HDL-реализация асинхронного приёмопередатчика

Алексей Гребенников (г. Актау, Казахстан)

В статье описана модель асинхронного приёмопередатчика на языке Verilog для связи с отладочной платой SP605, выполненной на базе ПЛИС Xilinx семейства Spartan 6.

Введение

Асинхронный приёмопередатчик (UART) является распространённым устройством для обмена данными. Одним из самых его известных применений является последовательный порт (COM) компьютера, где UART используется для передачи данных по протоколу RS-232. Важным достоинством асинхронного приёмопередатчика является простота реализации аппаратного и программного обеспечения. Недостаток – низкая скорость передачи данных. Несмотря на развитие и внедрение высокоскоростных интерфейсов, таких как USB и Ethernet, протокол передачи данных RS-232 до сих пор очень популярен и востребован.

В данной статье описано построение модели UART на языке Verilog для коммуникации с отладочной платой SP605, выполненной на базе ПЛИС Xilinx семейства Spartan 6. Плата имеет встроенный мост USB-UART, поэтому подключается к компьютеру через USB-порт. Драйвер моста создаёт виртуальный COM-порт (VCP), который позволяет использовать для связи с платой стандартные программы, такие как HyperTerminal.

Стандарт RS-232

Каждый пакет данных в RS-232 передаётся асинхронно и должен содержать в себе следующие компоненты: старт-бит, биты данных (от 5 до 8), бит чётности и стоп-бит. Наиболее распространённой является схема, изображённая на рисунке 1. В этом случае пакет данных состоит из старт-бита, восьми бит данных и стоп-бита. Бит чётности отсутствует. Как видно из рисунка 1, в режиме ожидания на линии присутствует лог. 1. Переход линии в лог. 0 воспринимается приёмником как начало пакета данных. В этот момент запускается счётчик, который отсчитывает шестнадцать периодов своей частоты и запоминает восьмой отсчёт. Если восьмой отсчёт, т.е. средний, равен нулю, значит, получен правильный старт-бит и начинается приём остальных битов пакета данных. Частота внутреннего счётчика определяет скорость передачи данных и может варьироваться в широких пределах. Она должна быть одинаковой для передатчика и приёмника.

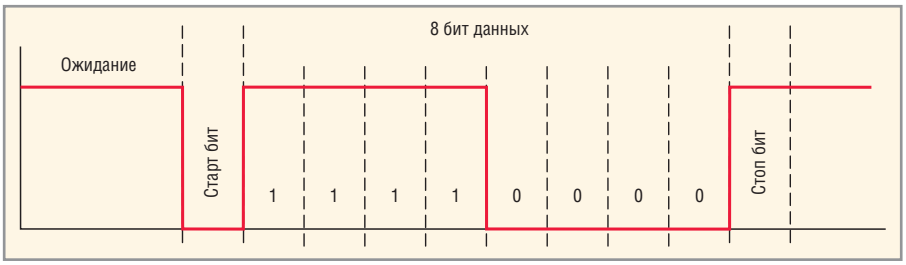

Рис. 1. Формат пакета данных RS-232

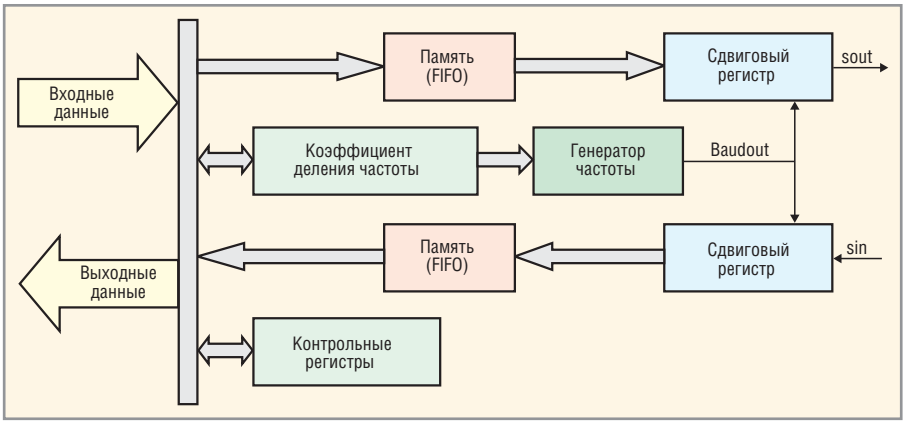

Рис. 2. Структурная схема UART

Структура UART

В данной статье описано построение модели UART на языке Verilog. Исходные файлы находятся в архиве *uart.zip*. Файл верхнего уровня приёмопередатчика – *uart_top.v*. Симуляция проекта проводилась при помощи программы ModelSim V6.4a, компиляция – при помощи пакета программ Plan Ahead V12.3. Файл верхнего уровня проекта – *sopc.v* – включает в себя *uart_top.v*, а также тестовую логику. UART был успешно синтезирован и протестирован для ПЛИС Xilinx XC6SLX45TFGG484-3, которая является частью отладочной платы SP605.

Структурная схема UART приведена на рисунке 2. Генератор частоты управляется двумя восьмибитными регистрами, доступными для чтения и записи. Эта пара регистров образует одно 16-битное число и задаёт коэффициент деления частоты. На выходе генератора формируется частота *baudout*. В цепи передатчика эта частота делится на 16, образуя непосредственно частоту передачи данных. В цепи приёмника каждый бит разбивается на 16 частей, т.е. используется частота *baudout* без дальнейшего деления. Среднее значение, восьмая выборка, считается величиной бита. Приёмник и передатчик имеют память FIFO размером 512 байт. Классический вариант UART типа 16550 имеет память FIFO размером 16 байт. Однако в данном случае для обеспечения связи на более высоких скоростях, чем позволяет 16550, был выбран буфер памяти большего объёма. Это необходимо для совместимости с мостом USB-UART CP2103, являющегося частью платы SP605, который поддерживает скорос-

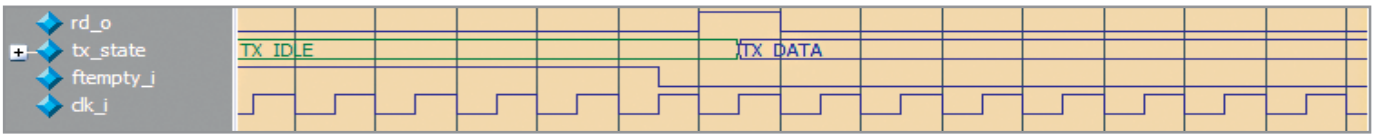

Рис. 3. Фрагмент симуляции процесса передачи

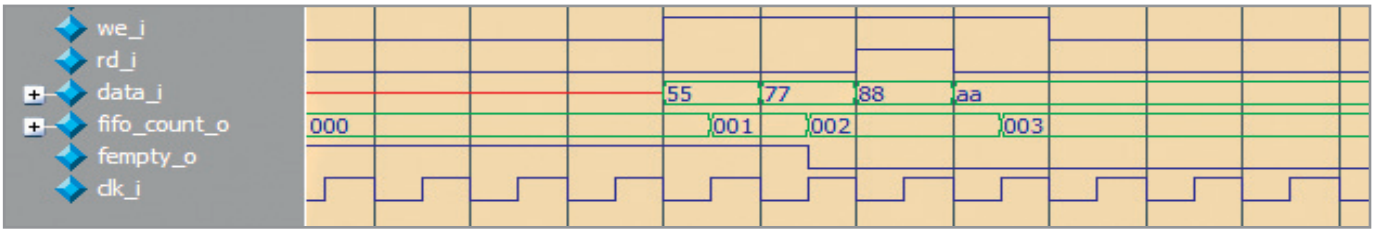

Рис. 4. Работа памяти FIFO

ти до 1 Мбит/с и также имеет буфер памяти 576 байт для приёмника и 640 байт для передатчика. Более подробно структура UART 16550 и моста CP2103 описаны в [1] и [2] соответственно.

Контрольные регистры задают такие параметры, как разрешение или запрещение прерываний, причины прерываний, порог срабатывания прерывания для буфера приёмника и некоторые другие. Все регистры адресуются при помощи трёх адресных линий. Рассмотрим более подробно работу отдельных блоков UART.

Преобразователь частоты

Блок генератора, исходный код которого находится в файле *baud_generator.v*, служит для преобразования основной тактовой частоты UART в частоту приёма и передачи данных. 16-битный регистр *divisor_reg* может быть установлен в любое значение, отличное от нуля. Для конфигурации этого регистра сначала необходимо установить бит 7 контрольного регистра *line_control* в единичное значение (в файле *uart_top.v*); затем записать по адресу 3'b000 младший байт регистра и по адресу 3'b001 – старший байт. Для расчёта частоты передачи данных необходимо поделить основную тактовую частоту на 2 и затем ещё раз – на коэффициент деления. На плате SP605 для тактирования UART использовался сигнал с частотой 27 МГц. При первоначальном включении регистр *divisor_reg* устанавливается в значение 16'h0058 (88 в десятичной системе), т.е. получаем выходную частоту *baudout* 27 000 000/(2*88) = 153 409 Гц. Частота передачи данных при этом будет равна *baudout*/16 = 9588 Гц. Отклонение от стандартной частоты передачи данных 9600 Гц составляет 0,1%, что вполне допустимо.

Блок передачи данных

Исходный код блока передачи данных находится в файле *tx_cntr.v*. Заметим, что для передачи и приёма используется одна и та же модель памяти FIFO, расположенная в файле *fifo.v*. В файле верхнего уровня *uart_top.v* эти два буфера включены под названиями *tx_fifo_top* и *rx_fifo_top* – для передатчика и приёмника соответственно.

Основой блока передачи данных является конечный автомат, который может находиться в двух состояниях: режим ожидания (*TX_IDLE*) и режим передачи данных (*TX_DATA*). Конечный автомат отслеживает состояние памяти FIFO при помощи сигнала *ftempty_i*. До тех пор, пока этот сигнал находится в лог. 1, память передатчика пуста. При записи одного или более байт данных в эту память сигнал *ftempty_i* переходит в лог. 0. При этом конечный автомат переходит в состояние *TX_DATA* и генерирует строб чтения слова *rd_o*, т.е. считывает один байт из буферной памяти и начинает передачу данных. После передачи одного байта данных автомат возвращается в состояние ожидания и, если в памяти ещё остались данные для передачи, сразу же переходит в состояние *TX_DATA*. Этот циклический процесс продолжается до тех пор, пока в буферной памяти передатчика остаётся хотя бы один байт. Фрагмент симуляции процесса передачи изображён на рисунке 3.

Как упоминалось выше, для передачи данных используется частота *baudout*, поделённая на 16. Этот сигнал называется *baud_16*. Для подсчёта переданных битов данных служит счётчик *bus_counter*, который равен нулю при нахождении конечного автомата в состоянии ожидания и увеличивает своё значение на единицу при каждом положительном фронте сигнала *baud_16*, если конечный автомат находится в состоянии передачи данных. Счётчик также обнуляется при достижении значения 9, что необходимо для избегания дополнительных тактов ожидания при непрерывной передаче нескольких байтов. Логика работы счётчика на языке Verilog выглядит следующим образом:

```
if ((tx_state == `TX_IDLE) ||
(bus_counter == 4'h9))
bus_counter <= 4'h0;
else
bus_counter <= bus_counter +
1'b1;
```

Как видно из рисунка 3, при переходе конечного автомата из состояния *TX_IDLE* в *TX_DATA* генерируется строб чтения памяти *FIFO rd_o*. После этого прочитанный байт загружается в сдвиговый регистр *tx_shiftreg*. Этот регистр относится к типу с параллельным входом и последовательным выходом. Однако до начала сдвига в регистре, т.е. передачи полезных данных, необходимо передать старт-бит. Это происходит на отрицательном фронте сигнала *baud_16* при значении счётчика bus_counter = 4'h1. В течение последующих восьми отрицательных фронтов сигнала сдвиговый регистр сдвигается на один бит за каждый цикл, таким образом, формируется последовательная цепочка на выходе *sout_o*, как это видно из фрагмента программы:

```
case (bus_counter)
4'h2,4'h3,4'h4,4'h5,4'h6,4'h7,4'h
8,4'h9:
begin
sout_o <= tx_shiftreg[0];
tx_shiftreg <= (tx_shiftreg >>
1'b1);
end
```

Блок приёма данных

Исходный код блока приёма данных находится в файле *rx_cntr.v*. Основой блока приёма также является конечный автомат с двумя состояниями – режим ожидания (*RX_IDLE*) и режим передачи данных (*RX_DATA*). Приём данных начинается после фиксирования старт-бита при условии, что память FIFO приёмника не полностью заполнена. Если необходимо принять байт, но в буферной памяти нет места, приёмник выдаёт сигнал о переполнении (*rx_overrun*) и остаётся в режиме ожидания. Фрагмент программы перехода автомата из состояния ожидания в состояние приёма данных выглядит следующим образом:

```
case (rx_state)
`RX_IDLE:
begin
if (start_rx && ~ffull_i)
rx_next_state = `RX_DATA;
else if (start_rx && ffull_i)
rx_overrun = 1'b1;
end
```

Для приёма данных используется частота *baudout*, которая синтезируется генератором частоты, без деления на 16. В состоянии ожидания линия *sin_i* находится в лог. 1. При фиксировании лог. 0 на линии *sin_i* приёмник начинает считать до восьми на частоте *baudout* (в модуле *rx_cntr.v* эта частота называется *baud_clk_i*). Если восьмой отсчёт равен нулю, сигнал *start_rx* становится активным, и конечный автомат переходит в состояние приёма данных. Ещё через восемь отсчётов фиксируется начало первого бита, и затем запоминается каждый средний отсчёт, т.е. восьмой из шестнадцати. Последовательные отсчёты сдвигаются в регистр *rx_shiftreg*, который является сдвиговым регистром с последовательным входом и параллельным выходом:

```
if ((rx_state == `RX_DATA) &&
(baud_counter == 4'h8))
rx_shiftreg[7:0] <=
{sin_i,rx_shiftreg[7:1]};
```

После приёма восьми полезных битов сигнала и проверки стоп-бита полученный байт записывается в память. При отсутствии стоп-бита регистрируется ошибка приёма байта сигналом *frame_err_o*.

Буфер памяти

Приёмник и передатчик используют одинаковый модуль памяти, исходный код которого содержится в файле *fifo.v*. Это память типа FIFO объёмом 512 байт. После инициализации память пуста, адрес для записи *adrw* равен адресу для чтения *adrr* и равен нулю. Счётчик байтов в памяти *fifo_count* также равен нулю. В рабочем состоянии сигнал *we_i* записывает данные в память, увеличивая счётчик байтов и адрес для записи, а сигнал *rd_i* – считывает данные, уменьшая счётчик байтов и увеличивая адрес чтения. Если при чтении адрес *adrr* становится равным адресу *adrw*, значит, память пуста:

```
else if (rd_i)
begin
data_o <= fifo[adrr];
adrr <= adrr + 1'b1;
if ((adrr + 1'b1 === adrw))
fifo_empty <= 1'b1;
end
```

Аналогично, если при записи адрес записи *adrw* становится равным адресу чтения *adrr*, значит, память полностью заполнена. Фрагмент цикла чтения/записи в память передатчика изображён на рисунке 4; показана запись 4 байт, при записи третьего байта одновременно происходит чтение, поэтому в этот момент счётчик числа байтов в памяти не увеличивает своё значение.

Заключение

Рассмотренный в статье приёмопередатчик был успешно испытан на отладочной плате SP605 для ПЛИС Xilinx семейства Spartan 6. Со стороны компьютера использовалась стандартная программа связи HyperTerminal.

Описанная модель UART является учебным материалом, демонстрирующим принципы действия устройства. Архив *uart.zip* с упомянутыми в статье файлами программ находится в электронном приложении к статье на сайте журнала.

Литература

1. PC16550D Universal Acynchronous Receiver/Transmitter with FIFOs. National Semiconductor.
2. CP2103 Single chip USB to UART Bridge. Silicon Laboratories.