Протокол передачи данных SPI

SPI (Serial Peripheral Interface) — последовательный синхронный протокол передачи данных в режиме полного дуплекса по определению, требующий соединения взаимодействующих устройств четырьмя проводами (с учетом общей земли 5). Субъекты, участвующие в передачи данных по SPI классифицируются по их роли. Выделяют роли главного «Master» и подчиняющегося «Slave» устройств. Принято, что устройство «Master» являются управляющим для всех подчиняющихся таким образом, что одной системе сообщения зачастую на одно устройство «Master» приходится на счетное количество устройств «Slave». В элементарном случае два устройства соединяются между собой посредством четырех проводов : MOSI, MISO, SCLK, CS.

Таблица 1. Цифровые сигналы в протоколе SPI.

Аббревиатура названия сигнала	Расшифровка аббревиатуры	Назначение цифрового сигнала
MOSI	Master Output Slave Input	Передача данных от ведущего к ведомому
MISO	Master Input Slave Output	Передача данных от ведомого к ведущему
SCLK	Serial Clock	Последовательный тактовый сигнал. Задается ведущим устройством.
CS	Chip Select	Сигнал выбора конкретной микросхемы. Количество этих проводов равно количеству подчиненных устройств в рассматриваемой параллельной топологии. В последовательной топологии провод CS один.

В зависимости от производителя аппаратных средств конкретные названия цифровых портов интерфейса SPI могут отличаться.

Синхронизация в SPI

Передача данных по проводам MISO и MOSI синхронизирована сигналом SCLK. Этот сигнал определяет частоту передачи данных. Важно, что ведомые не могут повлиять на частоту синхроимпульса. В ведущем и ведомых устройствах расположены счетчики, которые отсчитывают импульсы. Сброс счетчиков обеспечивается за счет изменения сигнала CS. Так как действия ведущего в ведомого устройств тактируются одним и тем же сигналом, то к стабильности синхроимпульса не предъявляются строгие требования.

Прием и передача данных

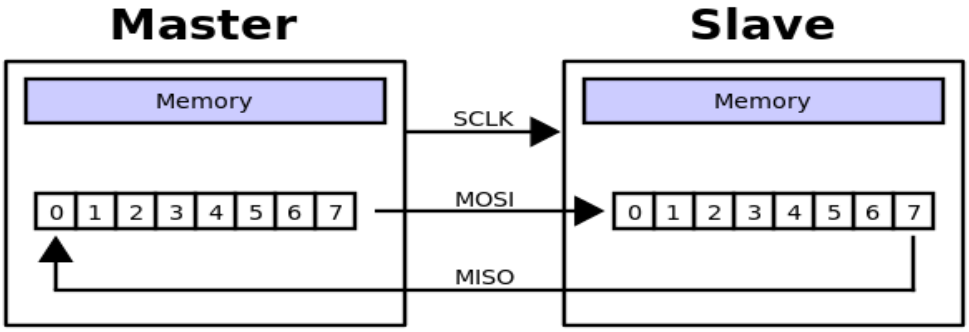

Рис 1. Классическая схема включения с одним подчиненны устройством.

Передача через данный протокол реализуется пакетами. В самом распространенном случае информационный объем пакета составляет 1 байт, но известны и иные длины пакетов, поэтому в данной работе модуль, написанный на языке проектирования аппаратуры Verilog HDL, параметризован таким образом, что можно изменять длину пакета элементарной транзакции.

Начало транзакции инициализирует «Master» установкой низкого сигнала на порт CS с выбранным устройством. В этом состоянии «Slave» начинает воспринимать сигналы SCLK и MOSI, а порт MISO переводится из Z состояния так, что там устанавливается первый бит для передачи данных. Данные побитово передаются от ведущего к ведомому и наоборот старшим битом вперед. В целях более качественной синхронизации, после передачи каждого пакета «Master» устанавливает на CS высокий уровень сигнала.

Вариации протокола SPI

Существуют 4 комбинации двух параметров CPHA и CPOL, задающих режим работы интерфейса.

Таблица 2. Характеристика параметров CPHA и CPOL, влияние последних на протокол передачи данных.

Параметр	Назначение параметра	Возможные значения параметров	Влияние значения параметров на протокол
CPHA (Clock Phase)	Соотношение между фазой сигнала SCLK и данными на шинах MOSI и MISO.	0	Выборка данных производится по переднему фронту сигнала синхронизации
		1	Выборка данных производится по заднему фронту сигнала синхронизации
CPOL (Clock Polarity)	Соотношение меду полярностью сигнала SCLK и данными на шинах	0	Сигнал синхронизации начинается с низкого уровня
		1	Сигнал синхронизации начинается с высокого уровня

	MOSI и MISO.		

Графическое представление изменения картины ансамбля сигналов элементарной транзакции 1 байта представлены на рисунке ниже.

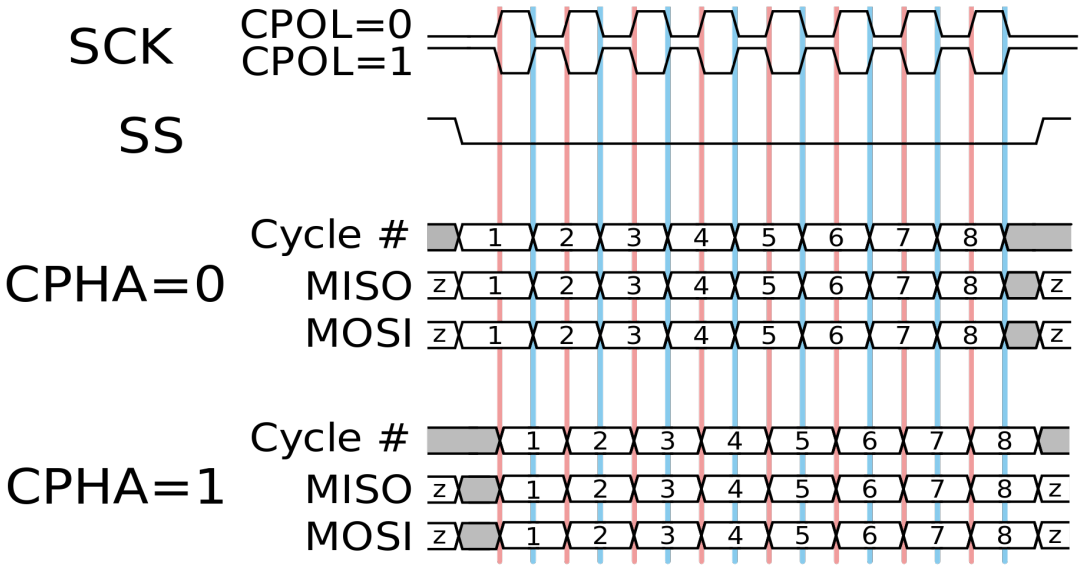

Рис. 2. Графическое представление четырех вариантов протокола SPI на примере передачи одного байта.

Топология реализованной системы связи

Наиболее часто используемой топологий связи устройств, общающихся через SPI, является топология типа «Звезда». В этом случае для общения с n устройствами потребуется n+3 (также имеется общая земля) проводов за счет предоставления каждому новому устройству отдельного провода CS. Смысл сказанного выше иллюстрирован на Рис. 3.

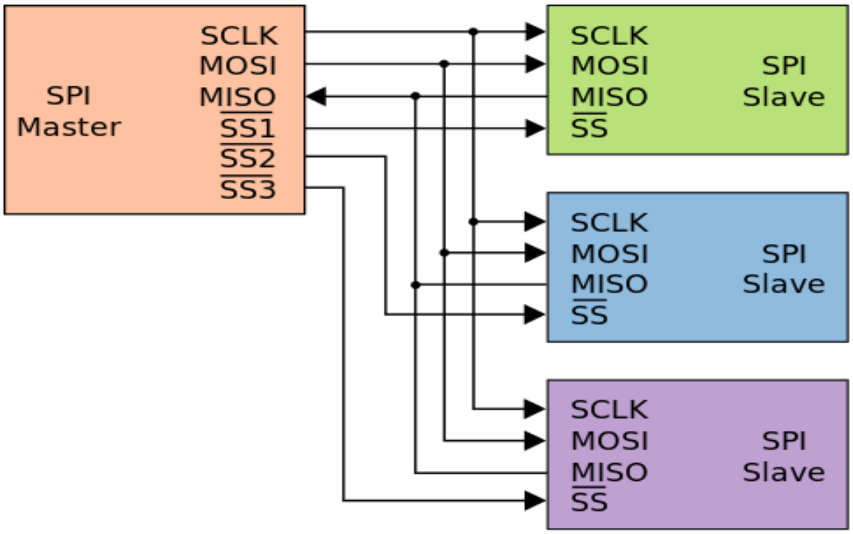

Рис. 3. Радиальная структура связи с несколькими подчиненными устройствами через SPI.

Легко видеть, что все устройства типа «Slave» подключены одновременно

к шинам SCLK, MOSI, MISO, но для каждого существует индивидуальный провод активации SS (Slave Select, аналог порту CS). Важно заметить, что такой вариант включения (такая топология) дает преимущество в скорости сообщения с конкретным подчиненным устройством. Из недостатков можно отметить пропорциональный рост количества портов при возрастании числа устройств «Slave».

Преимущества и недостатки протокола SPI

Лучшим образом проиллюстрировать качественные отличия рассматриваемого протокола можно через таблицу, что представлена ниже.

Таблица 3. Преимущества и недостатки протокола SPI.

Преимущества	**Недостатки**
Полнодуплексная передача данных по умолчанию	Количество проводов больше, чем в интерфейсах 1-wire, I2C, UART
Высокая пропускная способность, выше, чем у I2C	Отсутствует подтверждение приема данных со стороны ведомого устройства
Возможность произвольного выбора длины пакета элементарной транзакции, нет строгого ограничения в 8 бит	Стандарт не предусматривает возможность отслеживания ошибки (как это, например, имеется в UART через бит четности/нечетности)
Однонаправленный характер сигналов на портах позволяет организовать при необходимости гальваническую развязку	Существует 4 варианта протокола, нет единого соглашения
Ведомым устройствам не нужен уникальный адрес, обращение к каждому производится через выставление низкого уровня сигнала на порт CS	Подключение дополнительного устройства на горячую не предусмотрено (как это возможно, например, в I2C)
Возможность использования протокола в системах с низкостабильной тактовой частотой	
Используется 4 порта для соединения двух устройств, что меньше, чем в параллельных интерфейсах	

Реализация устройства типа Master на языке Verilog HDL

Параметризованный модуль «Master» имеет следующую структуру параметров и входных/выходных пинов. (см. Таблица 4.)

Таблица 4. Описание параметров и пинов модуля SPI_FPGA_MASTER

Параметры настройки модуля	**Описание параметра/пина**
BIT_PER_SECOND	Устанавливаемая пользователем скорость передачи данных в одном направлении, бит/сек.
CLOCK_FREQUENCY	Частота тактового сигнала. Сигнал является входным в модуль. Параметр CLOCK_FREQUENCY должен быть в 4 раза больше, чем BIT_PER_SECOND.
CLKS_PER_BIT_LOG_2	Округленный до целого числа логарифм по основанию 2 от частного параметров CLOCK_FREQUENCY/ (BIT_PER_SECOND*2). Это значение определяет объем регистра — счетчика.
PACK_LENGTH	Длина пакета элементарной транзакции
PACK_LENGTH_LOG_2	Логарифм по основанию 2, округленный до целого, от величины PACK_LENGTH
CPOL	Соотношение меду полярностью сигнала SCLK и данными на шинах MOSI и MISO. (См. Выше).
CPHA	Соотношение между фазой сигнала SCLK и данными на шинах MOSI и MISO. (См. выше)
Входные пины	
IN_CLOCK	Входной тактовый сигнал с частотой CLOCK_FREQUENCY.
IN_LAUNCH	Сигнал, обозначающий начало транзакции. Его полоса должна быть несколько периодов сигнала IN_CLOCK, но не больше длины одной транзакции. Активный уровень-высокий сигнал.
IN_DATA [PACK_LENGTH-1:0]	Данные, которые будут отправлены подчиненному устройству следующей

	транзакцией. Имеет размерность PACK_LENGTH – количество бит в одном пакете. При детектировании сигнала LAUNCH эти данные защелкиваются в модуле, поэтому нет необходимости их держать неизменными всю транзакцию.
MISO	Порт, по которому данные отправляются подчиненным устройством мастеру.
Выходные пины	
MOSI	Тип wire. Порт, по которому данные отправляются от мастера к подчиненному.
CS	Тип reg. Порт выбора подчиненного устройства.
SCLK	Тип reg. Тактовый сигнал, генерируемый мастером для синхронизации передачи данных.
OUT_RECEIVE_DATA [PACK_LENGTH-1:0]	Тип wire. Данные, которые мастер принимает от «Slave» в результате транзакции.
OUT_ACTION_DONE	Тип reg. Сигнал, кратковременно возникающий на этом порту после окончания процесса элементарной транзакции. Активный уровень - логическая «1».

Таблица 5. Описание используемых регистров модуля SPI_FPGA_MASTER

Регистр	**Размерность регистра**	**Описание регистра**
REG_FSM_STATE	[1:0]	Регистр содержит в себе текущее состояние конечного автомата. Всего состояний автомата насчитывается 3, что влечет за собой размерность данного регистра в два бита.
REG_CLOCK_COUNT	[CLKS_PER_BIT_LOG_2:0]	Регистр-счетчик. Размерность регистра определяется тем, насколько CLOCK_FREQUENCY больше BIT_PER_SECOND.

REG_TRANSMIT_DATA	[PACK_LENGTH-1:0]	Регистр данных, которые мастер отправит подчиненному устройству. В этот регистр защелкивается вектор данных при детектировании на IN_LAUNCH высокого уровня сигнала.
REG_BIT_INDEX	[PACK_LENGTH_LOG_2:0]	Регистр-счетчик номера бита для регистров принимаемых и отправляемых данных.
REG_RECEIVE_DATA_1	[PACK_LENGTH-1:0]	Регистр, в который записываются данные первым always- блоком при чтении их с шины MISO.
REG_RECEIVE_DATA_2	[PACK_LENGTH-1:0]	Регистр, в который записываются данные вторым always- блоком при чтении их с шины MISO.
REG_TRANSMIT_BIT_1	1	Регистр, в который заряжается текущий бит из REG_TRANSMIT_DATA в втором always- блоке для отправки в порт MOSI.
REG_TRANSMIT_BIT_2	1	Регистр, в который заряжается текущий бит из REG_TRANSMIT_DATA в третьем always- блоке для отправки в порт MOSI.
REG_MOSI_Z_STATE	1	Регистр- индикатор Z состояния на шине MOSI.
REG_FLAG_START	1	Регистр- индикатор начала сообщения.

Программный код модуля SPI_FPGA_MASTER (Шрифт 10).

```
module SPI_FPGA_MASTER
#(
        parameter BIT_PER_SECOND                =       12500000;
        parameter CLOCK_FREQUENCY               =       50000000;
        parameter CLKS_PER_BIT_LOG_2            =       12;
        parameter PACK_LENGTH_LOG_2             =       4;
        parameter PACK_LENGTH                   =       5;
        parameter CPOL                          =       1'b0;
        parameter CPHA                          =       1'b0
)
(
input                                           IN_CLOCK,
input                                           IN_LAUNCH,
```

```
input [PACK_LENGTH-1:0]                                 IN_DATA,
input                                                   MISO,
output wire                                             MOSI,
output reg                                              CS,
output reg                                              SCLK,
output [PACK_LENGTH-1:0]                                OUT_RECEIVE_DATA,
output reg                                              OUT_ACTION_DONE
);
initial begin
        CS                                      =       1;
        SCLK                                    =       CPOL;
        REG_FSM_STATE                           =       STATE_WAIT;
        REG_RECEIVE_DATA_2                      =       0;
        REG_RECEIVE_DATA_1                      =       0;
        REG_CLOCK_COUNT                         =       0;
        REG_TRANSMIT_DATA                       =       0;
        REG_BIT_INDEX                           =       PACK_LENGTH-1'b1;
        REG_RECEIVE_DATA_2                      =       0;
        REG_RECEIVE_DATA_1                      =       0;
        REG_TRANSMIT_BIT_1                      =       0;
        REG_TRANSMIT_BIT_2                      =       0;
        REG_MOSI_Z_STATE                        =       1;
        REG_FLAG_START                          =       0;
end
parameter BIT_PER_SECOND_MUL_2                  =BIT_PER_SECOND*2;
parameter CLKS_PER_BIT                          =CLOCK_FREQUENCY/BIT_PER_SECOND_MUL_2;
parameter STATE_WAIT                            =2'b00;
parameter STATE_ACTION                          =2'b01;
parameter STATE_WAIT_AFTER_TRANSACTION          =2'b10;
parameter RISE_EDGE                             =1'b1;
parameter FALLING_EDGE                          =1'b0;
parameter WRITE_MODE                            =CPOL^CPHA;
parameter READ_MODE                             =!WRITE_MODE;

reg [1:0]                                       REG_FSM_STATE;
reg [CLKS_PER_BIT_LOG_2:0]                      REG_CLOCK_COUNT;
reg [PACK_LENGTH-1:0]                           REG_TRANSMIT_DATA;
reg [PACK_LENGTH_LOG_2:0]                       REG_BIT_INDEX;
reg [PACK_LENGTH-1:0]                           REG_RECEIVE_DATA_2;
reg [PACK_LENGTH-1:0]                           REG_RECEIVE_DATA_1;
reg                                             REG_TRANSMIT_BIT_1;
reg                                             REG_TRANSMIT_BIT_2;
reg                                             REG_MOSI_Z_STATE;
reg                                             REG_FLAG_START;
assign MOSI= REG_MOSI_Z_STATE ? 1'bZ: (WRITE_MODE==RISE_EDGE)?
        REG_TRANSMIT_BIT_1:REG_TRANSMIT_BIT_2;
assign OUT_RECEIVE_DATA= (READ_MODE==RISE_EDGE)?
REG_RECEIVE_DATA_1:REG_RECEIVE_DATA_2;

always @ (posedge IN_CLOCK)
begin
        case (REG_FSM_STATE)
                STATE_WAIT:
                begin
                        REG_MOSI_Z_STATE=1;
                        CS<=1;
                        SCLK<=CPOL;
                        OUT_ACTION_DONE<=0;
                        if(IN_LAUNCH)
                                begin
                                        REG_FLAG_START<=1;
                                        REG_MOSI_Z_STATE<=0;
                                        CS<=0;
```

```
                                        REG_FSM_STATE<=STATE_ACTION;
                                        REG_BIT_INDEX<=PACK_LENGTH-1;
                                        REG_TRANSMIT_DATA<=IN_DATA;
                                end
                        end
                        STATE_ACTION:
                        begin
                                REG_FLAG_START<=0;
                                if(REG_CLOCK_COUNT<CLKS_PER_BIT*2-1)
                                begin
                                        REG_CLOCK_COUNT=REG_CLOCK_COUNT+1;
                                        if(REG_CLOCK_COUNT==CLKS_PER_BIT)
                                                SCLK=!SCLK;
                                end
                                else
                                begin
                                        REG_CLOCK_COUNT<=0;
                                        SCLK<=!SCLK;
                                        if(REG_BIT_INDEX>0)
                                        begin
                                                REG_BIT_INDEX=REG_BIT_INDEX-1'b1;
                                        end
                                        else
                                        begin
                                                REG_BIT_INDEX=PACK_LENGTH-1'b1;
                                                REG_FSM_STATE<=STATE_WAIT_AFTER_TRANSACTION;
                                                if (!CPHA)REG_MOSI_Z_STATE<=1;
                                        end
                                end
                        end
                        STATE_WAIT_AFTER_TRANSACTION:
                        begin
                                if(REG_CLOCK_COUNT<CLKS_PER_BIT-1)
                                        REG_CLOCK_COUNT=REG_CLOCK_COUNT+1;
                                else
                                begin
                                        OUT_ACTION_DONE<=1;
                                        CS<=1;
                                        REG_CLOCK_COUNT<=0;
                                        REG_FSM_STATE<=STATE_WAIT;
                                        REG_MOSI_Z_STATE<=1;
                                end

                        end
                endcase
end

always@(posedge SCLK or posedge REG_FLAG_START)
begin
        if (REG_FLAG_START)
        begin
                REG_TRANSMIT_BIT_1<=REG_TRANSMIT_DATA[PACK_LENGTH-1];
        end
        else
        begin
                if(WRITE_MODE==RISE_EDGE)
                begin
                        if(!CPHA)
                                REG_TRANSMIT_BIT_1 <= REG_TRANSMIT_DATA[REG_BIT_INDEX-1];
                        else
                                REG_TRANSMIT_BIT_1<=REG_TRANSMIT_DATA[REG_BIT_INDEX];
                end
                else
```

```
                        REG_RECEIVE_DATA_1[REG_BIT_INDEX]<=MISO;
        end
end
always@(negedge SCLK or posedge REG_FLAG_START)
begin
        if(REG_FLAG_START)
        begin
                REG_TRANSMIT_BIT_2<=REG_TRANSMIT_DATA[PACK_LENGTH-1];
        end
        else
        begin
                if(WRITE_MODE==FALLING_EDGE)
                begin
                        if(!CPHA)
                                REG_TRANSMIT_BIT_2<=REG_TRANSMIT_DATA[REG_BIT_INDEX-1];
                        else
                                REG_TRANSMIT_BIT_2<=REG_TRANSMIT_DATA[REG_BIT_INDEX];
                end
                else
                        REG_RECEIVE_DATA_2[REG_BIT_INDEX]<=MISO;
        end
end

endmodule
```

Описание структуры модуля

Модуль представлен 3 поведенческими блоками и несколькими комбинаторными выражениями.

Структура первого поведенческого блока представляет собой конечный автомат 3 состояниями :STATE_WAIT, STATE_ACTION, STATE_WAIT_AFTER_TRANSACTION. Список чувствительности первого always- блока замыкается на переднем фронте сигнала IN_CLOCK. Основная задача этого блока — формирование сигналов SCLK, REG_FLAG_START и REG_BIT_INDEX. Каждое из состояний включается последовательно. Изначально (по умолчанию) автомат находится в состоянии STATE_WAIT. В этом состоянии линия MOSI держится в Z состоянии, на проводе CS держится высокий уровень сигнала, на SCLK установлен уровень CPOL, OUT_ACTION_DONE притянуто к «0». При детектировании положительного уровня сигнала на порту IN_LAUNCH (по переднему фронту IN_CLOCK) в REG_FLAG_STARTS записывается «1», в REG_MOSI_Z_STATE пишется «1», что означает снятие Z состояния с шины MOSI, CS переходит в состояние низкого сигнала. В REG_FSM_STATE записывается состояние STATE_ACTION, что означает смену состояния конечного автомата. Счетчику индекса битов присваивается PACK_LENGTH-1. Кроме всего прочего, происходит защелкивание данных с шины IN_DATA в регистр REG_TRANSMIT_DATA.

В следующем состоянии STATE_ACTION сбрасывается REG_FLAG_START (Что означает, что REG_FLAG_START будет равным «1» ровно один такт основного генератора после детектирования сигнала на IN_LAUNH). В текущем состоянии присутствует комплексный счетчик. В котором полный цикл длится CLKS_PER_BIT*2 тактов. На половине и в конце цикла инвертируется состояние SCLK. По окончании цикла сбрасывается счетчик цикла REG_CLOCK_COUNT в ноль. Также в конце цикла есть условный оператор, проверяющий на нуль REG_BIT_INDEX: если

REG_BIT_INDEX>0, то происходит вычитание единицы из REG_BIT_INDEX и цикл повторяется, если REG_BIT_INDEX ==0, то происходит «сброс» регистра REG_BIT_INDEX присваиванием ему PACK_LENGTH-1. Состояние конечного автомата меняется на STATE_WAIT_AFTER_TRANSACTION, а также если CHPA==0, то шина MOSI ставится в Z состояние, если CPHA==1, то в Z состояние она ставится позже.

В состоянии STATE_WAIT_AFTER_TRANSACTION существует обычный счетчик в CLKS_PER_BIT тактов, по окончании работы которого следующие регистры изменяются следующим образом

Таблица 6. Изменение регистров по окончании состояния STATE_WAIT_AFTER_TRANSACTION

Изменение регистра	**Комментарий**
OUT_ACTION_DONE<=1	На линию OUT_ACTION_DONE ставится высокий уровень сигнала, но после входа в состояние STATE_WAIT на следующем такте OUT_ACTION_DONE будет сброшен в «0». Следовательно, появится узкий импульс, сигнализирующий об окончании транзакции.
CS<=1	Происходит отключение подчиненного устройства. Подчиненное устройство выставляет свои выходные порты в Z состояние.
REG_CLOCK_COUNT<=0	Сброс ходового счетчика.
REG_FSM_STATE<=STATE_WAIT	Конечный автомат переходит в состояние STATE_WAIT.
REG_MOSI_Z_STATE<=1	Шина MOSI переходит в Z состояние.

Второй и третий always- блоки чувствительны к переднему и заднему фронтам сигнала SCLK соответственно, а так же оба чувствительны к переднему фронту сигнала REG_FLAG_START. Задачи данных блоков — запись данных в регистры REG_TRANSMIT_BIT_1 и REG_TRANSMIT_BIT_2, считывание данных с порта MISO в REG_RECEIVE_DATA_1 и REG_RECEIVE_DATA_2. Поскольку язык Verilog HDL не позволяет изменять одну переменную сразу в нескольких поведенческих блоках, то все изменяемые в них регистры дублированы, а затем комбинационной логикой происходит выбор нужного регистра.

Код второго поведенческого блока

```
always@(posedge SCLK or posedge REG_FLAG_START)
begin
	if (REG_FLAG_START)
	begin
		REG_TRANSMIT_BIT_1<=REG_TRANSMIT_DATA[PACK_LENGTH-1];
	end
	else
	begin
		if(WRITE_MODE==RISE_EDGE)
		begin
			if(!CPHA)
				REG_TRANSMIT_BIT_1<=REG_TRANSMIT_DATA[REG_BIT_INDEX-1];
			else
				REG_TRANSMIT_BIT_1<=REG_TRANSMIT_DATA[REG_BIT_INDEX];
		end
		else
			REG_RECEIVE_DATA_1[REG_BIT_INDEX]<=MISO;
	end
end
```

Описание второго поведенческого блока

Легко видеть, что список чувствительности “posedge SCLK or posedge REG_FLAG_START” именно такой, как описан выше. Реакция на передний фронт сигнала записывается через указание posedge перед его названием в списке чувствительности.

Считается, что сигнал REG_FLAG_START является кратковременным, поэтому первым делом происходит сепарация списка чувствительности через высокий уровень сигнала REG_FLAG_START внутри поведенческого блока для выделения причины входа в сам блок. Эта операция проделывается для определения скачка CS вниз для выставления первого бита на шину MOSI (поскольку передача идет старшим битом вперед, то выставляется PACK_LENGTH-1 по счету бит).

Если выясняется что вход в блок произошел по причине появления переднего фронта сигнала SCLK, то дальше условный оператор выявляет конкретный вариант работы протокола (вариант работы протокола всецело определяется двумя параметрами : CPHA, CPOL) : если запись происходит по переднему фронту, то выставляется бит (номер которого зависит от CPHA), если нет, то, значит, по переднему фронту происходит считывание, что влечет за собой считывание в регистр REG_RECEIVE_DATA_1 с порта MISO.

Код третьего поведенческого блока

```
always@(negedge SCLK or posedge REG_FLAG_START)
```

```
begin
	if(REG_FLAG_START)
	begin
		REG_TRANSMIT_BIT_2<=REG_TRANSMIT_DATA[PACK_LENGTH-1];
	end
	else
	begin
		if(WRITE_MODE==FALLING_EDGE)
		begin
			if(!CPHA)
				REG_TRANSMIT_BIT_2<=REG_TRANSMIT_DATA[REG_BIT_INDEX-1];
			else
				REG_TRANSMIT_BIT_2<=REG_TRANSMIT_DATA[REG_BIT_INDEX];
		end
		else
			REG_RECEIVE_DATA_2[REG_BIT_INDEX]<=MISO;
	end
end
```

Описание третьего поведенческого блока

Работа 3 поведенческого блока аналогична работе второго. Однако, если тот блок детектировал передний фронт сигнала SCLK (posedge SCLK), то текущий блок реагирует на задний фронт SCLK (negedge SCLK). Условие стало проверять на запись по заднему фронту. В остальном их работа схожа. Важно заметить, если CHPA==0, то чтение опережает запись, поэтому ставить нужно следующий бит(то бишь индекс на единицу меньше в силу правила отправки старшим битом вперед), а не текущий, потому как текущий (поставленный при подтяжке CS к логическому нулю) уже прочитан. Это имеет место и во втором always- блоке.

Комбинаторная логика модуля

Код комбинаторной логики модуля

```
parameter RISE_EDGE		=1'b1;//передний фронт
parameter FALLING_EDGE		=1'b0;//задний фронт
parameter WRITE_MODE		=CPOL^CPHA;
parameter READ_MODE		=!WRITE_MODE;

assign MOSI= REG_MOSI_Z_STATE ? 1'bZ: (WRITE_MODE==RISE_EDGE)?
	REG_TRANSMIT_BIT_1:REG_TRANSMIT_BIT_2;

assign OUT_RECEIVE_DATA= (READ_MODE==RISE_EDGE)?
REG_RECEIVE_DATA_1:REG_RECEIVE_DATA_2;
```

Описание комбинаторной логики блока

Комбинаторная логика блока коммутирует регистры, изменяемые и наполняемые во 2 и 3 поведенческих блоках с выходными шинами модуля SPI_FPGA_MASTER и позволяет лаконично записывать условия в тех же

блоках на чтение/запись по конкретному фронту. Законы коммутации определяются состоянием CPOL и CPHA параметров, которые задаются в шапке модуля.

При анализе временных диаграмм в 4 разных состояниях протокола можно составить следующую таблицу.

Таблица 7. Сопоставление процессов считывания и записи фронтам (переднему и заднему) сигнала SCLK в зависимости от параметров CPHA и CPOL.

CPOL	CPHA	Запись	Чтение
0	0	Спад	Фронт
0	1	Фронт	Спад
1	0	Фронт	Спад
1	1	Спад	Фронт

Важно заметить, что считывают и записывают ведущее и подчиненные устройства по одним и тем же фронтам, например, если мастер ставит бит на MOSI по переднему фронту (записывает), то и подчиненные тоже ставят на MISO данные по переднему фронту.

Если ввести обозначения Спад=0 и Фронт=1 (FALLING_EDGE=0, RISE EDGE=1), то мы получаем следующую таблица истинности.

Таблица 7. Сопоставление процессов считывания и записи фронтам (переднему и заднему) сигнала SCLK в зависимости от параметров CPHA и CPOL с подстановкой значений процессов.

CPOL	CPHA	Запись	Чтение
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Можно заметить, что функция «Запись» равняется исключающему или между CPOL и CPHA, а функция «Чтение» является инверсией функции «Запись». Сказанное выше аккумулируется в 4 строчки комбинаторной логики модуля:

```
parameter RISE_EDGE     =1'b1;
parameter FALLING_EDGE  =1'b0;
parameter WRITE_MODE    =CPOL^CPHA;
parameter READ_MODE     =!WRITE_MODE;
```

На данном этапе прояснилась ситуация с логическими условиями в if –

операторах двух поведенческих блоков. Однако, коммутация регистров и выходной шины еще не декомпозирована. Следующие 2 операции assignn коммутируют по одной шине. Первая является двойным тернарным оператором (один вложен в другой), а вторая [операция] представляет собой тернарный оператор в чистом виде.

Рис. 4. Общий вид тернарного оператора.

```
assign MOSI= REG_MOSI_Z_STATE ? 1'bZ: (WRITE_MODE==RISE_EDGE)?
        REG_TRANSMIT_BIT_1:REG_TRANSMIT_BIT_2;
```

«assign» – команда непрерывного присваивания. Слева от знака равно располагается шина MOSI (типа wire), ей присваивается либо Z состояние (если регистр, отвечающий за Z состояние активен) , либо же REG_TRANSMIT_BIT_1, если запись идет по переднему фронту, в ином случае присваивается значение REG_TRANSMIT_BIT_2.

```
assign OUT_RECEIVE_DATA= (READ_MODE==RISE_EDGE)?
REG_RECEIVE_DATA_1:REG_RECEIVE_DATA_2;
```

Шине «OUT_RECEIVE_DATA» присваивается REG_RECEIVE_DATA_1, если чтение происходит по переднему фронте, а в обратном случае присваивается REG_RECEIVE_DATA_2.

Полученные результаты в процессе моделирования модуля MASTER_SPI

Режим CPOL=0, CHPA=0.

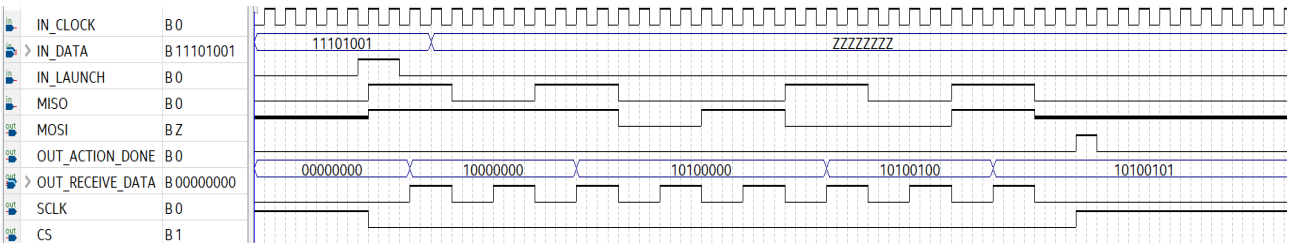

Рис. 5. Режим CPOL=0, CPHA=0. Передача 1 байта. Данные на шине MISO выставлены самостоятельно.

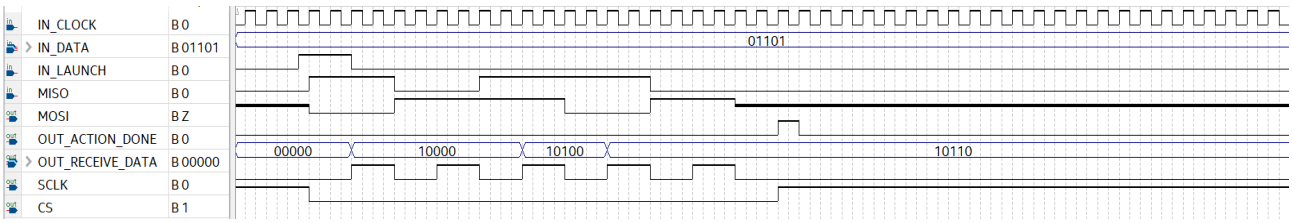

Рис. 6. Режим CPOL=0, CPHA=0. Передача 5 бит. Данные на шине MISO выставлены самостоятельно.

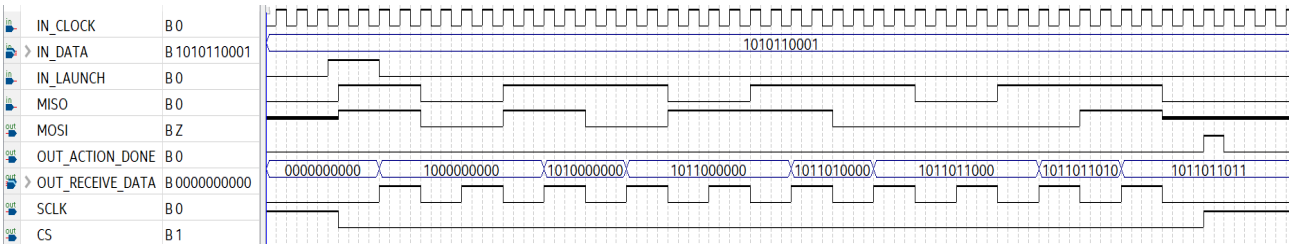

Рис. 7. Режим CPOL=0, CPHA=0. Передача 10 бит. Данные на шине MISO выставлены самостоятельно.

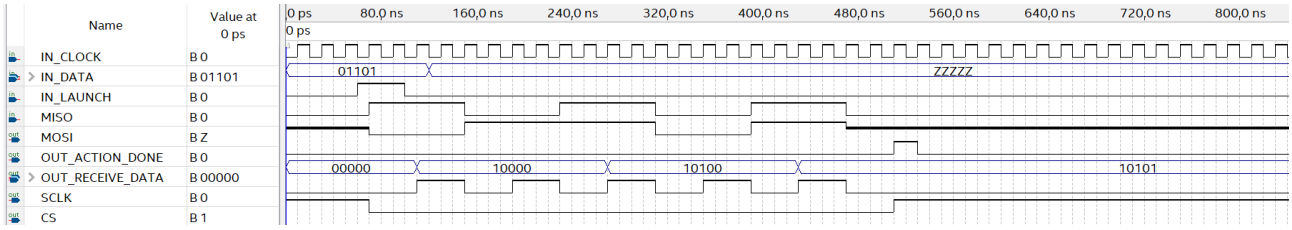

Рис. 8. Режим CPOL=0, CPHA=0. Передача 5 бит с частотой 50 МГц. Данные на шине MISO выставлены самостоятельно.

Режим CPOL=1, CHPA=0.

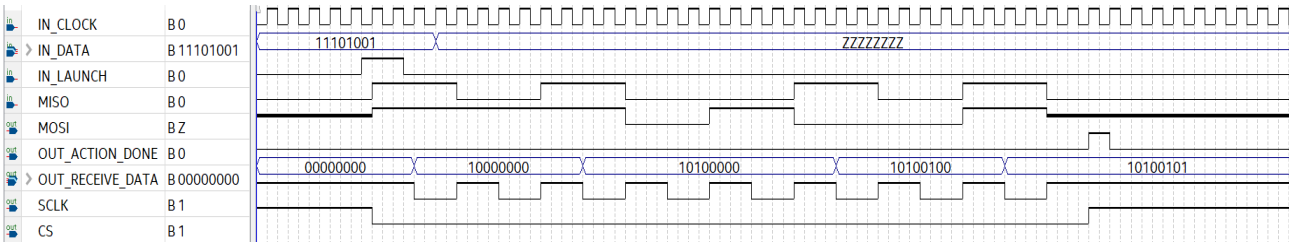

Рис. 9. Режим CPOL=1, CPHA=0. Передача 1 байта. Данные на шине MISO выставлены самостоятельно.

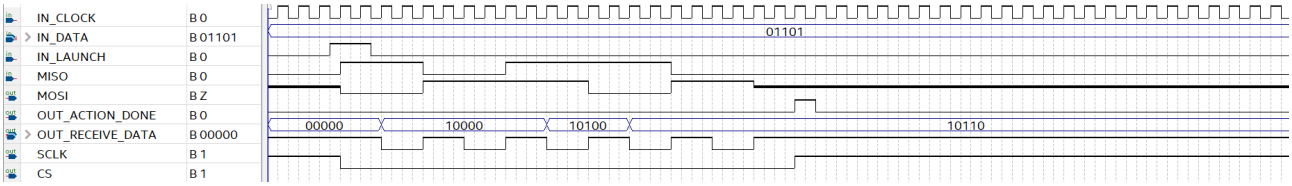

Рис. 10. Режим CPOL=1, CPHA=0. Передача 5 бит. Данные на шине MISO выставлены самостоятельно.

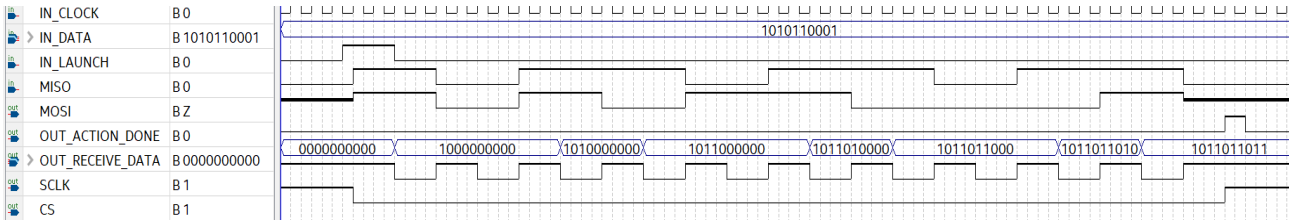

Рис. 11. Режим CPOL=1, CPHA=0. Передача 10 бит. Данные на шине MISO выставлены самостоятельно.

Режим CPOL=0, CHPA=1.

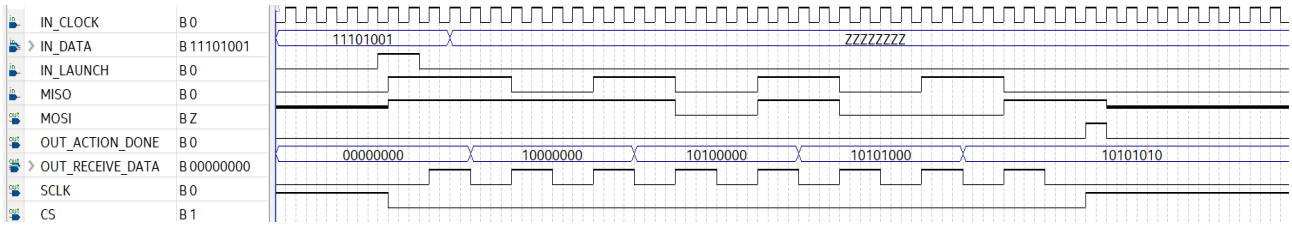

Рис. 12. Режим CPOL=0, CPHA=1. Передача 1 байта. Данные на шине MISO выставлены самостоятельно.

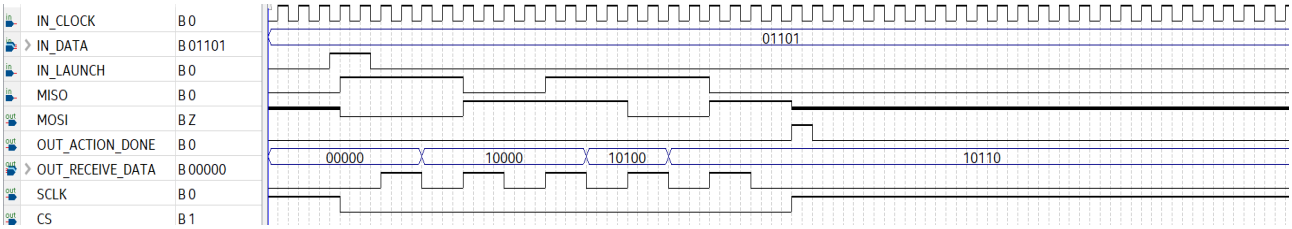

Рис. 13. Режим CPOL=0, CPHA=1. Передача 5 бит. Данные на шине MISO выставлены самостоятельно.

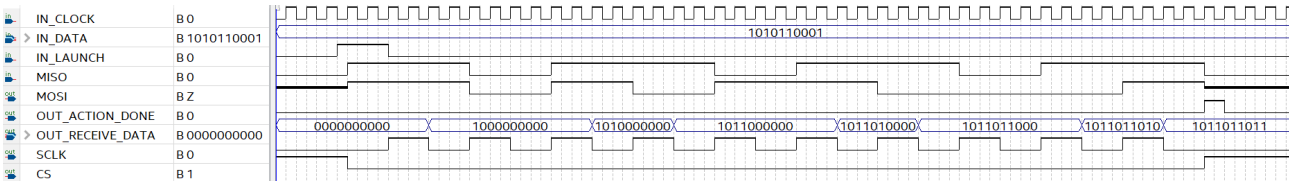

Рис. 14. Режим CPOL=0, CPHA=1. Передача 10 бит. Данные на шине MISO выставлены самостоятельно.

Режим CPOL=1, CHPA=1.

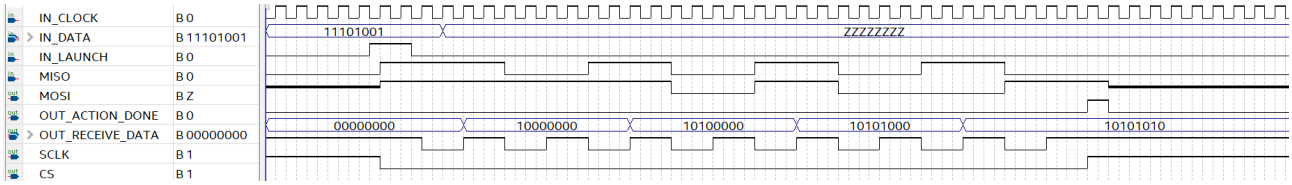

Рис. 15. Режим CPOL=1, CPHA=1. Передача 1 байта. Данные на шине MISO выставлены самостоятельно.

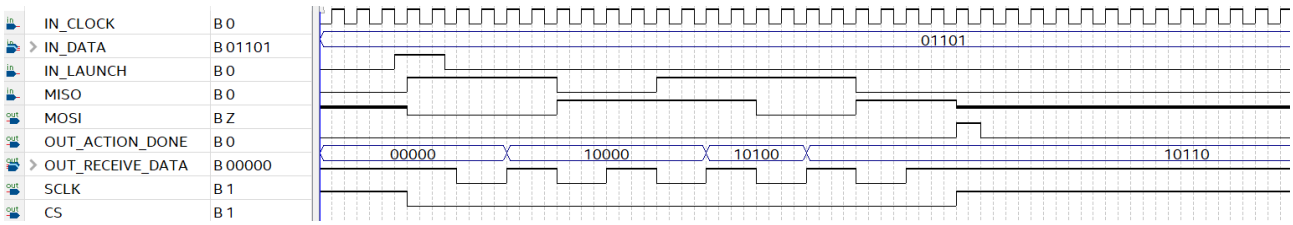

Рис. 16. Режим CPOL=1, CPHA=1. Передача 5 бит. Данные на шине MISO выставлены самостоятельно.

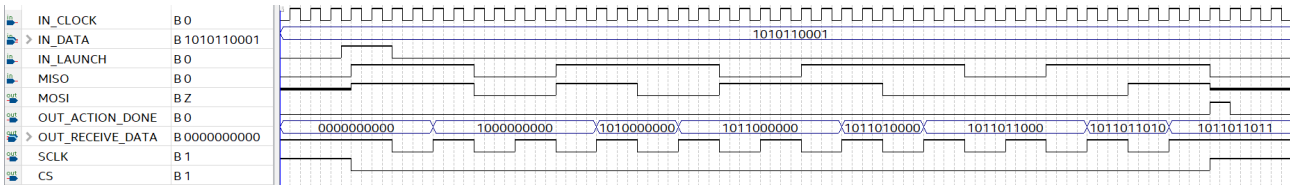

Рис. 17. Режим CPOL=1, CPHA=1. Передача 10 бит. Данные на шине MISO выставлены самостоятельно.

Отчет о компиляции модуля под микросхему ПЛИС Altera MAX2 EPM240T100C5 со следующими параметрами:

```
parameter BIT_PER_SECOND        =    12500000;
parameter CLOCK_FREQUENCY       =    50000000;
parameter CLKS_PER_BIT_LOG_2    =    3;
parameter PACK_LENGTH_LOG_2     =    3;
parameter PACK_LENGTH           =    8;
parameter CPOL                  =    1'b0;
parameter CPHA                  =    1'b0
```

Revision Name	SPI_FPGA_MASTER
Top-level Entity Name	SPI_FPGA_MASTER
Family	MAX II
Device	EPM240T100C5
Timing Models	Final
Total logic elements	48 / 240 (20 %)
Total pins	23 / 80 (29 %)
Total virtual pins	0
UFM blocks	0 / 1 (0 %)

Рис. 18. Отчет о компиляции модуля с указанными выше параметрами в среде Altera Quartus 18.1

Видно, что модуль расходует 48 логических ячеек и требует подключение по 23 пинам. Все сопутствующие материалы в приложении к КП.