به نام خدا

فایل گزارش پروژه

درس VHDL

Designing Hardware Block to handle AD4030 ADC with VHDL

رامین اصیلی

401611022
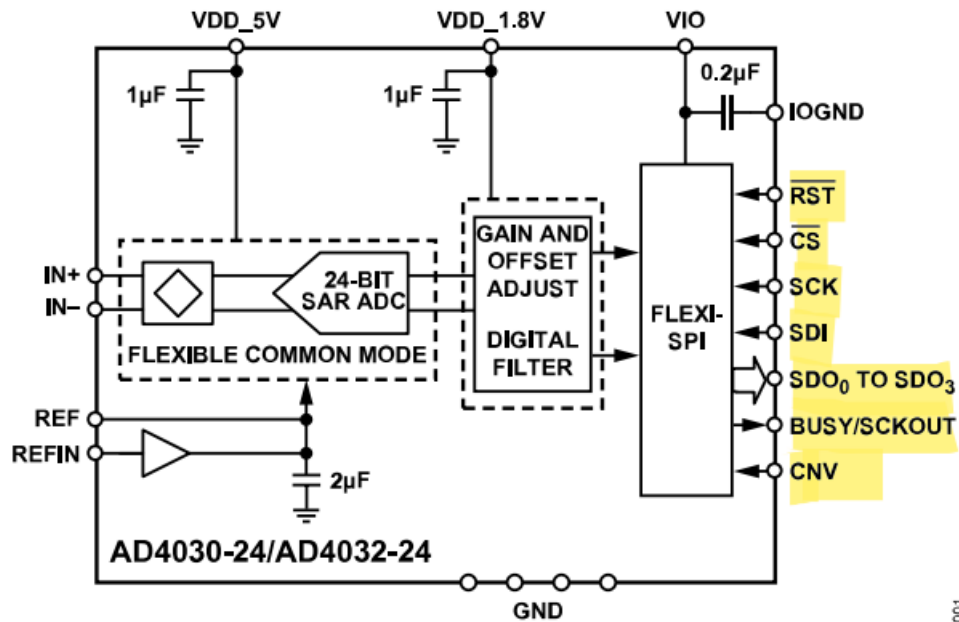
شمای المان های داخل ADC :

Figure 1. Functional Block Diagram

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

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.

Table 13. BUSY_SCKOUT Pin Behavior vs. Clocking Mode

Clocking Mode	Behavior
SPI Clocking Mode	Valid BUSY_SCKOUT pin signal for the ADC conversion status. The busy signal on the BUSY_SCKOUT pin goes high when a conversion is triggered by the CNV signal. The busy signal on the BUSY_SCKOUT pin goes low when the conversion is complete.
Echo Clock Mode	Bit clock. The BUSY_SCKOUT pin is a delayed version of SCK input.
Host Clock Mode	Bit clock. The BUSY_SCKOUT pin sources the clock signal from the internal oscillator.

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:
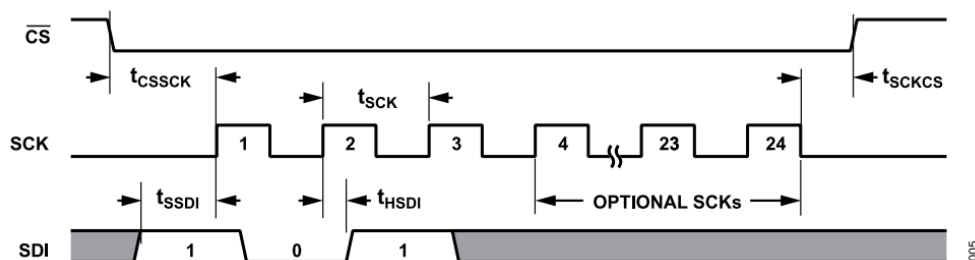
Figure 5. Register Configuration Mode Command Timing

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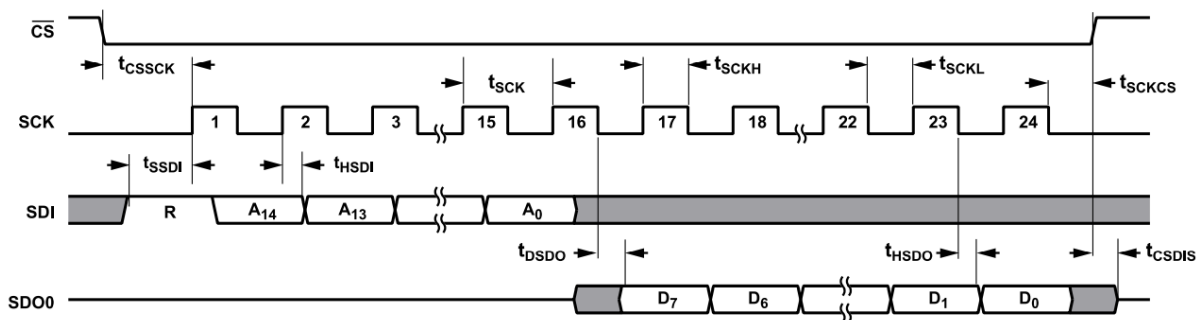

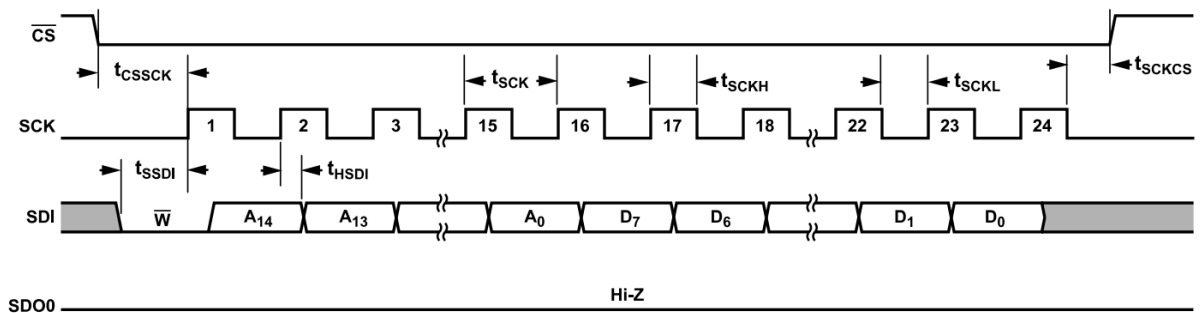Figure 4. Register Configuration Mode Read Timing

Figure 3. Register Configuration Mode Write Timing

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:

1. Perform a read back from a dummy register address 0x3FFF, to enter the register configuration mode.
2. Read back from or write to the desired user register addresses.
3. Exit the register configuration mode by writing 0x01 to register address 0x0014. Exiting register configuration mode causes the register updates to take effect.

4

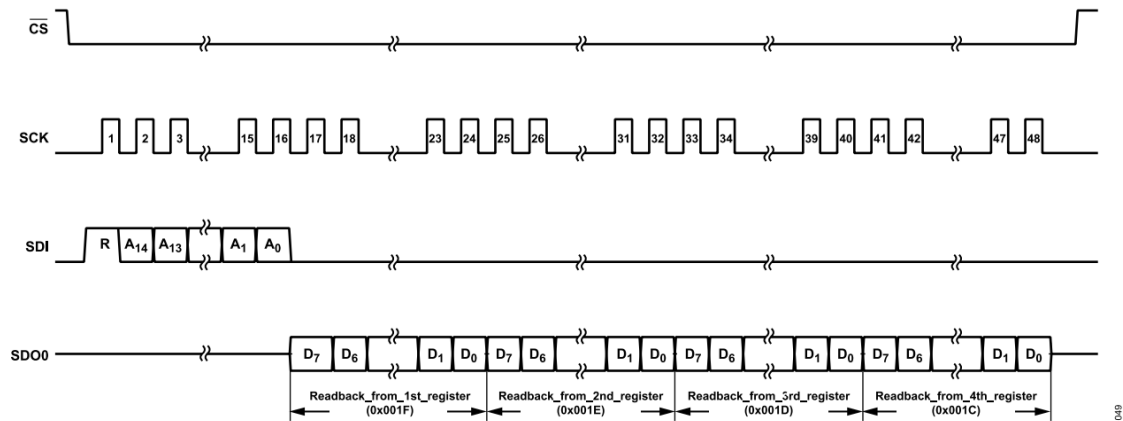همچنین میتوان به صورت stream با رجیستر ها ارتباط برقرار کرد:

Figure 51. Stream Mode Bulk Register Read Back Operation
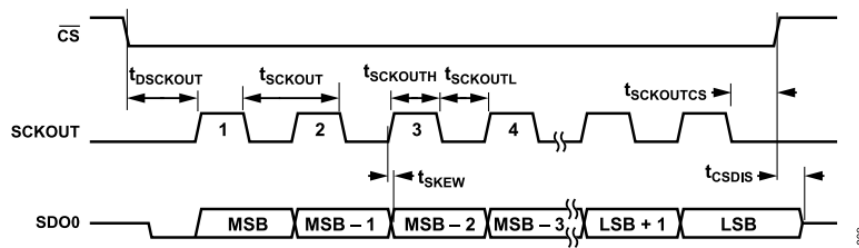
الزامات رعایت timing ها در طراحی

Figure 9. Host Clock Mode Timing, SDR, 1-Lane
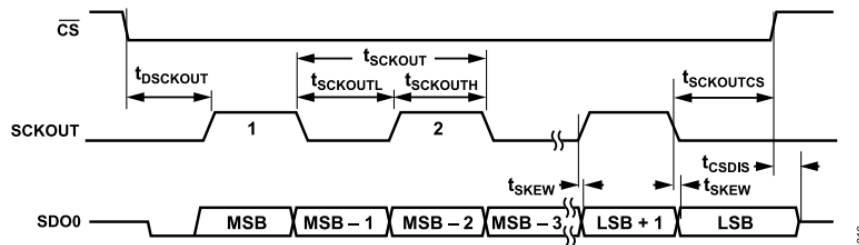
Figure 10. Host Clock Mode Timing, DDR, 1-Lane

Table 7. Host Clock Mode Timing

Parameter	Symbol	Min	Typ	Max	Unit
SCK Period	t_{SCKOUT}				
OSC_DIV = No Divide		11.8	12.5	13.3	ns
OSC_DIV = Divide by 2		23.6	25	26.6	ns
OSC_DIV = Divide by 4		47.4	50	53.2	ns
SCK Low Time	$t_{SCKOUTL}$	$0.45 \times t_{SCKOUT}$		$0.55 \times t_{SCKOUT}$	ns
SCK High Time	$t_{SCKOUTH}$	$0.45 \times t_{SCKOUT}$		$0.55 \times t_{SCKOUT}$	ns
$\overline{CS}$ Falling Edge to First SCKOUT Rising Edge	$t_{DSCKOUT}$				
VIO > 1.71 V		10	13.6	19	ns
VIO > 1.14 V		10	15	21	ns
Skew Between Data and SCKOUT	t_{SKEW}	−0.4	0	+0.4	ns
Last SCKOUT Edge to $\overline{CS}$ Rising Edge	$t_{SCKOUTCS}$	5.2			ns
$\overline{CS}$ Rising Edge to SDO High Impedance	t_{CSDIS}			9	ns

این آیسی دو نوع مود کاری دارد :

- Basic single sample
- Averaging mode

Figure 56. Basic Single Sample Conversion Cycle

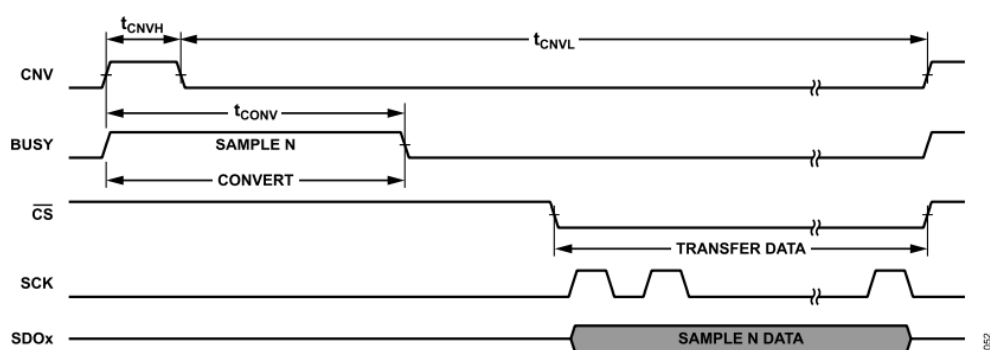جهت اجتناب از پیچیدگی های سخت افزاری و جهت درک ساده تر روند پروژه ، سخت افزار طراحی شده برای راه اندازی این آیسی در مود Basic Single Sample صورت خواهد گرفت

Table 16. AD4030-24/AD4032-24 Register Summary
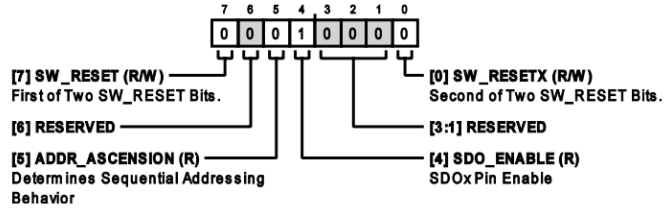
Reg	Name	Bits	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset	R/W
0x00	INTERFACE_CONFIG_A	[7:0]	SW_RESET	RESERVED	ADDR_ASCENSION	SDO_ENABLE	RESERVED			SW_RESETX	0x10	R/W
0x01	INTERFACE_CONFIG_B	[7:0]	SINGLE_INST	STALLING	RESERVED		SHORT_INSTRUCTION	RESERVED			0x00	R/W
0x02	DEVICE_CONFIG	[7:0]	RESERVED						OPERATING_MODES		0x00	R/W
0x03	CHIP_TYPE	[7:0]	RESERVED				CHIP_TYPE				0x07	R
0x04	PRODUCT_ID_L	[7:0]	PRODUCT_ID[7:0]								0x00	R
0x05	PRODUCT_ID_H	[7:0]	PRODUCT_ID[15:8]								0x20	R
0x06	CHIP_GRADE	[7:0]	GRADE				DEVICE_REVISION				0x00	R
0x0A	SCRATCH_PAD	[7:0]	SCRATCH_VALUE								0x00	R/W
0x0B	SPI_REVISION	[7:0]	SPI_TYPE			VERSION					0x81	R
0x0C	VENDOR_L	[7:0]	VID[7:0]								0x56	R
0x0D	VENDOR_H	[7:0]	VID[15:8]								0x04	R
0x0E	STREAM_MODE	[7:0]	LOOP_COUNT								0x00	R/W
0x11	INTERFACE_STATUS_A	[7:0]	RESERVED			CLOCK_COUNT_ERR	RESERVED				0x00	R/W
0x14	EXIT_CFG_MD	[7:0]	RESERVED							EXIT_CONFIG_MD	0x00	R/W
0x15	AVG	[7:0]	AVG_SYNC	RESERVED		AVG_VAL					0x00	R/W
0x16	OFFSET_LB	[7:0]	USER_OFFSET[7:0]								0x00	R/W
0x17	OFFSET_MB	[7:0]	USER_OFFSET[15:8]								0x00	R/W
0x18	OFFSET_HB	[7:0]	USER_OFFSET[23:16]								0x00	R/W
0x19	UNUSED1_LB	[7:0]	UNUSED1[7:0]								0x00	R/W
0x1A	UNUSED1_MB	[7:0]	UNUSED1[15:8]								0x00	R/W
0x1B	UNUSED1_HB	[7:0]	UNUSED1[23:16]								0x00	R/W
0x1C	GAIN_LB	[7:0]	USER_GAIN[7:0]								0x00	R/W
0x1D	GAIN_HB	[7:0]	USER_GAIN[15:8]								0x80	R/W
0x1E	UNUSED2_LB	[7:0]	UNUSED2[7:0]								0x00	R/W
0x1F	UNUSED2_HB	[7:0]	UNUSED2[15:8]								0x80	R/W
0x20	MODES	[7:0]	LANE_MD		CLK_MD		DDR_MD	OUT_DATA_MD			0x00	R/W
0x21	OSCILLATOR	[7:0]	OSC_LIMIT				OSC_DIV				0x00	R/W
0x22	IO	[7:0]	RESERVED							IO2X	0x00	R/W
0x23	TEST_PAT_BYTE0	[7:0]	TEST_DATA_PAT[7:0]								0x0F	R/W
0x24	TEST_PAT_BYTE1	[7:0]	TEST_DATA_PAT[15:8]								0x0F	R/W
0x25	TEST_PAT_BYTE2	[7:0]	TEST_DATA_PAT[23:16]								0x5A	R/W
0x26	TEST_PAT_BYTE3	[7:0]	TEST_DATA_PAT[31:24]								0x5A	R/W
0x34	DIG_DIAG	[7:0]	POWERUP_COMPLETED	RESET_OCCURRED	RESERVED					FUSE_CRC_EN	0x40	R/W
0x35	DIG_ERR	[7:0]	RESERVED							FUSE_CRC_ERR	0x00	R/W

رجیستر های مورد نیاز برای کنترل اولیه ی ADC :

Address: 0x00, Reset: 0x10, Name: INTERFACE_CONFIG_A

Interface configuration settings.

```
       7  6  5  4  3  2  1  0
      ┌──┬──┬──┬──┬──┬──┬──┬──┐
      │ 0│ 0│ 0│ 1│ 0│ 0│ 0│ 0│
      └──┴──┴──┴──┴──┴──┴──┴──┘
```

[7] SW_RESET (R/W)
First of Two SW_RESET Bits.

[6] RESERVED

[5] ADDR_ASCENSION (R)
Determines Sequential Addressing Behavior

[0] SW_RESETX (R/W)
Second of Two SW_RESET Bits.

[3:1] RESERVED

[4] SDO_ENABLE (R)
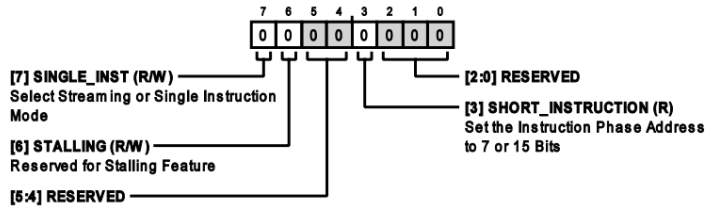SDOx Pin Enable

INTERFACE CONFIGURATION B REGISTER

Address: 0x01, Reset: 0x00, Name: INTERFACE_CONFIG_B

Additional interface configuration settings.

```
       7  6  5  4  3  2  1  0
      ┌──┬──┬──┬──┬──┬──┬──┬──┐
      │ 0│ 0│ 0│ 0│ 0│ 0│ 0│ 0│
      └──┴──┴──┴──┴──┴──┴──┴──┘
```
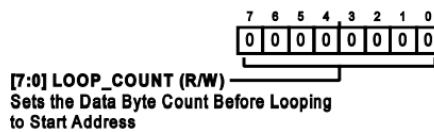
[7] SINGLE_INST (R/W)
Select Streaming or Single Instruction Mode

[6] STALLING (R/W)
Reserved for Stalling Feature

[5:4] RESERVED

[2:0] RESERVED

[3] SHORT_INSTRUCTION (R)
Set the Instruction Phase Address to 7 or 15 Bits

DEVICE CONFIGURATION REGISTER

Address: 0x02, Reset: 0x00, Name: DEVICE_CONFIG

```
       7  6  5  4  3  2  1  0
      ┌──┬──┬──┬──┬──┬──┬──┬──┐
      │ 0│ 0│ 0│ 0│ 0│ 0│ 0│ 0│
      └──┴──┴──┴──┴──┴──┴──┴──┘
```

[7:2] RESERVED

[1:0] OPERATING_MODES (R/W)
Power Modes
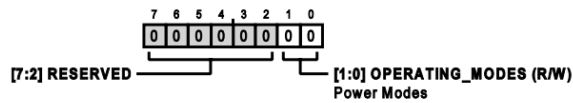
STREAM MODE REGISTER

Address: 0x0E, Reset: 0x00, Name: STREAM_MODE

Defines the length of the loop when streaming data.

```
       7  6  5  4  3  2  1  0
      ┌──┬──┬──┬──┬──┬──┬──┬──┐
      │ 0│ 0│ 0│ 0│ 0│ 0│ 0│ 0│
      └──┴──┴──┴──┴──┴──┴──┴──┘
```

[7:0] LOOP_COUNT (R/W)
Sets the Data Byte Count Before Looping to Start Address

EXIT CONFIGURATION MODE REGISTER

Address: 0x14, Reset: 0x00, Name: EXIT_CFG_MD

```
       7  6  5  4  3  2  1  0
      ┌──┬──┬──┬──┬──┬──┬──┬──┐
      │ 0│ 0│ 0│ 0│ 0│ 0│ 0│ 0│
      └──┴──┴──┴──┴──┴──┴──┴──┘
```

[7:1] RESERVED

[0] EXIT_CONFIG_MD (R/W)
Exit Register Config Mode

8

MODES REGISTER

Address: 0x20, Reset: 0x00, Name: MODES

```
  7 6 5 4 3 2 1 0
 ┌─┬─┬─┬─┬─┬─┬─┬─┐
 │0│0│0│0│0│0│0│0│
 └─┴─┴─┴─┴─┴─┴─┴─┘
```

[7:6] LANE_MD (R/W)
Lane Mode Select

[5:4] CLK_MD (R/W)
Clock Mode Select

[2:0] OUT_DATA_MD (R/W)
Output Data Mode Select

[3] DDR_MD (R/W)
DDR Mode Enable/Disable
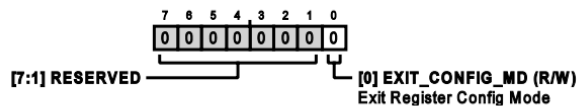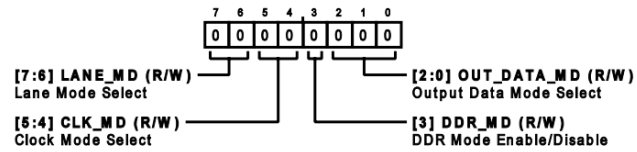
TEST PATTERN REGISTERS

Address: 0x23, Reset: 0x0F, Name: TEST_PAT_BYTE0

```
  7 6 5 4 3 2 1 0
 ┌─┬─┬─┬─┬─┬─┬─┬─┐
 │0│0│0│0│1│1│1│1│
 └─┴─┴─┴─┴─┴─┴─┴─┘
```
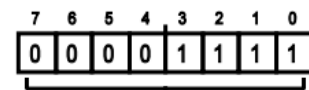
[7:0] TEST_DATA_PAT[7:0] (R/W)
32-Bit Test Pattern

Address: 0x24, Reset: 0x0F, Name: TEST_PAT_BYTE1

```
  7 6 5 4 3 2 1 0
 ┌─┬─┬─┬─┬─┬─┬─┬─┐
 │0│0│0│0│1│1│1│1│
 └─┴─┴─┴─┴─┴─┴─┴─┘
```
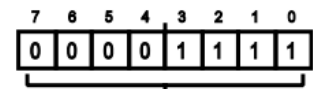
[7:0] TEST_DATA_PAT[15:8] (R/W)
32-Bit Test Pattern

Address: 0x25, Reset: 0x5A, Name: TEST_PAT_BYTE2

```
  7 6 5 4 3 2 1 0
 ┌─┬─┬─┬─┬─┬─┬─┬─┐
 │0│1│0│1│1│0│1│0│
 └─┴─┴─┴─┴─┴─┴─┴─┘
```
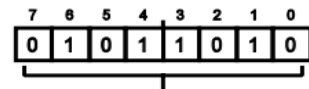
[7:0] TEST_DATA_PAT[23:16] (R/W)
32-Bit Test Pattern

Address: 0x26, Reset: 0x5A, Name: TEST_PAT_BYTE3

```
  7 6 5 4 3 2 1 0
 ┌─┬─┬─┬─┬─┬─┬─┬─┐
 │0│1│0│1│1│0│1│0│
 └─┴─┴─┴─┴─┴─┴─┴─┘
```
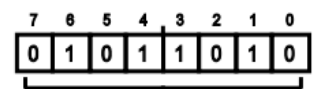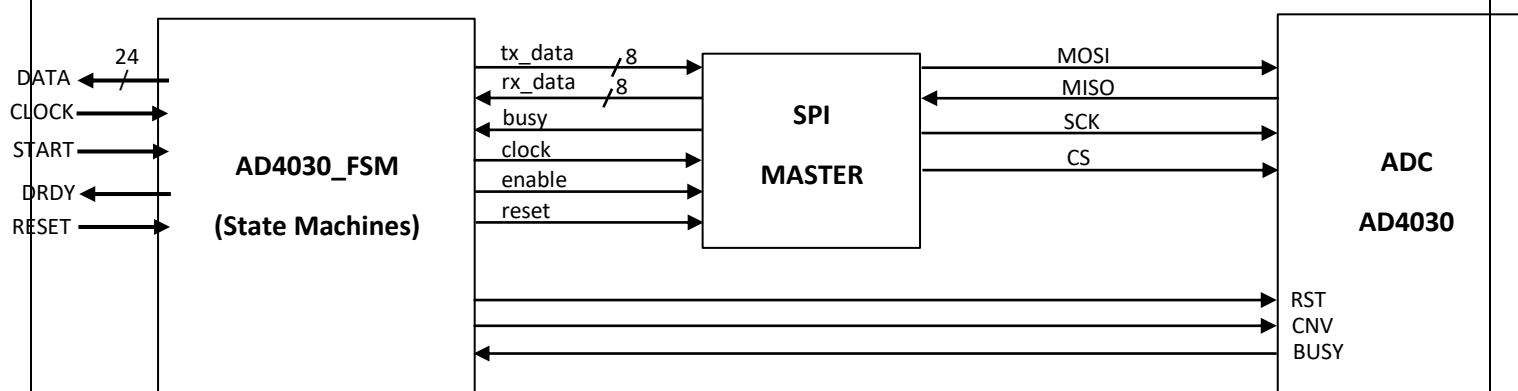
[7:0] TEST_DATA_PAT[31:24] (R/W)
32-Bit Test Pattern

فرمت DATA

دیتای جمع آوری شده توسط مبدل به صورت زیر است که فرمت چینش آن توسط رجیستر MODE_REGISTER قابل تنظیم است.
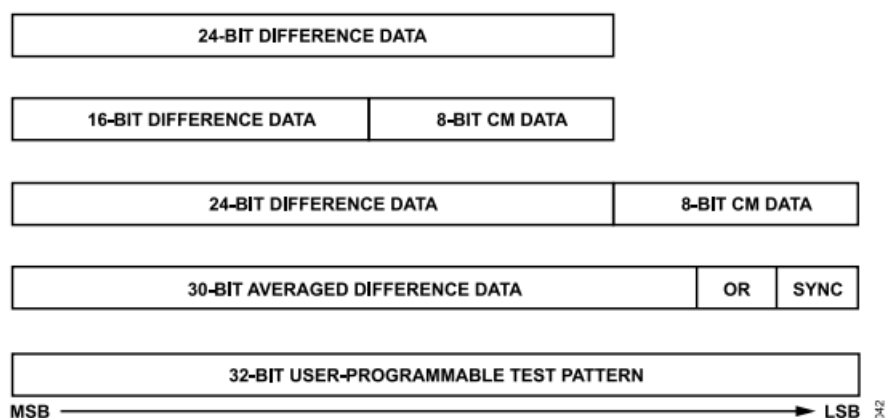
Figure 43. Summary of Selectable Output Sample Formats

ارائه بلوک پیشنهادی سخت افزار برای کنترل ADC مورد نظر:

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.
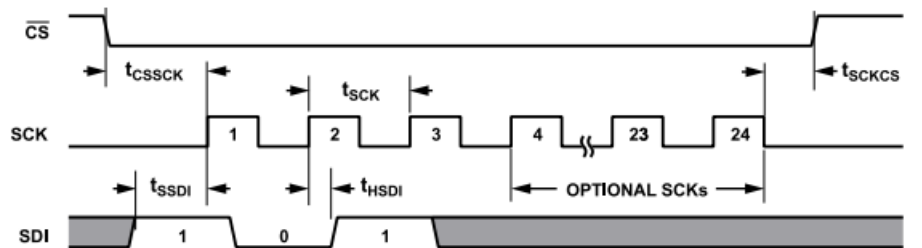

Figure 5. Register Configuration Mode Command Timing
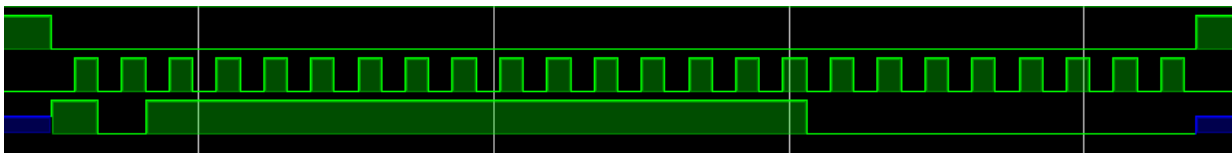
2. سپس جهت تنظیم single instruction mode ، مقدار 0x80 را در رجیستر INTERFACR CONGIGURATION B که در آدرس 0x01 قرار دارد میریزیم

INTERFACE CONFIGURATION B REGISTER

Address: 0x01, Reset: 0x00, Name: INTERFACE_CONFIG_B

Additional interface configuration settings.
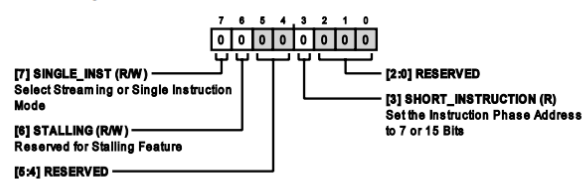
Table 18. Bit Descriptions for INTERFACE_CONFIG_B

Bits	Bit Name	Description	Reset	Access
7	SINGLE_INST	Select Streaming or Single Instruction Mode. 0: streaming mode is enabled. The address decrements as successive data bytes are received. 1: single instruction mode is enabled.	0x0	R/W
6	STALLING	Reserved for Stalling Feature.	0x0	R/W
[5:4]	RESERVED	Reserved.	0x0	R
3	SHORT_INSTRUCTION	Set the Instruction Phase Address to 7 Bits or 15 Bits. 0: 15-bit addressing. 1: 7-bit addressing.	0x0	R
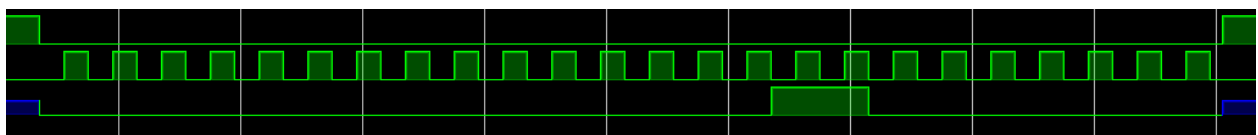
3. سپس جهت تنظیم فومت دیتای خروجی باید رجیستر MODES را تنظیم کنیم . برای تنظیم خروجی سریال یک لاین و مود SPI clocking و همچنین فرمت خروجی 24 بیت ، مقدار 0x00 را در این رجیستر قرار می دهیم.

MODES REGISTER

Address: 0x20, Reset: 0x00, Name: MODES

```
          7  6  5  4  3  2  1  0
         ┌──┬──┬──┬──┬──┬──┬──┬──┐
         │ 0│ 0│ 0│ 0│ 0│ 0│ 0│ 0│
         └──┴──┴──┴──┴──┴──┴──┴──┘
```

[7:6] LANE_MD (R/W) ─────┘   └──── [2:0] OUT_DATA_MD (R/W)
Lane Mode Select   Output Data Mode Select

[5:4] CLK_MD (R/W) ─────┘   └──── [3] DDR_MD (R/W)
Clock Mode Select   DDR Mode Enable/Disable

Table 37. Bit Descriptions for MODES

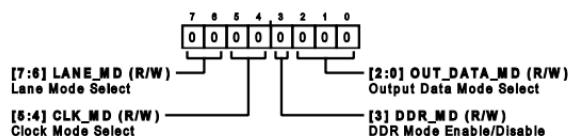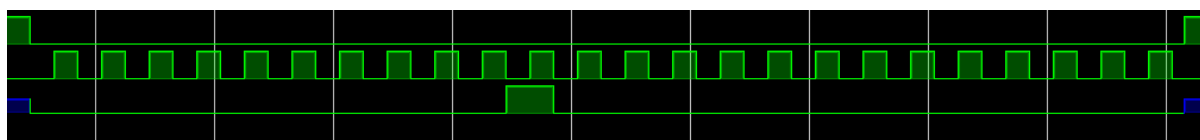| Bits	Bit Name	Description	Reset	Access
[7:6]	LANE_MD	Lane Mode Select. 00 = one lane. 01 = two lanes. 10 = four lanes. 11 = invalid setting.	0x0	R/W
[5:4]	CLK_MD	Clock Mode Select. 00 = SPI clocking mode. 01 = echo clock mode. 10 = host clock mode. 11 = invalid setting.	0x0	R/W
3	DDR_MD	DDR Mode Enable/Disable. 0 = SDR. 1 = DDR (only valid for echo clock mode and host clock mode).	0x0	R/W
[2:0]	OUT_DATA_MD	Output Data Mode Select. 000 = 24-bit differential data. 001 = 16-bit differential data + 8-bit common-mode data. 010 = 24-bit differential data + 8-bit common-mode data. 011 = 30-bit averaged differential data + OR bit + SYNC bit. 100 = 32-bit test data pattern (TEST_DATA_PAT).	0x0	R/W

4. سپس برای خروج از مود تنظیمات باید مقدار 0x01 را در رجیسترِ EXIT CONFIGURATION MODE قرار دهیم.

EXIT CONFIGURATION MODE REGISTER

Address: 0x14, Reset: 0x00, Name: EXIT_CFG_MD

```
          7  6  5  4  3  2  1  0
         ┌──┬──┬──┬──┬──┬──┬──┬──┐
         │ 0│ 0│ 0│ 0│ 0│ 0│ 0│ 0│
         └──┴──┴──┴──┴──┴──┴──┴──┘
```

[7:1] RESERVED ─────┘   └──── [0] EXIT_CONFIG_MD (R/W)
                                     Exit Register Config Mode
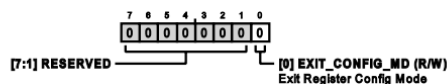
Table 30. Bit Descriptions for EXIT_CFG_MD

Bits	Bit Name	Description	Reset	Access
[7:1]	RESERVED	Reserved.	0x0	R
0	EXIT_CONFIG_MD	Exit Register Config Mode. Write 1 to exit register configuration mode. Self clearing upon CS = 1.	0x0	R/W

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

<u>طراحی بلوک SPI_MASTER</u>


```vhdl
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;
```

```vhdl
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;
```

14

```vhdl
                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;
```
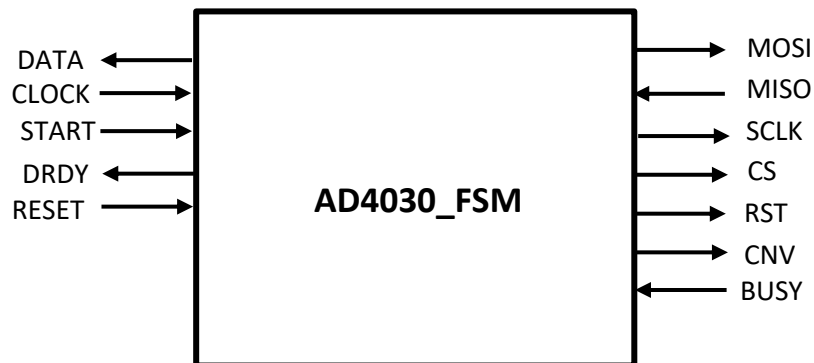
```
entity AD4030_FSM is
    PORT(
        DATA    : OUT    STD_LOGIC_VECTOR(23 DOWNTO 0);    --24-bit data achived from adc convertion
        CLOCK   : IN     STD_LOGIC;                        --main clock
        START   : IN     STD_LOGIC;                        --start of convert states
        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;
```

```
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
```

```vhdl
        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
```

```vhdl
                );

        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
```

```vhdl
          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
```

19

```vhdl
                    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

```vhdl
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;
```

20

```vhdl
---------------------------------------------------------
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;
```

```vhdl
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';
```

22

```vhdl
    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;
```

23

```vhdl
            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
```

24

```vhdl
    );

    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;
```