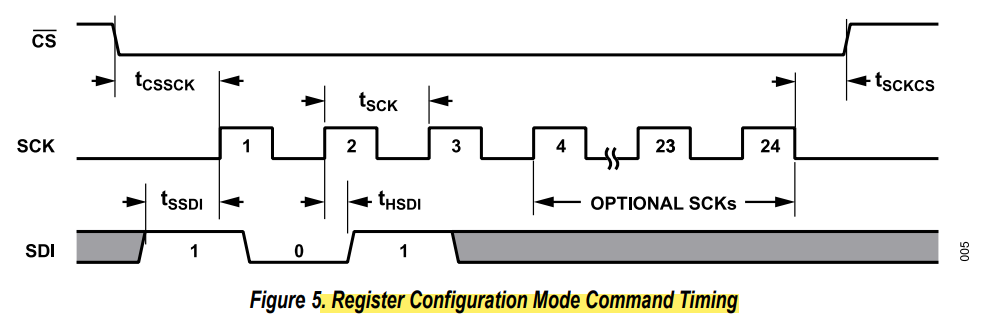
* **BUSY\_SCKOUT**

نشان دهنده ی مشغول بودن ADC وقتی که در مود SPI Clocking است. این پین در شروع یک تبدیل 1 میشود و زمانی که فرایند تبدیل پایان یافت 0 میشود. وقتی در مود SCKOUT است این پین یک بازتاب از کلاک ورودی توسط master یا منبع کلاک داخلی است.

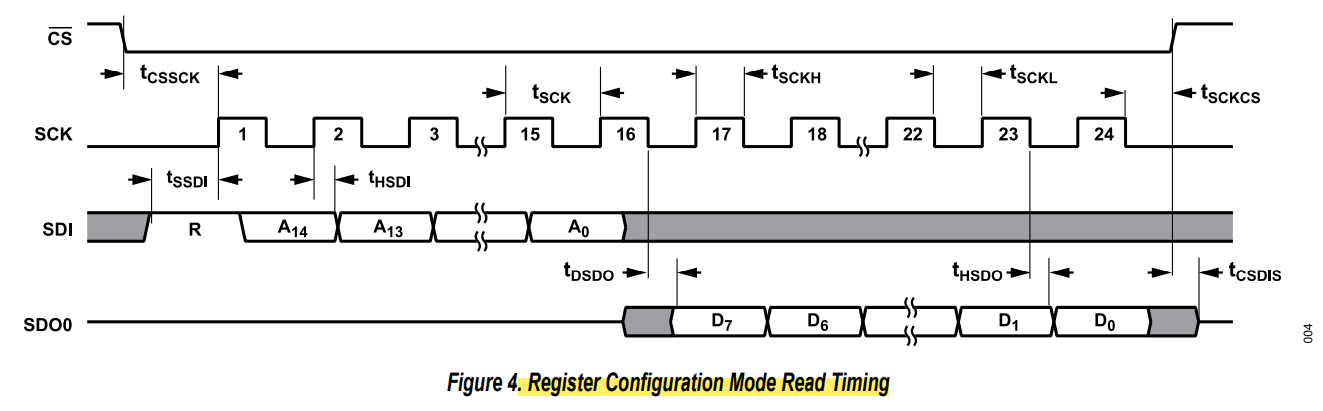


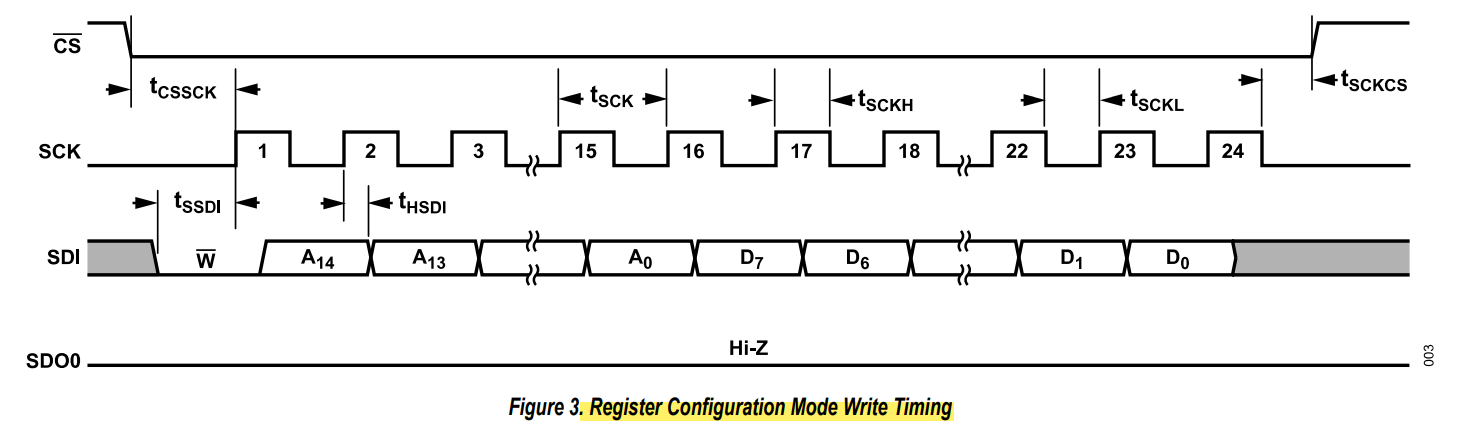
**مود دسترسی به رجیستر ها**

به صورت پیش فرض در زمان روشن شدن دستگاه در مود conversion است بنابر این برای دسترسی کاربر به رجیستر ها باید دستور مخصوصی توسط master ارسال شود. مانند زیر:



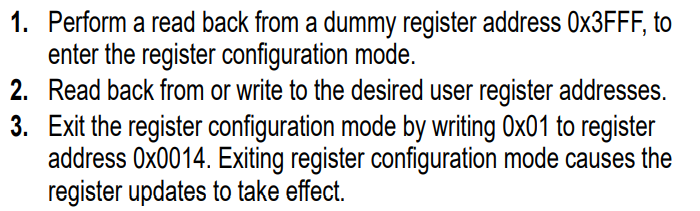
همچنین برای خواندن از و نوشتن در رجیستر ها باید طبق الگوی زیر رفتار کنیم:



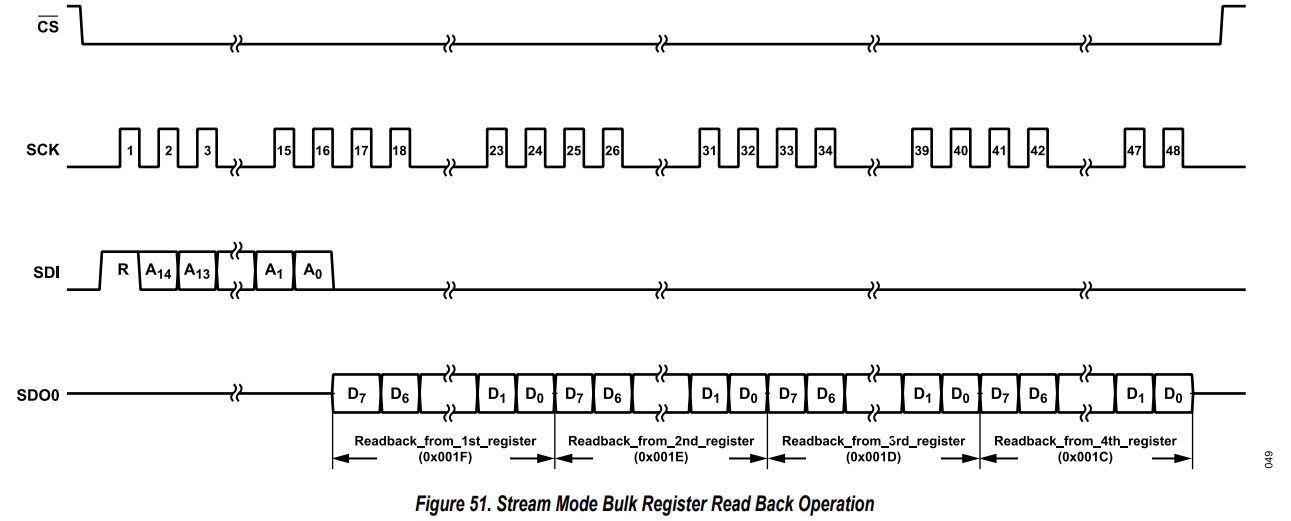


بعد از عمل خواندن و نوشتن رجیستر ها از این مود خارج شویم. خارج شدن از مود تنظیمات رجیستر با نوشتن مقدار 0x01 در رجیستر EXIT configuration با آدرس 0x0014 میباشد.

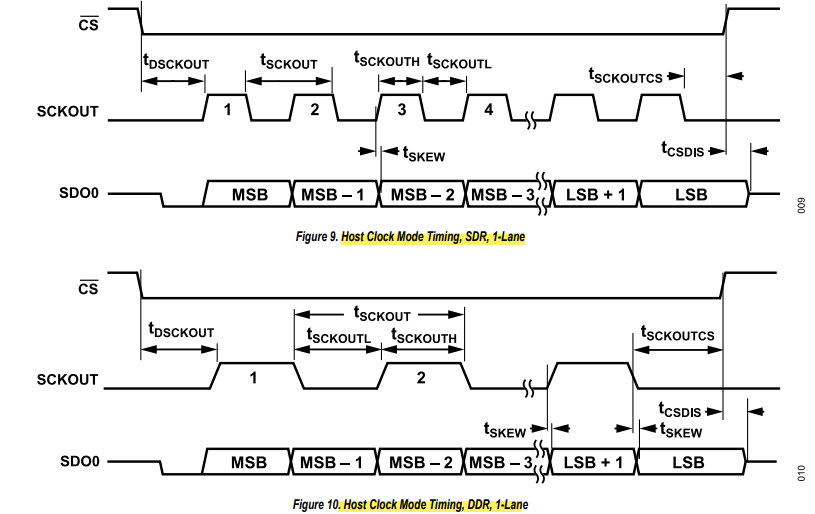
به طور خلاصه:

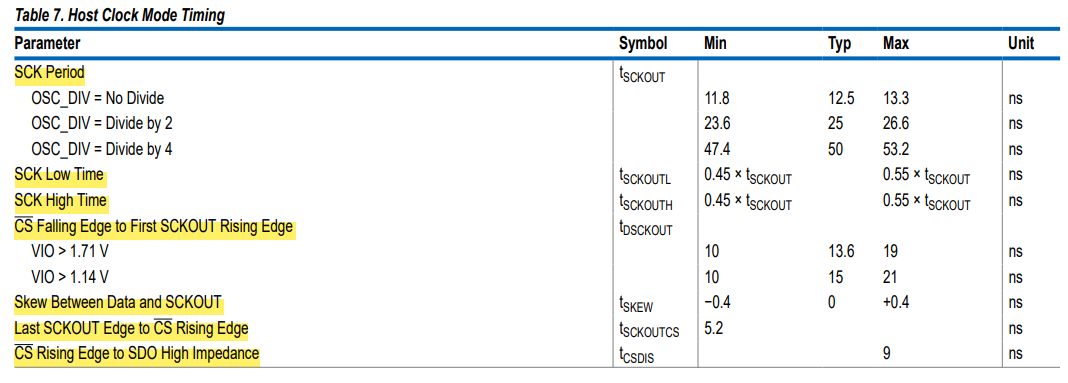


همچنین میتوان به صورت stream با رجیستر ها ارتباط برقرار کرد:



**الزامات رعایت timing ها در طراحی**

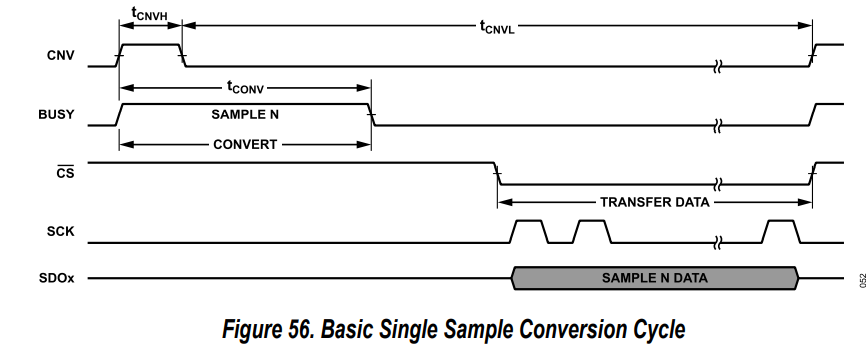
****

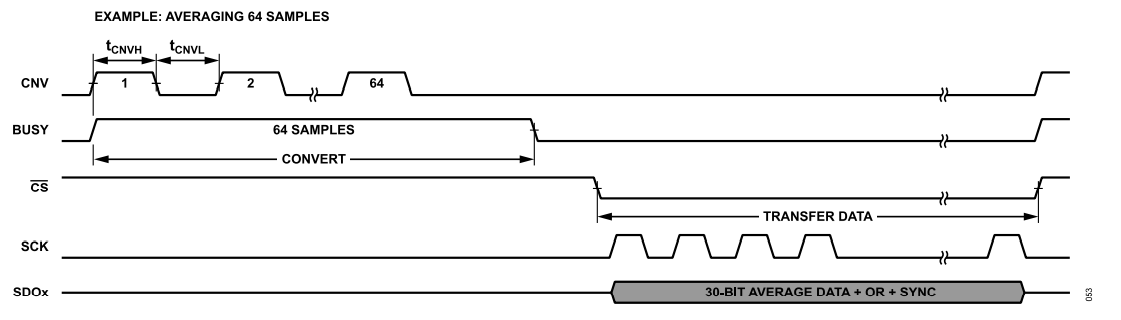
****

**مود های کاری :**

این آیسی دو نوع مود کاری دارد :

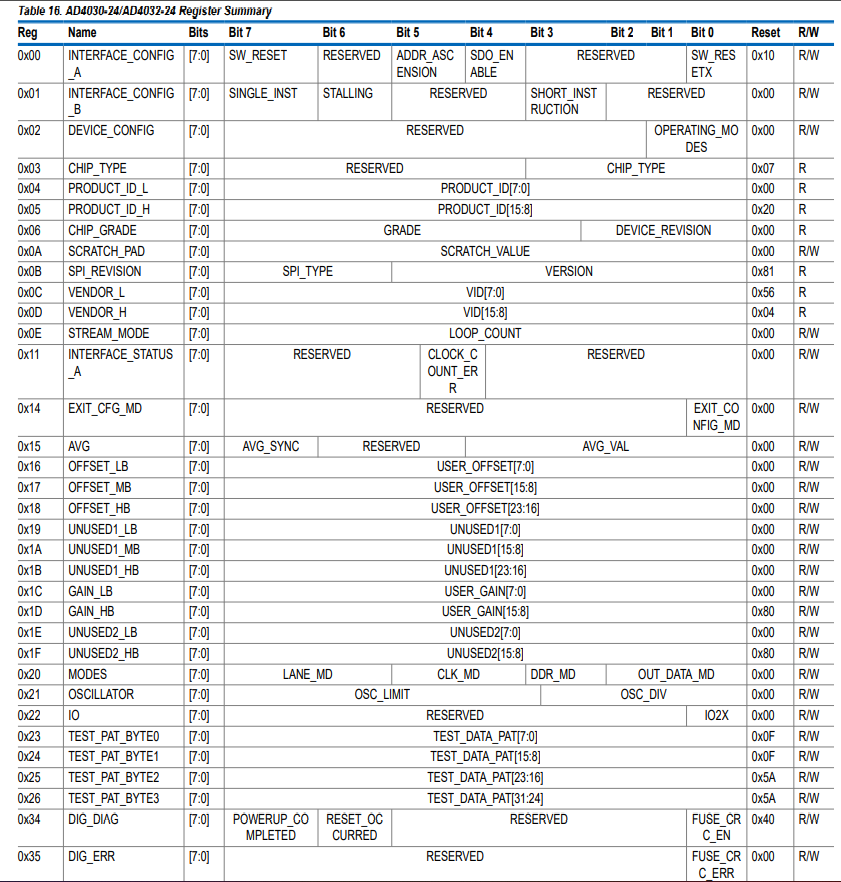
* Basic single sample
* Averaging mode



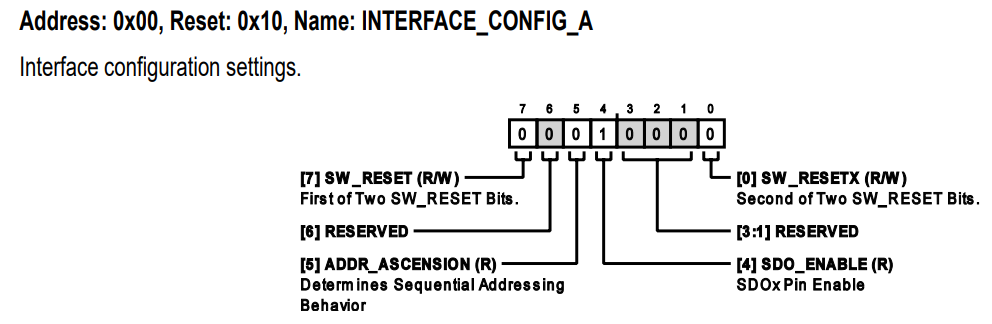


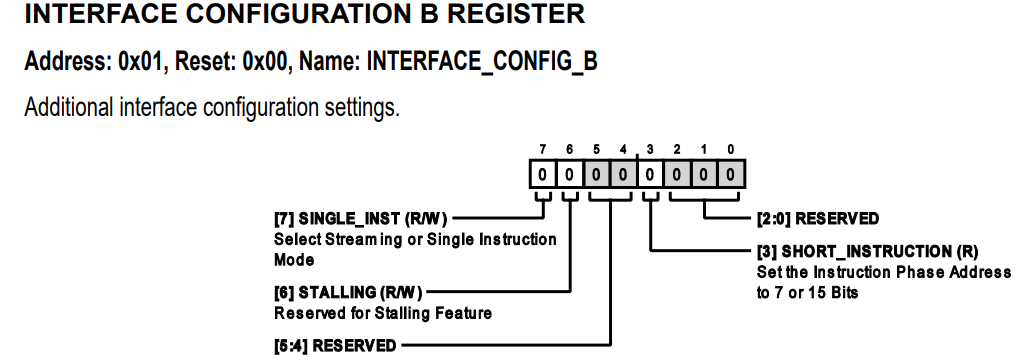
جهت اجتناب از پیچیدگی های سخت افزاری و جهت درک ساده تر روند پروژه ، سخت افزار طراحی شده برای راه اندازی این آیسی در مود Basic Single Sample صورت خواهد گرفت

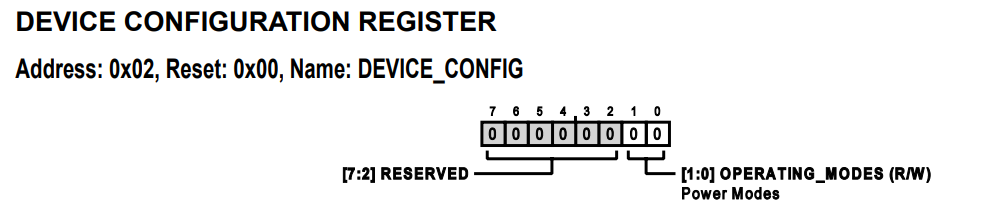
**خلاصه ای از کل رجیستر ها :**

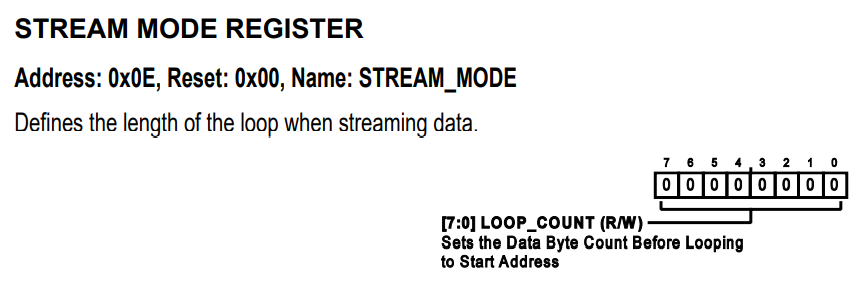


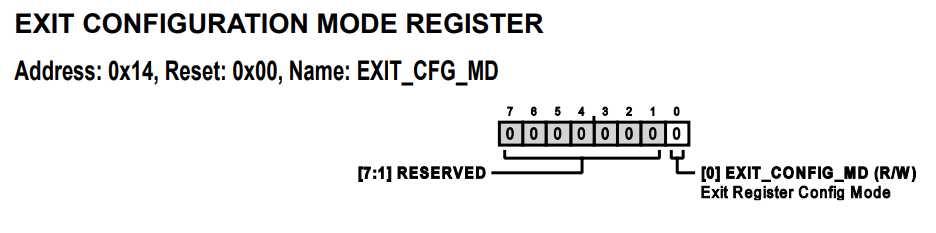
**رجیستر های مورد نیاز برای کنترل اولیه ی ADC :**

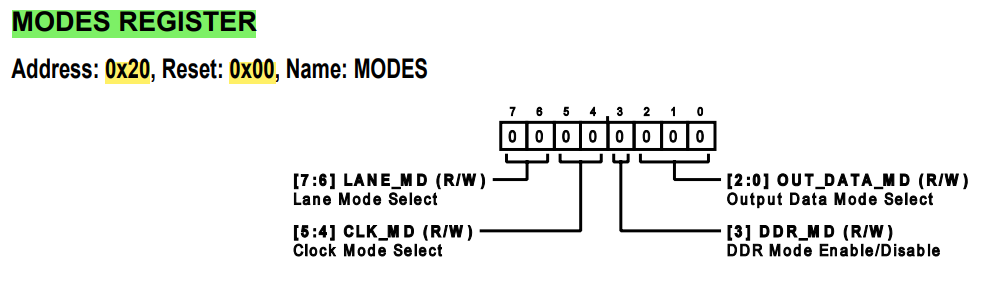
****

****

****

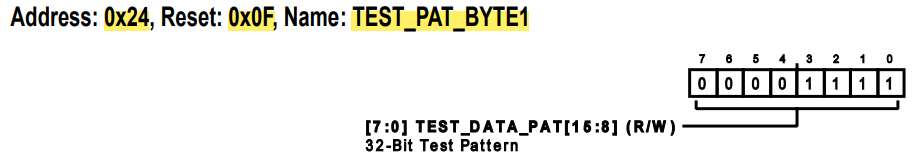
****

****

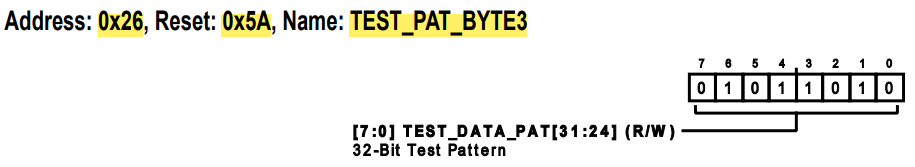
****

****

****

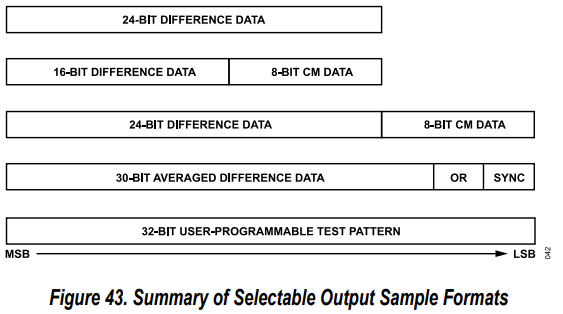
****

****

****

**فرمت DATA**

دیتای جمع آوری شده توسط مبدل به صورت زیر است که فرمت چینش آن توسط رجیستر MODE\_REGISTER قابل تنظیم است.



**ارائه بلوک پیشنهادی سخت افزار برای کنترل ADC مورد نظر:**

**SPI**

**MASTER**

8

8

reset

enable

busy

clock

rx\_data

tx\_data

BUSY

CNV

RST

CS

SCK

MISO

MOSI

**ADC**

**AD4030**

**AD4030\_FSM**

**(State Machines)**

24

/

CLOCK

DATA

START

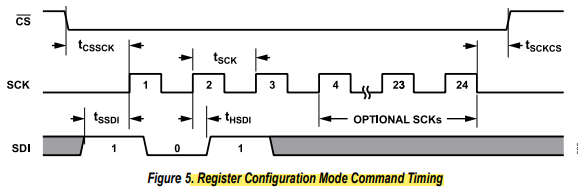
RESET

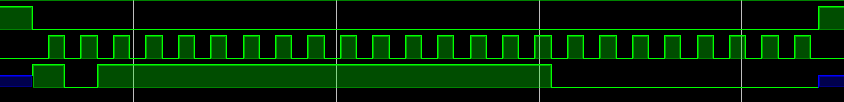
DRDY

طرح پیشنهادی برای کنترل این آیسی به این صورت است که ابتدا باید سخت افزار SPI را جهت ارتباط با ADC طراحی کرد ، سپس بلوکی را جهت فرمان دادن حالت های مورد نیاز جهت کنترل این آیسی طرح و پیاده سازی کنیم.

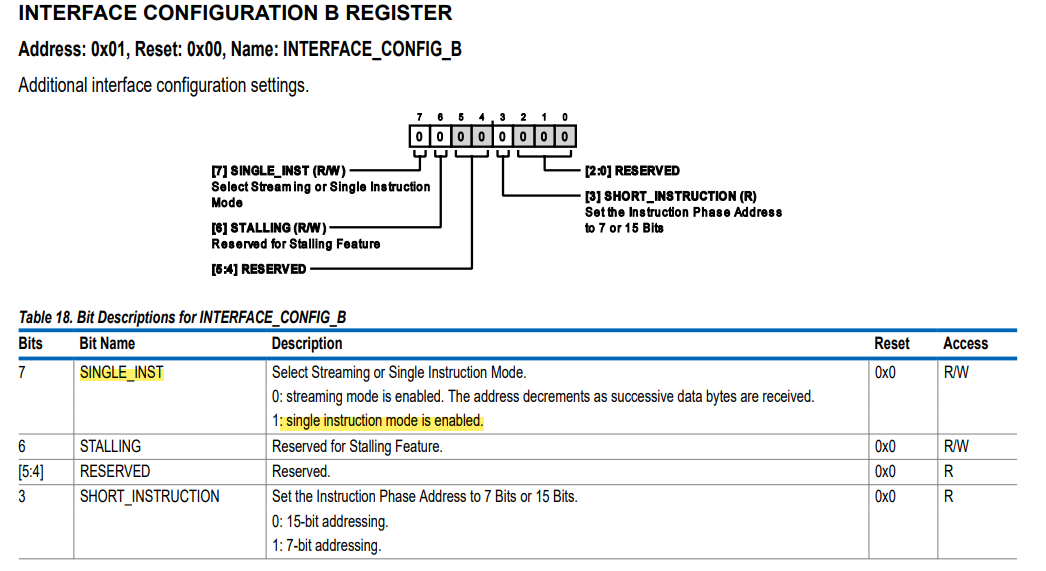
روند کار به این صورت است که:

1. هنگامی که ADC روشن میشود به صورت پیشفزض در مود conversion است . برای اینکه بتوان به مود تنظیمات این آیسی رفت باید ابتدا دستور 0xBFFF00 را توسط spi به آیسی ارسال کنیم.



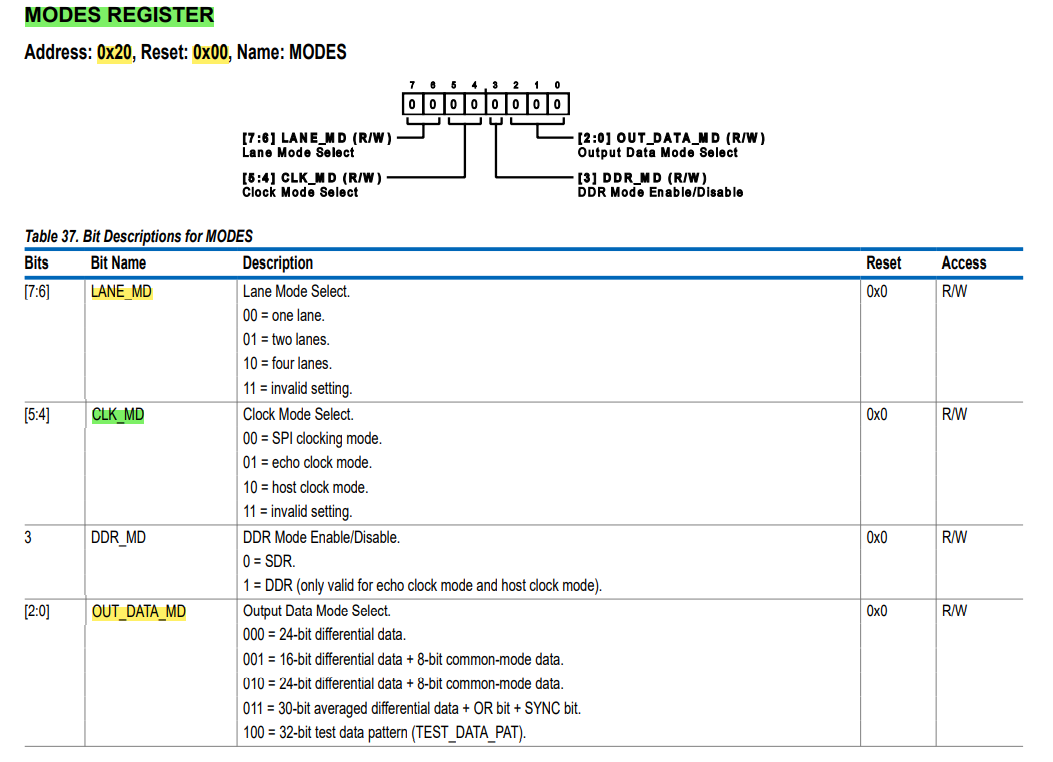
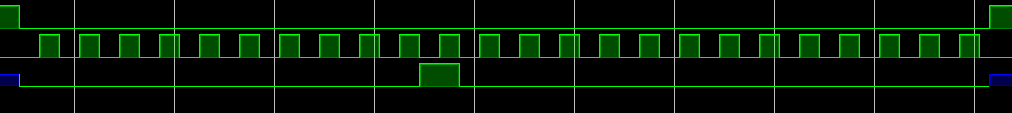


1. سپس جهت تنظیم single instruction mode ، مقدار 0x80 را در رجیستر INTERFACR CONGIGURATION B که در آدرس 0x01 قرار دارد میریزیم



A picture containing diagram

Description automatically generated

1. سپس جهت تنظیم فومت دیتای خروجی باید رجیستر MODES را تنظیم کنیم . برای تنظیم خروجی سریال یک لاین و مود SPI clockingو همچنین فرمت خروجی 24 بیت ، مقدار 0x00 را در این رجیستر قرار می دهیم.
2. سپس برای خروج از مود تنظیمات باید مقدار 0x01 را در رجیستر ٍEXIT CONFIGURATION MODE قرار دهیم.

Graphical user interface

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

1. Diagram

   Description automatically generatedحال میتوان با پالس دادن به پایه ی CNV آیسی ، شروع به تبدیل کردن و خواندن از ADC کرد . به صورت زیر:

**طراحی بلوک SPI\_MASTER**

**SPI\_BLOCK\_FINAL**

enable

reset

busy

clock

rx\_data

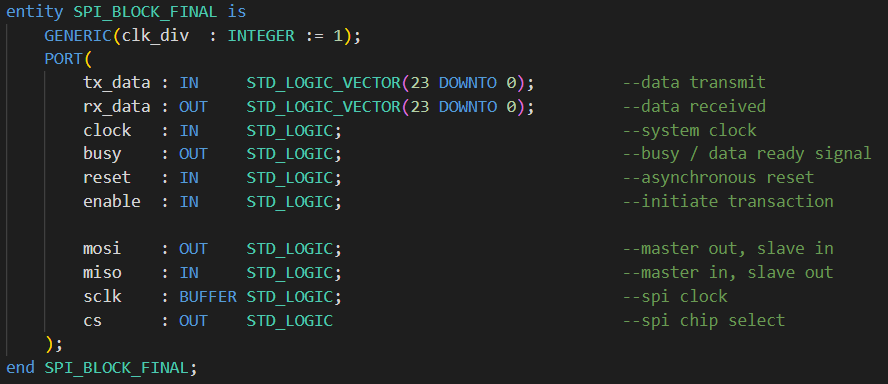
tx\_data

sclk

cs

miso

mosi



library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity SPI\_BLOCK\_FINAL is

GENERIC(clk\_div : INTEGER := 1);

PORT(

tx\_data : IN STD\_LOGIC\_VECTOR(23 DOWNTO 0); --data transmit

rx\_data : OUT STD\_LOGIC\_VECTOR(23 DOWNTO 0); --data received

clock : IN STD\_LOGIC; --system clock

busy : OUT STD\_LOGIC; --busy / data ready signal

reset : IN STD\_LOGIC; --asynchronous reset

enable : IN STD\_LOGIC; --initiate transaction

mosi : OUT STD\_LOGIC; --master out, slave in

miso : IN STD\_LOGIC; --master in, slave out

sclk : BUFFER STD\_LOGIC; --spi clock

cs : OUT STD\_LOGIC --spi chip select

);

end SPI\_BLOCK\_FINAL;

architecture Behavioral of SPI\_BLOCK\_FINAL is

TYPE Tstate IS (idle, delay, clock0, clock1, compelete);

SIGNAL state : Tstate;

SIGNAL S\_rx\_data : STD\_LOGIC\_VECTOR(23 DOWNTO 0) := (others=>'0');

SIGNAL S\_tx\_data : STD\_LOGIC\_VECTOR(23 DOWNTO 0) := (others=>'0');

SIGNAL S\_miso : STD\_LOGIC;

SIGNAL ClockCounter : integer := 0;

SIGNAL index : integer range 0 to 23 := 0;

begin

process(clock, reset, enable)

begin

if(reset = '0') then

state <= idle;

S\_rx\_data <= (others=>'0');

S\_tx\_data <= (others=>'0');

ClockCounter <= 0;

sclk <= '0';

index <= 0;

busy <= '0';

mosi <= 'Z';

cs <= '1';

elsif(clock'EVENT and clock = '1') then

case state is

when idle =>

if(enable = '1') then

S\_tx\_data <= tx\_data;

S\_rx\_data <= (others=>'0');

state <= delay;

sclk <= '0';

busy <= '1';

ClockCounter <= 0;

index <= 23;

end if;

when delay =>

cs <= '0';

mosi <= S\_tx\_data(index);

index <= index - 1;

state <= clock0;

when clock0 =>

ClockCounter <= ClockCounter + 1;

if(ClockCounter = clk\_div) then

sclk <= '1';

ClockCounter <= 0;

state <= clock1;

end if;

when clock1 =>

ClockCounter <= ClockCounter + 1;

if(ClockCounter = clk\_div) then

sclk <= '0';

state <= clock0;

ClockCounter <= 0;

index <= index - 1;

if(index >= 0) then

mosi <= S\_tx\_data(index);

else

state <= compelete;

end if;

S\_rx\_data(index+1) <= S\_miso;

end if;

when compelete =>

state <= idle;

ClockCounter <= 0;

sclk <= '0';

index <= 0;

busy <= '0';

mosi <= 'Z';

rx\_data <= S\_rx\_data;

cs <= '1';

end case;

end if;

end process;

S\_miso <= miso;

end Behavioral;

**طراحی بلوک AD4030\_FSM**

**AD4030\_FSM**

RESET

START

CLOCK

DATA

BUSY

CNV

RST

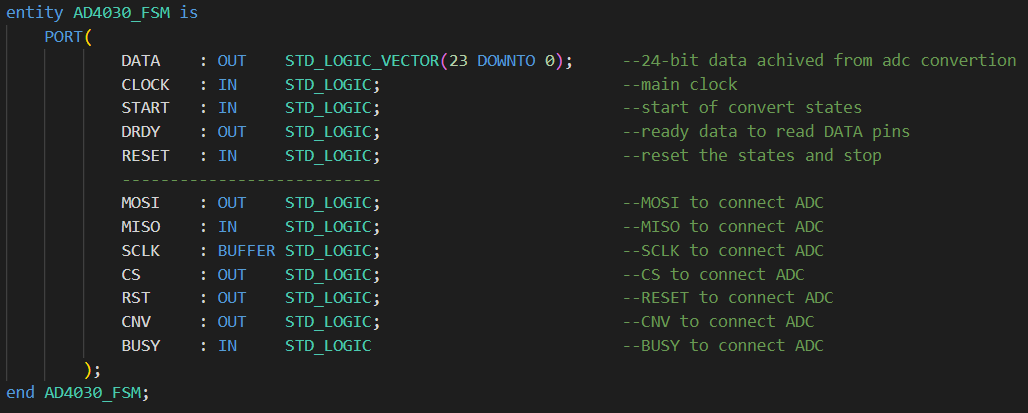
DRDY

SCLK

CS

MISO

MOSI



library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity AD4030\_FSM is

PORT(

DATA : OUT STD\_LOGIC\_VECTOR(23 DOWNTO 0); --24-bit data achived from adc convertion

CLOCK : IN STD\_LOGIC; --main clock

START : IN STD\_LOGIC; --start of convert states

DRDY : OUT STD\_LOGIC; --ready data to read DATA pins

RESET : IN STD\_LOGIC; --reset the states and stop

---------------------------

MOSI : OUT STD\_LOGIC; --MOSI to connect ADC

MISO : IN STD\_LOGIC; --MISO to connect ADC

SCLK : BUFFER STD\_LOGIC; --SCLK to connect ADC

CS : OUT STD\_LOGIC; --CS to connect ADC

RST : OUT STD\_LOGIC; --RESET to connect ADC

CNV : OUT STD\_LOGIC; --CNV to connect ADC

BUSY : IN STD\_LOGIC --BUSY to connect ADC

);

end AD4030\_FSM;

architecture Behavioral of AD4030\_FSM is

COMPONENT SPI\_BLOCK\_FINAL

generic(clk\_div : INTEGER := 1);

PORT(

tx\_data : IN STD\_LOGIC\_VECTOR(23 DOWNTO 0); --data transmit

rx\_data : OUT STD\_LOGIC\_VECTOR(23 DOWNTO 0); --data received

clock : IN STD\_LOGIC; --system clock

busy : OUT STD\_LOGIC; --busy / data ready signal

reset : IN STD\_LOGIC; --asynchronous reset

enable : IN STD\_LOGIC; --initiate transaction

---------------------------

mosi : OUT STD\_LOGIC; --master out, slave in

miso : IN STD\_LOGIC; --master in, slave out

sclk : BUFFER STD\_LOGIC; --spi clock

cs : OUT STD\_LOGIC --spi chip select

);

END COMPONENT;

signal spi\_reset : std\_logic := '1';

signal spi\_enable : std\_logic := '0';

signal spi\_tx\_data : std\_logic\_vector(23 downto 0) := X"000000";

signal spi\_rx\_data : std\_logic\_vector(23 downto 0) := X"000000";

signal spi\_busy : std\_logic := '0';

------------------------------------------------------------------------------

signal s\_rst : std\_logic := '1';

signal s\_cnv : std\_logic := '0';

signal s\_busy : std\_logic := '0';

------------------------------------------------------------------------------

TYPE Tstate IS (

IDLE,

REG\_INIT\_CMD, REG\_SEND\_CMD, CHK\_SPIBUSY\_CMD,

REG\_INIT\_CFG\_B, REG\_SEND\_CFG\_B, CHK\_SPIBUSY\_CFG,

REG\_INIT\_MODES\_SINGLE, REG\_SEND\_MODES\_SINGLE, CHK\_SPIBUSY\_MODES,

REG\_INIT\_EXITCMD, REG\_SEND\_EXITCMD, CHK\_SPIBUSY\_EXITCMD,

START\_CONVERSION, WAIT\_FOR\_BUSY, CHK\_SPIBUSY\_READ\_DATA, READ\_24BIT\_DATA

);

SIGNAL state : Tstate;

signal s\_start : std\_logic := '0';

signal s\_reset : std\_logic := '0';

signal s\_temp\_data : std\_logic\_vector(23 downto 0) := X"000000";

signal CycleCount : INTEGER := 0;

begin

SPIBLOCK: SPI\_BLOCK\_FINAL

generic map (1)

port map (

tx\_data => spi\_tx\_data,

rx\_data => spi\_rx\_data,

clock => CLOCK,

busy => spi\_busy,

reset => RESET,

enable => spi\_enable,

-------------------------------

miso => MISO,

mosi => MOSI,

sclk => SCLK,

cs => CS

);

process(CLOCK, s\_reset, s\_busy)

begin

if( s\_reset = '0') then

state <= IDLE;

s\_rst <= '1';

s\_cnv <= '0';

------------------

--spi\_reset <= '1';

------------------

DATA <= X"000000";

DRDY <= '0';

CycleCount <= 0;

elsif(clock'EVENT and clock = '1') then

case state is

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-- Initiate States

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when IDLE =>

if(s\_start = '1') then

state <= REG\_INIT\_CMD;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when REG\_INIT\_CMD =>

spi\_tx\_data <= x"BFFF00";

state <= REG\_SEND\_CMD;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when REG\_SEND\_CMD =>

spi\_enable <= '1';

state <= CHK\_SPIBUSY\_CMD;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when CHK\_SPIBUSY\_CMD =>

if(spi\_busy = '1') then

state <= REG\_INIT\_CFG\_B;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when REG\_INIT\_CFG\_B =>

spi\_enable <= '0';

spi\_tx\_data <= x"000180";

if(spi\_busy = '0') then

state <= REG\_SEND\_CFG\_B;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when REG\_SEND\_CFG\_B =>

spi\_enable <= '1';

state <= CHK\_SPIBUSY\_CFG;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when CHK\_SPIBUSY\_CFG =>

if(spi\_busy = '1') then

state <= REG\_INIT\_MODES\_SINGLE;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when REG\_INIT\_MODES\_SINGLE =>

spi\_enable <= '0';

spi\_tx\_data <= x"002000";

if(spi\_busy = '0') then

state <= REG\_SEND\_MODES\_SINGLE;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when REG\_SEND\_MODES\_SINGLE =>

spi\_enable <= '1';

state <= CHK\_SPIBUSY\_MODES;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when CHK\_SPIBUSY\_MODES =>

if(spi\_busy = '1') then

state <= REG\_INIT\_EXITCMD;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when REG\_INIT\_EXITCMD =>

spi\_enable <= '0';

spi\_tx\_data <= x"001401";

if(spi\_busy = '0') then

state <= REG\_SEND\_EXITCMD;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when REG\_SEND\_EXITCMD =>

spi\_enable <= '1';

state <= CHK\_SPIBUSY\_EXITCMD;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when CHK\_SPIBUSY\_EXITCMD =>

if(spi\_busy = '1') then

state <= START\_CONVERSION;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-- Conversion AND Read DATA

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when START\_CONVERSION =>

spi\_tx\_data <= x"000000";

spi\_enable <= '0';

if(spi\_busy = '0') then

s\_cnv <= '1';

if(s\_busy = '1') then

state <= WAIT\_FOR\_BUSY;

end if;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when WAIT\_FOR\_BUSY =>

s\_cnv <= '0';

if(s\_busy = '0') then

spi\_enable <= '1';

if(spi\_enable = '1') then

state <= CHK\_SPIBUSY\_READ\_DATA;

end if;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when CHK\_SPIBUSY\_READ\_DATA =>

spi\_enable <= '0';

if(spi\_busy = '0') then

DRDY <= '0';

state <= READ\_24BIT\_DATA;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

when READ\_24BIT\_DATA =>

if(s\_busy = '0') then

DATA <= spi\_rx\_data;

DRDY <= '1';

state <= START\_CONVERSION;

end if;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

end case;

end if;

end process;

spi\_reset <= RESET;

RST <= s\_rst;

CNV <= s\_cnv;

s\_busy <= BUSY;

-----------------

s\_start <= START;

s\_reset <= RESET;

end Behavioral;

**مدل سازی AD4030 در تست بنچ جهت تست عملکرد صحیح بلاک ها**

PROCEDURE AD4030\_BEHAVIOR (

SIGNAL MISO : OUT STD\_LOGIC;

SIGNAL MOSI : IN STD\_LOGIC;

SIGNAL SCLK : IN STD\_LOGIC;

SIGNAL CS : IN STD\_LOGIC;

SIGNAL RST : IN STD\_LOGIC;

SIGNAL CNV : IN STD\_LOGIC;

SIGNAL BUSY : OUT STD\_LOGIC

) IS

VARIABLE SPI\_RX : STD\_LOGIC\_VECTOR(23 DOWNTO 0) := (others => '0');

VARIABLE SPI\_TX : STD\_LOGIC\_VECTOR(23 DOWNTO 0) := X"000000";

-----------------------------------------------------------

PROCEDURE SPI\_SLAVE\_READ IS

BEGIN

WAIT UNTIL (falling\_edge(CS));

FOR i IN 23 DOWNTO 0 LOOP

WAIT UNTIL (rising\_edge(SCLK));

SPI\_RX(i) := MOSI;

END LOOP;

END SPI\_SLAVE\_READ;

-----------------------------------------------------------

PROCEDURE SPI\_SLAVE\_WRITE IS

BEGIN

WAIT UNTIL (falling\_edge(CS));

FOR i IN 23 DOWNTO 0 LOOP

MISO <= SPI\_TX(i);

WAIT UNTIL (falling\_edge(SCLK));

END LOOP;

END SPI\_SLAVE\_WRITE;

-----------------------------------------------------------

BEGIN

SPI\_SLAVE\_READ;

IF SPI\_RX=X"BFFF00" THEN

SPI\_SLAVE\_READ;

IF SPI\_RX=X"000180" THEN

SPI\_SLAVE\_READ;

IF SPI\_RX=X"002000" THEN

SPI\_SLAVE\_READ;

IF SPI\_RX=X"001401" THEN

WHILE(TRUE) LOOP

WAIT UNTIL (rising\_edge(CNV));

BUSY <= '1';

WAIT FOR 300NS;

BUSY <= '0';

SPI\_SLAVE\_WRITE;

SPI\_TX := std\_logic\_vector(unsigned(SPI\_TX)+1);

END LOOP;

END IF;

END IF;

END IF;

END IF;

END AD4030\_BEHAVIOR;

**کد تست بنچ**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

use std.textio.all;

use std.env.finish;

entity AD4030\_FSM\_tb is

end AD4030\_FSM\_tb;

architecture Behavioral of AD4030\_FSM\_tb is

constant clk\_hz : integer := 50e6;

constant clk\_period : time := 1 sec / clk\_hz;

COMPONENT AD4030\_FSM

PORT(

DATA : OUT STD\_LOGIC\_VECTOR(23 DOWNTO 0);

CLOCK : IN STD\_LOGIC;

START : IN STD\_LOGIC;

DRDY : OUT STD\_LOGIC;

RESET : IN STD\_LOGIC;

---------------------------

MOSI : OUT STD\_LOGIC;

MISO : IN STD\_LOGIC;

SCLK : BUFFER STD\_LOGIC;

CS : OUT STD\_LOGIC;

RST : OUT STD\_LOGIC;

CNV : OUT STD\_LOGIC;

BUSY : IN STD\_LOGIC

);

END COMPONENT;

signal tb\_DATA : std\_logic\_vector(23 downto 0) := X"000000";

signal tb\_CLOCK : std\_logic := '0';

signal tb\_START : std\_logic := '0';

signal tb\_DRDY : std\_logic := '0';

signal tb\_RESET : std\_logic := '1';

signal tb\_MOSI : std\_logic := '0';

signal tb\_MISO : std\_logic := '0';

signal tb\_SCLK : std\_logic := '0';

signal tb\_CS : std\_logic := '0';

signal tb\_RST : std\_logic := '0';

signal tb\_CNV : std\_logic := '0';

signal tb\_BUSY : std\_logic := '0';

--===================================================================================

PROCEDURE AD4030\_BEHAVIOR (

SIGNAL MISO : OUT STD\_LOGIC;

SIGNAL MOSI : IN STD\_LOGIC;

SIGNAL SCLK : IN STD\_LOGIC;

SIGNAL CS : IN STD\_LOGIC;

SIGNAL RST : IN STD\_LOGIC;

SIGNAL CNV : IN STD\_LOGIC;

SIGNAL BUSY : OUT STD\_LOGIC

) IS

VARIABLE SPI\_RX : STD\_LOGIC\_VECTOR(23 DOWNTO 0) := (others => '0');

VARIABLE SPI\_TX : STD\_LOGIC\_VECTOR(23 DOWNTO 0) := X"000000";

-----------------------------------------------------------

PROCEDURE SPI\_SLAVE\_READ IS

BEGIN

WAIT UNTIL (falling\_edge(CS));

FOR i IN 23 DOWNTO 0 LOOP

WAIT UNTIL (rising\_edge(SCLK));

SPI\_RX(i) := MOSI;

END LOOP;

END SPI\_SLAVE\_READ;

-----------------------------------------------------------

PROCEDURE SPI\_SLAVE\_WRITE IS

BEGIN

WAIT UNTIL (falling\_edge(CS));

FOR i IN 23 DOWNTO 0 LOOP

MISO <= SPI\_TX(i);

WAIT UNTIL (falling\_edge(SCLK));

END LOOP;

END SPI\_SLAVE\_WRITE;

-----------------------------------------------------------

BEGIN

SPI\_SLAVE\_READ;

IF SPI\_RX=X"BFFF00" THEN

SPI\_SLAVE\_READ;

IF SPI\_RX=X"000180" THEN

SPI\_SLAVE\_READ;

IF SPI\_RX=X"002000" THEN

SPI\_SLAVE\_READ;

IF SPI\_RX=X"001401" THEN

WHILE(TRUE) LOOP

WAIT UNTIL (rising\_edge(CNV));

BUSY <= '1';

WAIT FOR 300NS;

BUSY <= '0';

SPI\_SLAVE\_WRITE;

SPI\_TX := std\_logic\_vector(unsigned(SPI\_TX)+1);

END LOOP;

END IF;

END IF;

END IF;

END IF;

END AD4030\_BEHAVIOR;

--===================================================================================

begin

tb\_CLOCK <= not tb\_CLOCK after clk\_period / 2;

UUT: AD4030\_FSM

port map (

DATA => tb\_DATA ,

CLOCK => tb\_CLOCK,

START => tb\_START,

DRDY => tb\_DRDY ,

RESET => tb\_RESET,

MOSI => tb\_MOSI ,

MISO => tb\_MISO ,

SCLK => tb\_SCLK ,

CS => tb\_CS ,

RST => tb\_RST ,

CNV => tb\_CNV ,

BUSY => tb\_BUSY

);

HANDLE\_BLOCK\_PROC : process

begin

wait until (falling\_edge(tb\_CLOCK));

wait until (falling\_edge(tb\_CLOCK));

wait until (falling\_edge(tb\_CLOCK));

tb\_RESET <= '0';

wait until (falling\_edge(tb\_CLOCK));

tb\_RESET <= '1';

wait until (falling\_edge(tb\_CLOCK));

wait until (falling\_edge(tb\_CLOCK));

wait until (falling\_edge(tb\_CLOCK));

wait until (falling\_edge(tb\_CLOCK));

tb\_START <= '1';

wait;

end process;

AD4030\_MODEL\_PROC :process

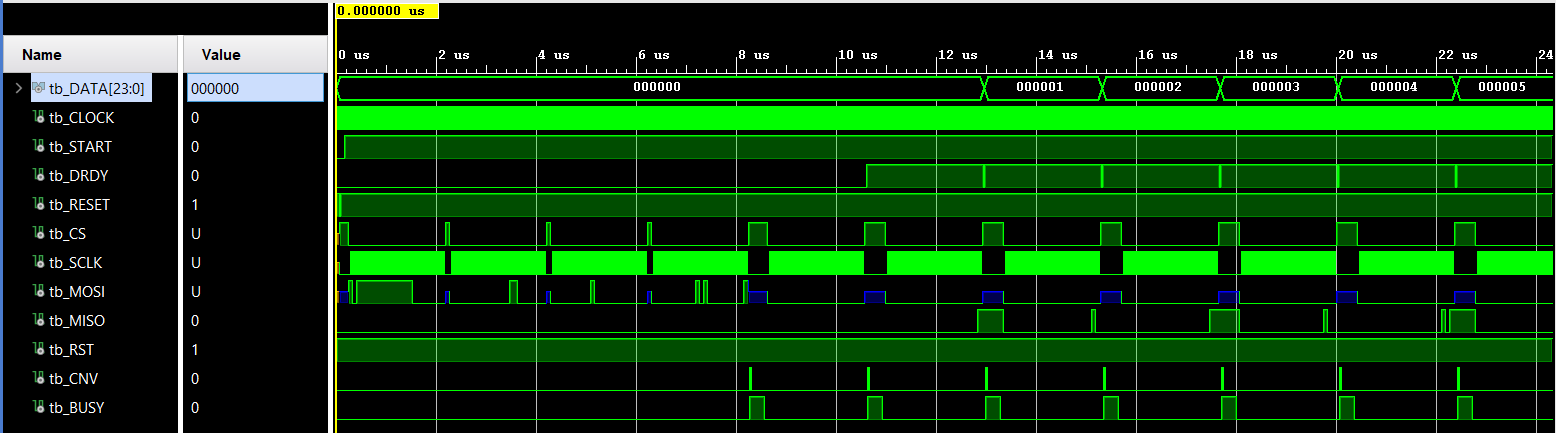
begin

AD4030\_BEHAVIOR(tb\_MISO, tb\_MOSI, tb\_SCLK, tb\_CS, tb\_RST, tb\_CNV, tb\_BUSY);

end process;

end Behavioral;

**نتایج شبیه سازی**

****

