

Оглавление

1.	Как акселерометр подключен к ПЛИС на плате	2
2.	Демонстрационный проект GSensor (DE10_LITE_GSensor).....	4
2.1.	Модуль сброса reset_delay.....	7
2.2.	Документация на аксель, настройки SPI, И-файл адресов регистров	8
2.3.	SPI модуль spi_controller	12
2.4.	Чтение данных из акселя.....	14

1. Как акселерометр подключен к ПЛИС на плате

На отладочной плате есть акселерометр, по 6 линиям подключён к ПЛИС:

1.4 Block Diagram of the Board

Figure 1-4 gives the block diagram of the board. To provide maximum flexibility for the user, all connections are made through the MAX 10 FPGA device. Thus, the user can configure the FPGA to implement any system design.

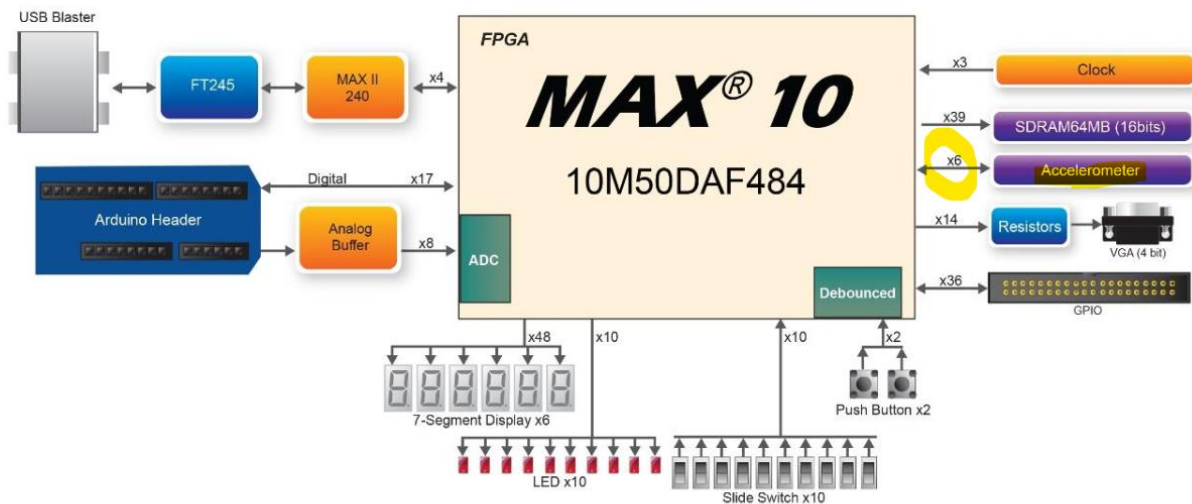


Figure 1-4 Board Block Diagram

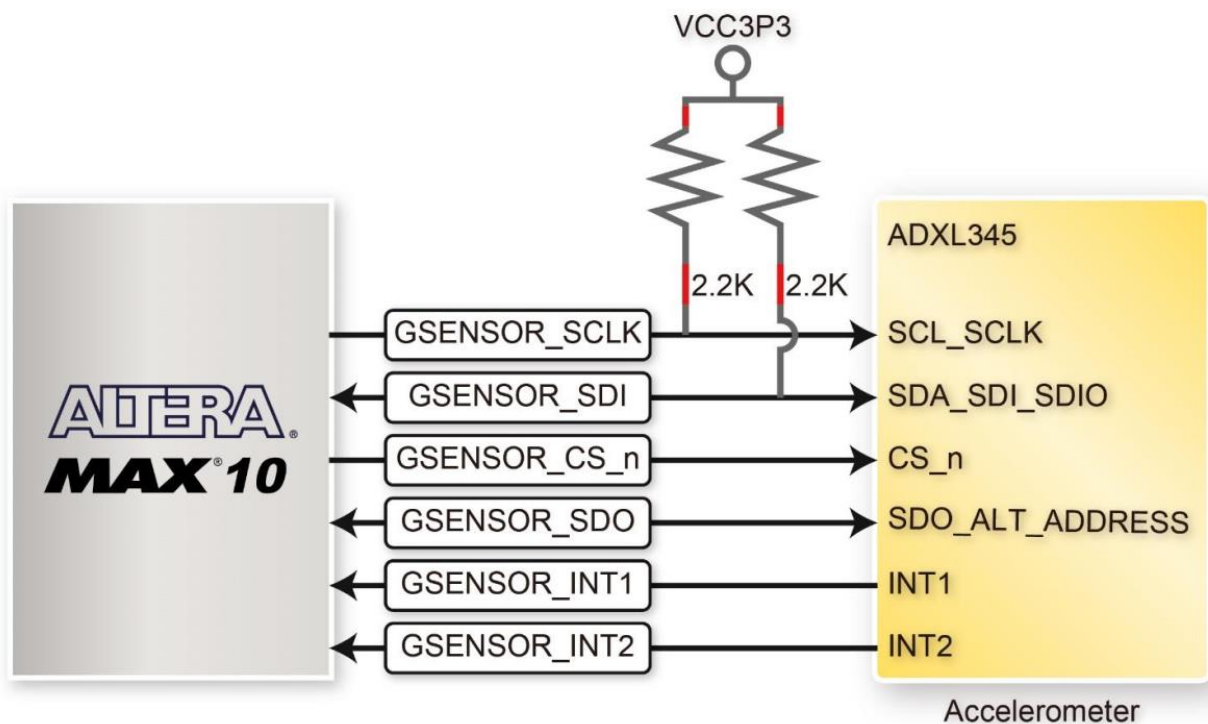


Figure 3-24 shows the connections between the accelerometer sensor and MAX 10 FPGA.

Что мы видим? 4 линии SPI и 2 линии для прерываний. По ним акселерометр может уведомить ПЛИС о наступлении тех или иных событий. Например, «начало движения», в документации на аксель более подробно написано.

Table 3-13 Pin Assignment of Accelerometer Sensor

Signal Name	FPGA Pin No.	Description	I/O Standard
GSENSOR_SDI	PIN_V11	I2C serial data SPI serial data input (SPI 4-wire) SPI serial data input and output (SPI 3-wire)	3.3-V LVTTL
GSENSOR_SDO	PIN_V12	SPI serial data output (SPI 4-wire) Alternate I2C address select	3.3-V LVTTL
GSENSOR_CS_n	PIN_AB16	I2C/SPI mode selection: 1: SPI idle mode / I2C communication enabled 0: SPI communication mode / I2C disabled SPI Chip Select	3.3-V LVTTL
GSENSOR_SCLK	PIN_AB15	I2C serial clock SPI serial clock (3- and 4-wire)	3.3-V LVTTL
GSENSOR_INT1	PIN_Y14	Interrupt pin 1	3.3-V LVTTL
GSENSOR_INT2	PIN_Y13	Interrupt pin 2	3.3-V LVTTL

Как интересно! Акселерометр поддерживает 2 интерфейса, I2C или SPI на тех же самых ногах. Кстати, вот поэтому вверху на рис мы видим подтягивающие резисторы 2.2кОм. Они нужны для I2C. Также есть описание способа выбора интерфейса.

With GSENSOR_CS_N signal to high, the ADXL345 is in I2C mode. With the GSENSOR_SDO signal to high, the 7-bit I2C address for the device is 0x1D, followed by the R/W bit. This translates to 0x3A for a write and 0x3B for a read. An alternate I2C address of 0x53 (followed by the R/W bit) can be chosen by low the GSENSOR_SDO signal. This translates to 0xA6 for a write and 0xA7 for a read.

Есть описание тестового проекта – раздел 5. 5 G-Sensor:

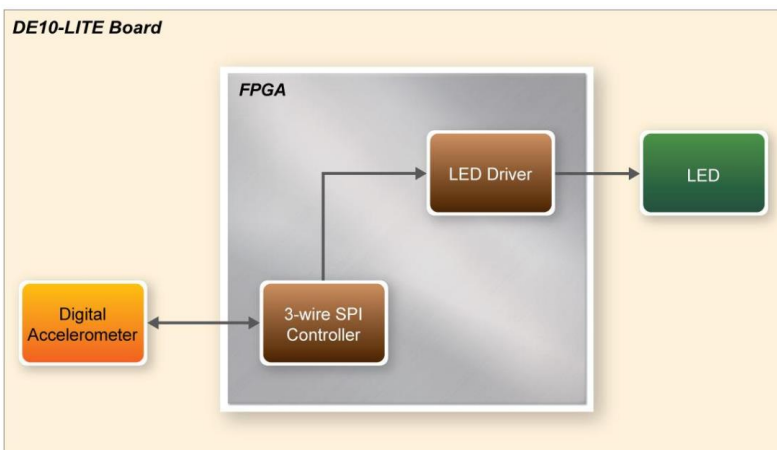
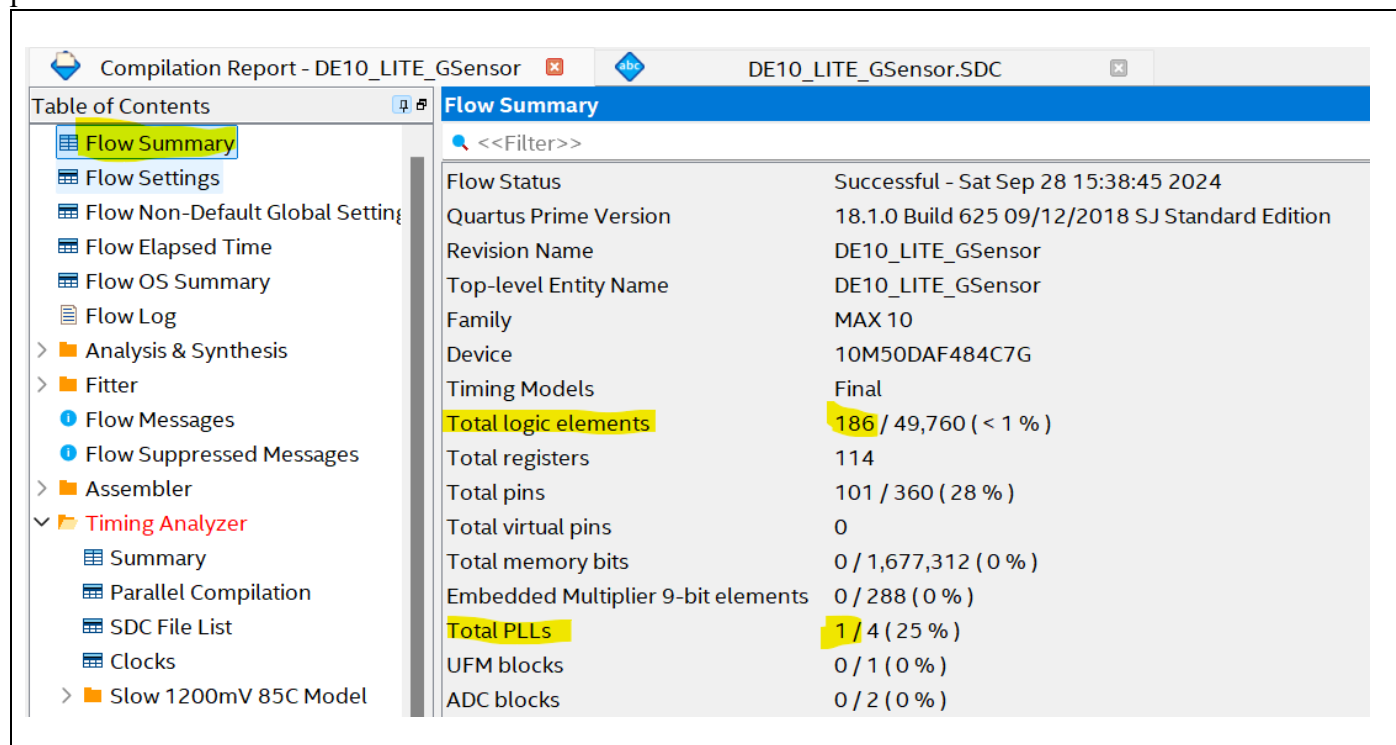


Figure 5-8 Block diagram of the G-Sensor

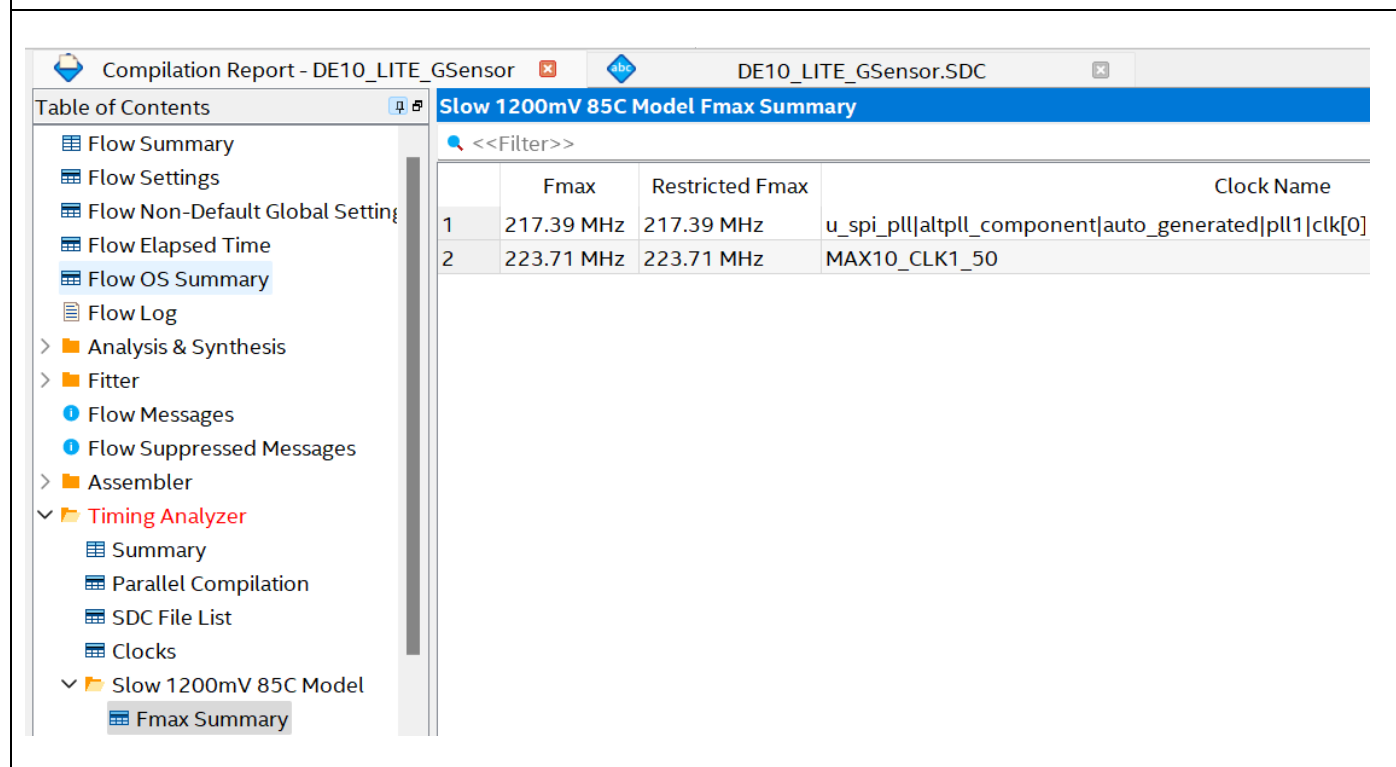
2. Демонстрационный проект GSensor (DE10_LITE_GSensor)

Идем смотреть проект, смотрим сколько логики занимает, на какой максимальной частоте работает.



The screenshot shows the Quartus Prime IDE with the 'Flow Summary' report open. The 'Table of Contents' on the left lists various reports, with 'Flow Summary' selected. The main area displays the following data:

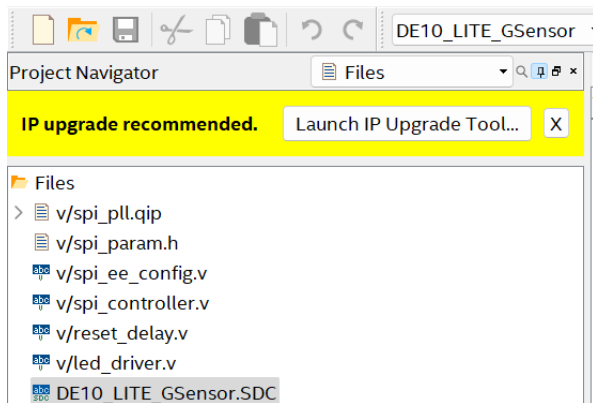
Flow Summary	
Flow Status	Successful - Sat Sep 28 15:38:45 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Standard Edition
Revision Name	DE10_LITE_GSensor
Top-level Entity Name	DE10_LITE_GSensor
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	186 / 49,760 (< 1 %)
Total registers	114
Total pins	101 / 360 (28 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	1 / 4 (25 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)



The screenshot shows the Quartus Prime IDE with the 'Slow 1200mV 85C Model Fmax Summary' report open. The 'Table of Contents' on the left lists various reports, with 'Fmax Summary' selected. The main area displays the following data:

	Fmax	Restricted Fmax	Clock Name
1	217.39 MHz	217.39 MHz	u_spi_pll altpll_component auto_generated pll1 clk[0]
2	223.71 MHz	223.71 MHz	MAX10_CLK1_50

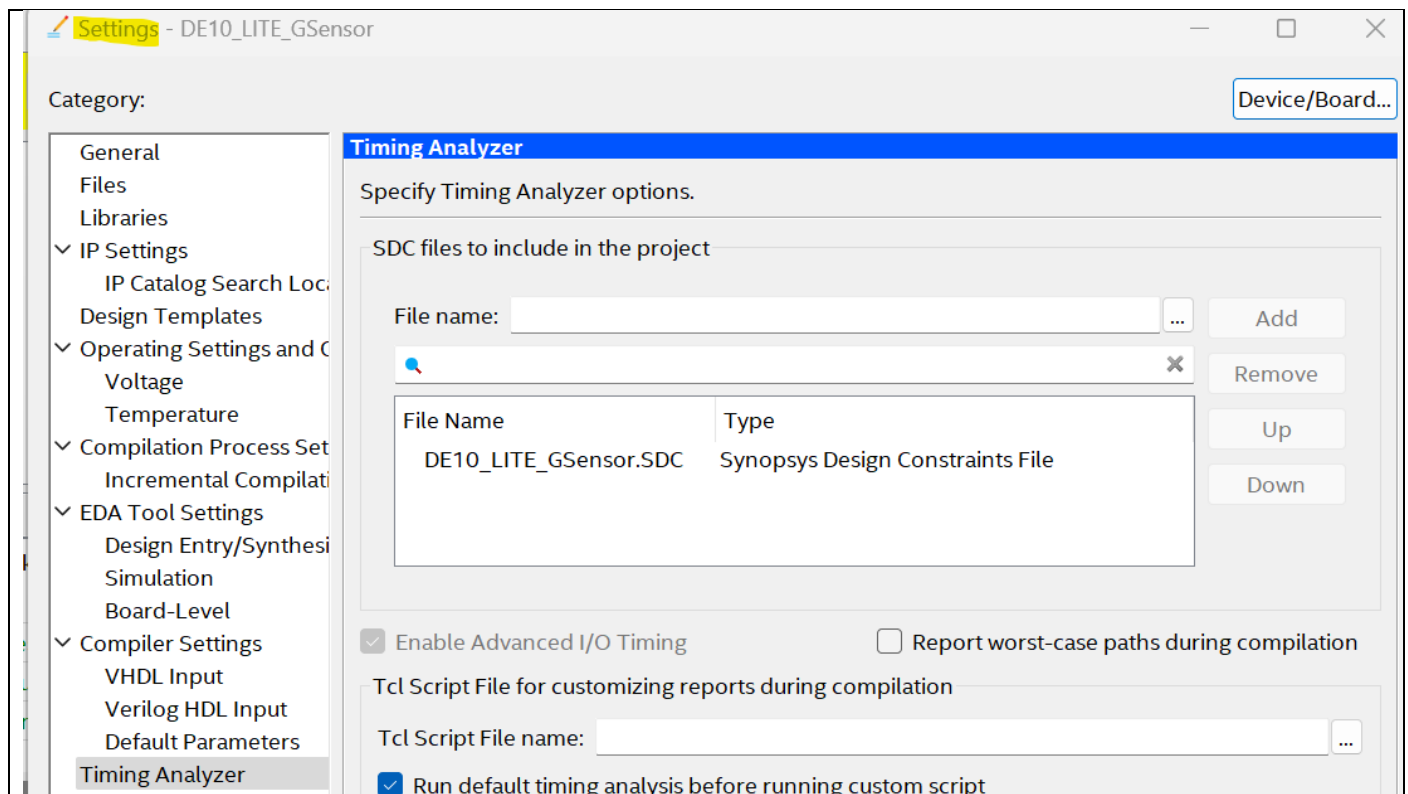
Кстати, тут есть файл SDC – synopsis design constraints – описание временных ограничений на проект. Так называемый «файл констрейнов». В нашем случае мы говорим, какие клоки заходят на кристалл (на какой пин и какая частота тактового сигнала). Вообще, SDC – это как ТЗ для синтезатора, он пробует разместить на кристалле так, чтобы констрейны были выполнены. Поэтому в серьезных проектах это очень важный файл.



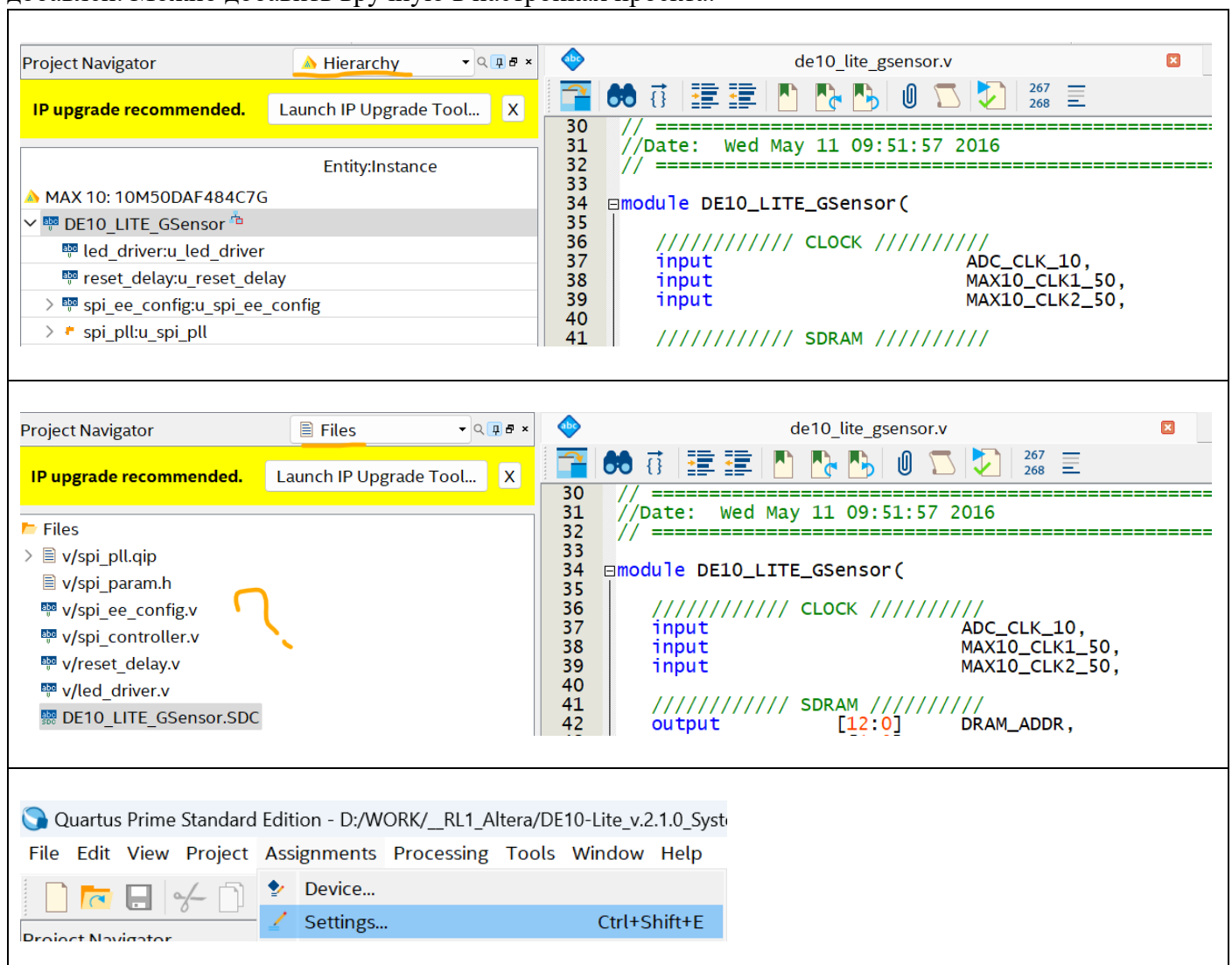
```

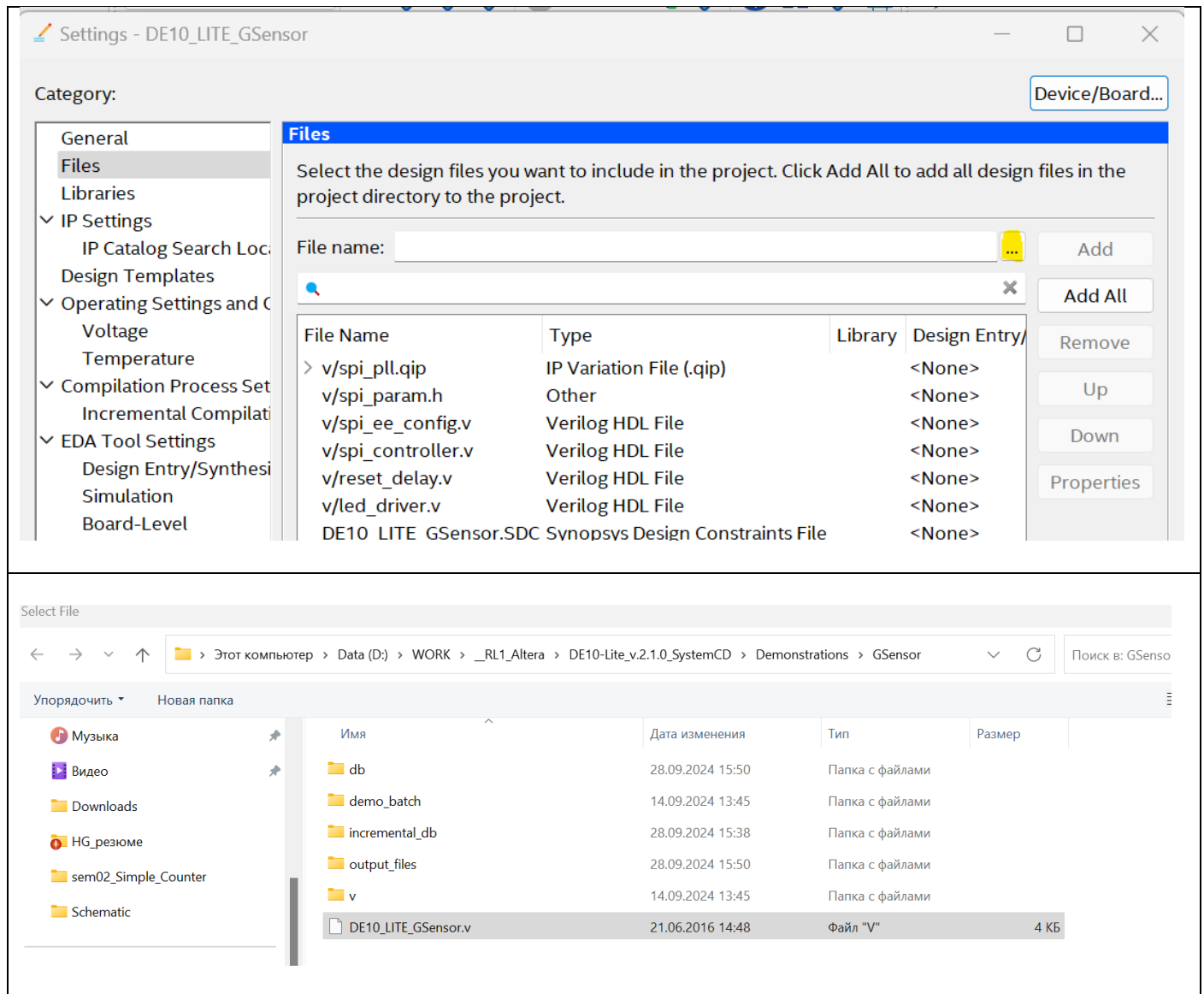
1 *****
2 # This .sdc file is created by Terasic Tool.
3 # Users are recommended to modify this file to match users logic.
4 *****
5
6 *****
7 # Create Clock
8 *****
9 create_clock -period "10.0 MHz" [get_ports ADC_CLK_10]
10 create_clock -period "50.0 MHz" [get_ports MAX10_CLK1_50]
11 create_clock -period "50.0 MHz" [get_ports MAX10_CLK2_50]
12
13 *****
14 # Create Generated Clock
15 *****
16 derive_pll_clocks
17
18
19
20 *****
21 # Set Clock Latency
22 *****
23
24
25
26 *****
27 # Set Clock Uncertainty
28 *****
29 derive_clock_uncertainty
30

```



В готовом проекте есть момент – top_level модуль виден в hierarchy, а в файлах он почему-то не добавлен. Можно добавить вручную в настройках проекта.





2.1. Модуль сброса reset_delay

Смотрим код. Модуль сброса, который при нажатии на кнопку и в течении $2 \times 20 = 1$ млн тактов на 50 МГц = 20 мс формирует положительный сигнал сброса, который идет на PLL.

```
// Reset
//при нажатии на кнопку и еще некоторое время после отпускания oRST = 1
reset_delay u_reset_delay
(
    .iRSTN (KEY[0]),
    .iCLK  (MAX10_CLK1_50),
    .oRST  (dly_rst)
);

module reset_delay(iRSTN, iCLK, oRST);

input      iRSTN;
input      iCLK;
output reg oRST;

reg [20:0] cont;

always @(posedge iCLK or negedge iRSTN)
    if (!iRSTN)
        begin
            cont    <= 21'b0;
            oRST    <= 1'b1;
        end
    else if (!cont[20])
        begin
            cont <= cont + 21'b1;
            oRST <= 1'b1;
        end
    else
        ;
endmodule
```



```
ORST <= 1'b0;
```

```
endmodule
```

2.2. Документация на аксель, настройки SPI, И-файл адресов регистров

Идем смотреть документацию на акселерометр. Нас интересуют тайминги SPI и настроечные регистры.

> __RL1_Altera > DE10-Lite_v.2.1.0_SystemCD > Datasheet > G-Sensor



Сортировать ▾



Просмотреть ▾

...

Имя

Дата изменения

Тип



ADXL345

15.06.2016 18:48

Foxit PhantomPDF P...

SPI

For SPI, either 3- or 4-wire configuration is possible, as shown in the connection diagrams in Figure 34 and Figure 35. Clearing the SPI bit (Bit D6) in the DATA_FORMAT register (Address 0x31) selects 4-wire mode, whereas setting the SPI bit selects 3-wire mode. The maximum SPI clock speed is 5 MHz with 100 pF maximum loading, and the timing scheme follows clock polarity (CPOL) = 1 and clock phase (CPHA) = 1. If power is applied to the ADXL345 before the clock polarity and phase of the host processor are configured, the CS pin should be brought high before changing the clock polarity and phase. When using 3-wire SPI, it is recommended that the SDO pin be either pulled up to V_{DD I/O} or pulled down to GND via a 10 kΩ resistor.

Use of the 3200 Hz and 1600 Hz output data rates is only recommended with SPI communication rates greater than or equal to 2 MHz. The 800 Hz output data rate is recommended only for communication speeds greater than or equal to 400 kHz, and the remaining data rates scale proportionally. For example, the minimum recommended communication speed for a 200 Hz output data rate is 100 kHz. Operation at an output data rate

Table 10. SPI Timing (T_A = 25°C, V_S = 2.5 V, V_{DD I/O} = 1.8 V)¹

Parameter	Limit ^{2, 3}		Unit	Description
	Min	Max		
f _{SCLK}		5	MHz	SPI clock frequency
t _{SCLK}	200		ns	1/(SPI clock frequency) mark-space ratio for the SCLK input is 40/60 to 60/40
t _{DELAY}	5		ns	\overline{CS} falling edge to SCLK falling edge
t _{QUIET}	5		ns	SCLK rising edge to \overline{CS} rising edge
t _{DIS}		10	ns	\overline{CS} rising edge to SDO disabled
t _{CS,DIS}	150		ns	\overline{CS} deassertion between SPI communications
t _S	0.3 × t _{SCLK}		ns	SCLK low pulse width (space)
t _M	0.3 × t _{SCLK}		ns	SCLK high pulse width (mark)
t _{SETUP}	5		ns	SDI valid before SCLK rising edge
t _{HOLD}	5		ns	SDI valid after SCLK rising edge
t _{SDO}		40	ns	SCLK falling edge to SDO/SDIO output transition
t _R ⁴		20	ns	SDO/SDIO output high to output low transition
t _F ⁴		20	ns	SDO/SDIO output low to output high transition

¹ The \overline{CS} , SCLK, SDI, and SDO pins are not internally pulled up or down; they must be driven for proper operation.

² Limits based on characterization results, characterized with f_{SCLK} = 5 MHz and bus load capacitance of 100 pF; not production tested.

³ The timing values are measured corresponding to the input thresholds (V_{IL} and V_{IH}) given in Table 9.

⁴ Output rise and fall times measured with capacitive load of 150 pF.

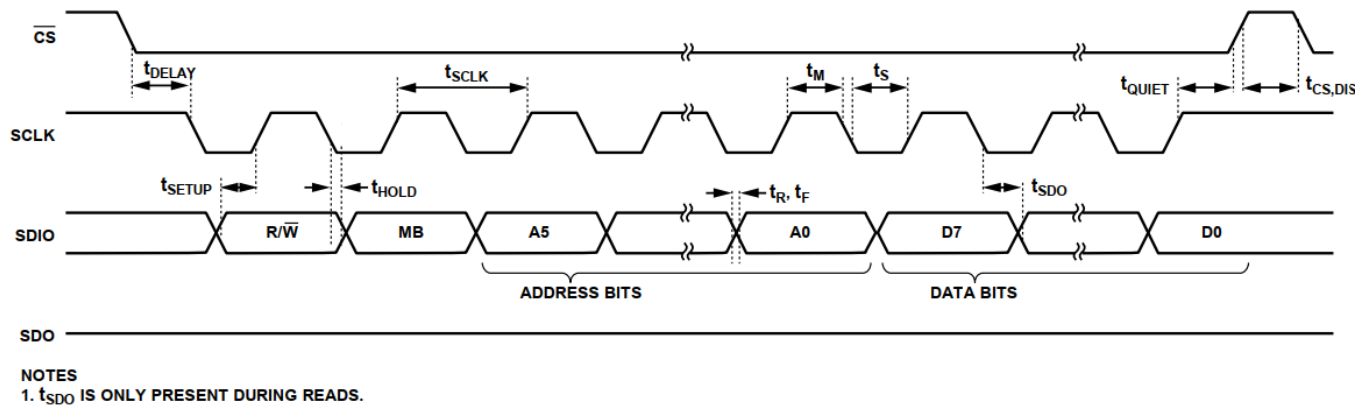
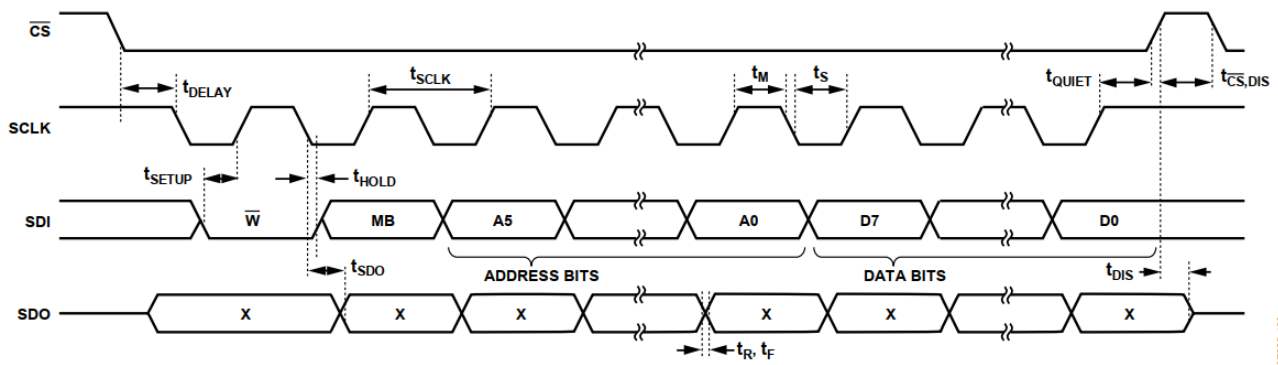


Figure 38. SPI 3-Wire Read/Write

REGISTER MAP

Table 19.

Address		Name	Type	Reset Value	Description
Hex	Dec				
0x00	0	DEVID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/ \overline{W}	00000000	Tap threshold
0x1E	30	OFSX	R/ \overline{W}	00000000	X-axis offset
0x1F	31	OFSY	R/ \overline{W}	00000000	Y-axis offset
0x20	32	OFSZ	R/ \overline{W}	00000000	Z-axis offset
0x21	33	DUR	R/ \overline{W}	00000000	Tap duration
0x22	34	Latent	R/ \overline{W}	00000000	Tap latency
0x23	35	Window	R/ \overline{W}	00000000	Tap window
0x24	36	THRESH_ACT	R/ \overline{W}	00000000	Activity threshold
0x25	37	THRESH_INACT	R/ \overline{W}	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/ \overline{W}	00000000	Inactivity time

Сравниваем Н файл и PDF даташит:

```
1 module spi_ee_config (
2
3     iRSTN,
4     iSPI_CLK,
5     iSPI_CLK_OUT,
6     iG_INT2,
7     oDATA_L,
8     oDATA_H,
9     SPI_SDIO,
10    oSPI_CSN,
11    oSPI_CLK);
12
13    //подключаем файл констант - адреса регистров в акселе
14    include "spi_param.h"
```

Project Navigator

Files

IP upgrade recommended. Launch IP Upgrade Tool..

Files

v/spi_pll.qip

v/spi_param.h

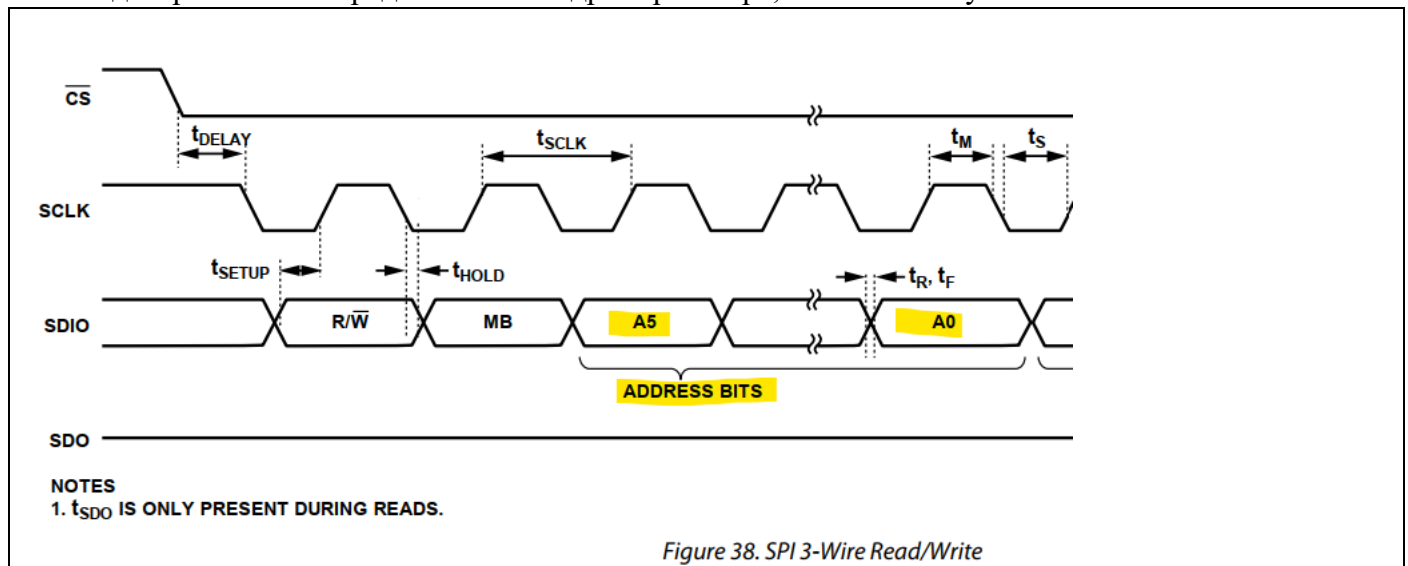
```
// Write Reg Address
parameter BW_RATE          = 6'h2c;
parameter POWER_CONTROL   = 6'h2d;
parameter DATA_FORMAT    = 6'h31;
parameter INT_ENABLE      = 6'h2E;
parameter INT_MAP         = 6'h2F;
parameter THRESH_ACT      = 6'h24;
parameter THRESH_INACT    = 6'h25;
parameter TIME_INACT      = 6'h26;
parameter ACT_INACT_CTL   = 6'h27;
parameter THRESH_FF       = 6'h28;
parameter TIME_FF         = 6'h29;

// Read Reg Address
parameter INT_SOURCE       = 6'h30; // INT Status
parameter X_LB             = 6'h32; // Low Byte
parameter X_HB             = 6'h33; // High Byte
parameter Y_LB             = 6'h34; // Low Byte
parameter Y_HB             = 6'h35; // High Byte
parameter Z_LB             = 6'h36; // Low Byte
parameter Z_HB             = 6'h37; // High Byte
```

0x2C	44	BW_RATE	R/W	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/W	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/W	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/W	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupts
0x31	49	DATA_FORMAT	R/W	00000000	Data format control
0x32	50	DATA_X0	R	00000000	X-Axis Data 0
0x33	51	DATA_X1	R	00000000	X-Axis Data 1
0x34	52	DATA_Y0	R	00000000	Y-Axis Data 0
0x35	53	DATA_Y1	R	00000000	Y-Axis Data 1
0x36	54	DATA_Z0	R	00000000	Z-Axis Data 0
0x37	55	DATA_Z1	R	00000000	Z-Axis Data 1

!!! Обращаем внимание, что в Verilog модуле нет отдельных SDI, SDO пинов, а общий SDIO, т.е. модуль рассчитан на 3-wire SPI. Поэтому в даташите будем искать тайминги и моменты касавшего 3-wire SPI.

По диаграмме SPI передается 6 бит адреса регистра, и константы у нас тоже 6-битные.



```

17 // Write Reg Address
18 parameter BW_RATE = 6'h2c;
19 parameter POWER_CONTROL = 6'h2d;
20 parameter DATA_FORMAT = 6'h31;
21 parameter INT_ENABLE = 6'h2E;

```

После адреса идут 8 бит данных, а перед адресом – какие-то биты RW и MB, итого имеем:
 { 1'RW, 1'MB, 6'ADDR, 8'DATA }

To read or write multiple bytes in a single transmission, the multiple-byte bit, located after the R/W bit in the first byte transfer (MB in Figure 36 to Figure 38), must be set. After the register addressing and the first byte of data, each subsequent set of clock pulses (eight clock pulses) causes the ADXL345 to point to the next register for a read or write. This shifting continues until the clock pulses cease and \overline{CS} is deasserted. To perform reads or writes on different, nonsequential registers, \overline{CS} must be deasserted between transmissions and the new register must be addressed separately.

```

reg [3:0] ini_index;
reg [SI_DataL-2:0] write_data; //[15-2:0] => [13:0]

// Initial Reg Number - сколько регистров будем настраивать в чипе
parameter INI_NUMBER = 4'd11;

// данные, которые шлем по 3-wire SPI, { 6'REG_ADDR, 8'REG_DATA }
// Initial Setting Table
always @ (ini_index)
  case (ini_index)
    0 : write_data = {THRESH_ACT, 8'h20};
    1 : write_data = {THRESH_INACT, 8'h03};
    2 : write_data = {TIME_INACT, 8'h01};
    3 : write_data = {ACT_INACT_CTL, 8'h7f};
    4 : write_data = {THRESH_FF, 8'h09};
    5 : write_data = {TIME_FF, 8'h46};
    6 : write_data = {BW_RATE, 8'h09}; // output data rate : 50 Hz
    7 : write_data = {INT_ENABLE, 8'h10};
    8 : write_data = {INT_MAP, 8'h10};
    9 : write_data = {DATA_FORMAT, 8'h40};
    default: write_data = {POWER_CONTROL, 8'h08};
  endcase

```

```

6 // Write/Read Mode
7 // { 1'RW, 1'MB }, 2 первых бита, передаваемых в SPI транзакции
8 parameter WRITE_MODE = 2'b00;
9 parameter READ_MODE = 2'b10;

```

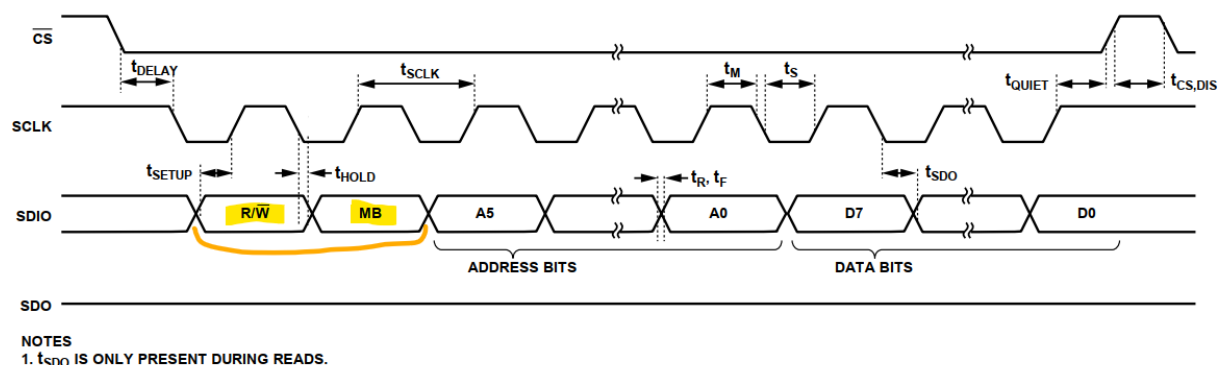


Figure 38. SPI 3-Wire Read/Write

```

104 // initial setting (write mode)
105 else if(ini_index < INI_NUMBER)
106     case(spi_state)
107     IDLE : begin
108         p2s_data <= {WRITE_MODE, write_data};
109         spi_go    <= 1'b1;
110         spi_state <= TRANSFER;
111     end

```

Итого, сразу после снятия ресет – начинается настройка акселя. Для этого в него посылаются данные 11 настроечных регистров.

```

reg [3:0] ini_index; //индекс из 'таблицы настроечных регистров'
reg [SI_DataL-2:0] write_data; //[15-2:0] => [13:0], { 6'REG_ADDR, 8'REG_DATA }
reg [SI_DataL:0] p2s_data; //[15:0] - все 16 бит SPI транзакции
//{ 1'RW, 1'MB, 6'REG_ADDR, 8'REG_DATA }

always@(posedge iSPI_CLK or negedge iRSTN)
    if(!iRSTN)
        begin
            ini_index <= 4'b0; //индекс в 'таблице настроечных регистров'
            spi_go <= 1'b0;
            spi_state <= IDLE;
            read_idle_count <= 0; // read mode only
            high_byte <= 1'b0; // read mode only
            read_back <= 1'b0; // read mode only
            clear_status <= 1'b0;

            end
            // по очереди перебираем регистры из таблицы настроеч регистров, и собираем SPI транзакцию
            // для отправки. Собираем { 1'RW, 1'MB, 6'REG_ADDR, 8'REG_DATA } в p2s_data и запускаем
            // SPI_FSM. После окончания транзакции (spi_end) увеличиваем индекс в таблице регистров
            // initial setting (write mode)
            else if(ini_index < INI_NUMBER)
                case(spi_state)
                    IDLE : begin
                        p2s_data <= {WRITE_MODE, write_data};
                        spi_go <= 1'b1;
                        spi_state <= TRANSFER;

                        end
                    TRANSFER : begin
                        if (spi_end)
                            begin
                                ini_index <= ini_index + 4'b1;
                                spi_go <= 1'b0;
                                spi_state <= IDLE;

                                end
                            end
                        endcase

```

2.3. SPI модуль spi_controller

```

module spi_controller (
    iRSTN,
    iSPI_CLK, //2MHz
    iSPI_CLK_OUT, //2MHz phase shift
    iP2S_DATA, //передаваемая транзакция { 1'RW, 1'MB, 6'REG_ADDR, 8'REG_DATA }
    iSPI_GO, //сигнал, что надо запустить SPI транзакцию
    oSPI_END, //индикатор, что SPI транзакция завершена
    oS2P_DATA,
    SPI_SDIO,
    oSPI_CSN,
    oSPI_CLK
);

`include "spi_param.h"

//=====
// PORT declarations
//=====
// Host Side
input iRSTN;
input iSPI_CLK;
input iSPI_CLK_OUT;
input [SI_DataL:0] iP2S_DATA; //[15:0]
input iSPI_GO;
output oSPI_END;
output reg [SO_DataL:0] oS2P_DATA; //[7:0]
// SPI Side

```

```

inout      SPI_SDIO;
output     oSPI_CSN;
output     oSPI_CLK;

//=====
// REG/WIRE declarations
//=====
wire       read_mode, write_address;
reg        spi_count_en;
reg        [3:0] spi_count; //счетчик бит в транзакции,
                             //считает с 15 до 0, т.к. передача старшим битом вперед

//=====
// Structural coding
//=====
assign read_mode = iP2S_DATA[SI_DataL]; // [15], бит RW

//передаем 16 бит, [15:0], { 1'RW, 1'MB, 6'REG_ADDR, 8'REG_DATA }
//даже если транзакция чтения, то все равно сначала надо первые 8 бит передать в чип
//т.е. биты [15:8], когда счетчик spi_count = 15,14..8, у него бит [3] всегда в '1'
assign write_address = spi_count[3];

//oSPI_END - индикатор, что все 16 бит переданы/приняты, и счетчик бит сейчас стал == 0
assign oSPI_END = ~|spi_count; // '1' если все биты равны 0
                                // spi_count устанавливается в 15 в начале транзакции
                                // и считает вниз до 0

assign oSPI_CSN = ~iSPI_GO;

//если нет транзакции, то неактивный уровень линии CLK '1'
//выходной блок даем "задержанным" на фазовый сдвиг относительно сигнала тактирования
//синхр логики этого блока, чтобы данные успели выставиться на линии
assign oSPI_CLK = spi_count_en ? iSPI_CLK_OUT : 1'b1;

//если нет транзакции (en=0), то на линии HiZ
//если Write транзакция (read=0), то выставляем биты из iP2S_DATA[spi_count]
//если Read транзакция (read=1), но идет передача адресных + сервисных битов,
//то выставляем биты из iP2S_DATA[spi_count]
assign SPI_SDIO = spi_count_en && (!read_mode || write_address) ? iP2S_DATA[spi_count] : 1'bz;

always @ (posedge iSPI_CLK or negedge iRSTN)
  if (!iRSTN)
    begin
      spi_count_en <= 1'b0;
      spi_count <= 4'hf; //счетчик бит
    end
  else
    begin
      if (oSPI_END) //как только счетчик бит == 0, транзакция завершена
        spi_count_en <= 1'b0;
      else if (iSPI_GO) //сигнал запуска транзакции от головного модуля
        spi_count_en <= 1'b1; //засчелкиваем и "удерживаем" EN

      //как только spi count_en = 1, начинается счет бит 15,14...0
      if (!spi_count_en)
        spi_count <= 4'hf;
      else
        spi_count <= spi_count - 4'b1; //считает с 15-ого бита до 0

      //если транзакция чтения и биты RW, MB, 6'REG_ADDR передали, то теперь
      //принимаем биты 8'REG_DATA
      if (read_mode && !write_address)
        oS2P_DATA <= {oS2P_DATA[SO_DataL-1:0], SPI_SDIO}; //сдвиговый регистр, новый бит
      //вдвигается в младший разряд
    end
  end
endmodule

```

2.4. Чтение данных из акселя

У нас есть 2 условия начала чтения данных из акселя:

- (1) Счетчик досчитал
- (2) На пине IG_INT2 появился сигнал прерывания

```
reg [IDLE_MSB:0] read_idle_count; // reducing the reading rate, [14:0]
    case (spi_state)
        IDLE : begin
            read_idle_count <= read_idle_count + 1;
        end
    else if (!clear_status_d[3]&&iG_INT2 || read_idle_count[IDLE_MSB])
    begin
        p2s_data[15:8] <= {READ_MODE, INT_SOURCE};
        clear_status <= 1'b1;
    end
```

Эти условия запускают чтение регистра «причина прерывания» **INT_SOURCE**. Как только начинаем цикл чтения, сразу ставим триггер **clear_status**, и он будет удерживаться в 1, пока не закончится чтение.

Также **clear_status** задерживаем в сдвиговом регистре. И повторный цикл чтения можно запустить только после того, как задержанный **clear_status** сбросится. Т.е. даже если у нас запросы на чтение **INT_SOURCE** друг за другом идут (аля только по счетчику времени опросили регистр, и тут же прерывание пришло, или прерывания друг за другом), то между соседними SPI транзакциями будет пауза в несколько тактов.

После окончания транзакции также сбрасывается счетчик времени **read_idle_count**.

```
always@(posedge iSPI_CLK or negedge iRSTN)
    if (!iRSTN)
    begin
        ...
        clear_status_d <= 4'b0;
    end
    else
    begin
        ...
        //задержка clear_status на N тактов
        clear_status_d <= {clear_status_d[2:0], clear_status};
    end
end

TRANSFER : begin
    if (spi_end)
    begin
        ...
    else
    begin
        clear_status <= 1'b0;
        read_ready <= s2p_data[6];
        read_idle_count <= 0;
```

Результат чтения – значение регистра **INT_SOURCE**, и там 6й бит может сказать, что было кратковременной ускорение “tap”. В этом случае надо считать значение ускорения из регистров **X_LB**, **X_HB**. Выставляется флаг **read_ready**.

Register 0x30—INT_SOURCE (Read Only)

D7 DATA_READY	D6 SINGLE_TAP	D5 DOUBLE_TAP	D4 Activity
D3 Inactivity	D2 FREE_FALL	D1 Watermark	D0 Overrun

Bits set to 1 in this register indicate that their respective functions have triggered an event, whereas a value of 0 indicates that the corresponding event has not occurred. The DATA_READY, watermark, and overrun bits are always set if the corresponding events occur, regardless of the INT_ENABLE register settings, and are cleared by reading data from the DATA_X, DATA_Y, and DATA_Z registers. The DATA_READY and watermark bits may require multiple reads, as indicated in the FIFO mode descriptions in the FIFO section. Other bits, and the corresponding interrupts, are cleared by reading the INT_SOURCE register.

SINGLE_TAP

The SINGLE_TAP bit is set when a single acceleration event that is greater than the value in the THRESH_TAP register (Address 0x1D) occurs for less time than is specified in the DUR register (Address 0x21).

У **read_ready** больше приоритет, чем у запуска чтения INT_SOURCE, поэтому, если чтение INT_SOURCE показало, что было прерывание, то **read_ready** запускает чтение **X_LB**, даже если там есть повторный запрос на чтение INT_SOURCE.

```

else if (read_ready)
begin
    p2s_data[15:8] <= {READ_MODE, X_LB};
    read_back      <= 1'b1;
end

else if (!clear_status_d[3]&&iG_INT2 || read_idle_count[IDLE_MSB])
begin
    p2s_data[15:8] <= {READ_MODE, INT_SOURCE};
    clear_status   <= 1'b1;
end

```

При начале цикла чтения **X_LB** защелкивается “1” в **read_back**. И держится до окончания транзакции. Кстати, при чтении **X_LB** у нас **high_byte** равен 0, а вот в конце чтения делаем инверсию и он равен 1. У проверки это флага приоритет еще выше, чем у **read_ready**, и он запустит цикл чтения **X_HB**.

```

TRANSFER : begin
    if (spi_end)
    begin
        ...
        if (read_back)
        begin
            read_back      <= 1'b0;
            high_byte      <= !high_byte; // X_LB, X_HB, X_LB...
            read_ready     <= 1'b0;
        end
    end

else
    case(spi_state)
        IDLE : begin
            read_idle_count <= read_idle_count + 1;

        if (high_byte) // multiple-byte read
        begin
            p2s_data[15:8] <= {READ_MODE, X_HB};
            read_back      <= 1'b1;
        end

        else if (read_ready)
        begin
            p2s_data[15:8] <= {READ_MODE, X_LB};
            read_back      <= 1'b1;
        end

        else if (!clear_status_d[3]&&iG_INT2 || read_idle_count[IDLE_MSB])
        begin
            p2s_data[15:8] <= {READ_MODE, INT_SOURCE};
            clear_status   <= 1'b1;
        end
    end

```

Есть сдвиговые регистры – задержки на 1 такт для high_byte, read_back. По ним защелкиваются данные X_LB и X_HB в регистры oDATA_L, oDATA_H, которые потом передаются в модуль светодиодов.

```
if (read_back_d) // update the read back data
begin
    if (high_byte_d)
    begin
        oDATA_H <= s2p_data;
        oDATA_L <= low_byte_data;
    end
    else
        low_byte_data <= s2p_data;
    end
end
```