Как проектировать схемы с шинными структурами

Урок 9

Знакомство с пакетом OrCad 9.1

В сложной радиоэлектронной аппаратуре, и в частности в вычислительных системах, широко используется блочный принцип организации структур из отдельных унифицированных модулей (блоков, устройств), связанных воедино общей шиной. Шина представляет собой совокупность линий (проводников), общих для всех подключенных к ней устройств, и служит для обмена данными.

Александр Шалагинов, к. т. н.

shalag@vt.cs.nstu.ru

При использовании шинной структуры необходимо иметь соглашение о том, кому разрешено передавать данные по шине. Понятно, что в каждый момент времени передавать данные может только одно устройство, а остальные (или одно из них) могут только принимать. Мы не станем касаться вопросов бесконфликтного управления шиной, предполагая, что они решены.

Для возбуждения шины нельзя использовать вентили с активным выходом, поскольку они всегда выдают один из двух логических уровней: 0 или 1. Если такие выходы подключить к проводнику шины, то они станут мешать друг другу и возникнет неразбериха. Для этого случая необходимы вентили, выходы которых можно отключать от шины, например, схемы с открытым коллектором или вентили с тремя состояниями. При работе на внутреннюю шину используется также мультиплексирование или логическое объединение выходов.

Некоторые устройства, например, центральный процессор, диск или оперативная память, могут в одни моменты времени быть источником данных (передатчиком), а в другие моменты — их приемником. Следовательно, шина данных и вентили, работающие на нее, должны быть двунаправленными.

Обычно к шине подключено много устройств и, следовательно, она создает значительную нагрузку на выход передатчика. По этой причине между выходом устройства и шиной приходится включать буферный усилитель. Часто все перечисленные функции совмещаются в одном элементе, который называется шинным драйвером.

Итак, создавая схемы с шинной организацией обмена данных, надо прежде всего уметь рисовать шины и описывать действующие на них шинные сигналы, а уж затем проектировать (или использовать в готовом виде) элементы, работающие на шину, такие, как схемы с открытым коллектором, вентили с тремя состояниями и двунаправленные шинные драйверы.

Как же это делается на практике? Запустим графический редактор OrCAD Capture и нарисуем небольшой сегмент шины (команда **Place/Bus**). Присвоим ей имя, например, A[3..0]. Выделим все объекты схемы (Ctrl+A) и вызовем редактор свойств Property Editor (Ctrl+E). На закладке Schematic Nets вы увидите имена A0, A1, A2 и A3. *Цепей с такими именами на схеме еще нет, но редактор уже знает об их существовании.* Удобно считать, что они «спрятаны» в самой шине.

Создадим VHDL-список цепей и посмотрим, какие сигналы объявлены в нашей схеме. В разделе SIGNALS мы увидим всего одну строчку:

```
SIGNAL A : std_logic_vector (3 DOWNTO 0);
```

Содержимое данной строки означает, что сигнал A объявлен как четырехразрядный одномерный массив с убывающим диапазоном (3 DOWNTO 0). Элементы массива A(3), A(2), A(1) и A(0) соответствуют отдельным сигналам шины с названиями A[3], A[2], A[1] и A[0]. Эти сигналы действуют на проводниках A3, A2, A1 и A0, «спрятанных» в шине.

В том, что это действительно так, легко убедиться, отправив нашу схему на моделирование. Откроем в программе OrCAD Simulate диалоговую панель Interactive Stimulus и щелкнем на кнопке **Browse…**. На панели Browse Signals (в нижнем окне) вы обнаружите пять сигналов — шинный сигнал A и четыре одиночных сигнала A[3], A[2], A[1] и A[0]. Пока это все, что есть в нашей схеме.

Вернемся в окно графического редактора Capture и дорисуем схему так, как показано на рис. 1. Обра-

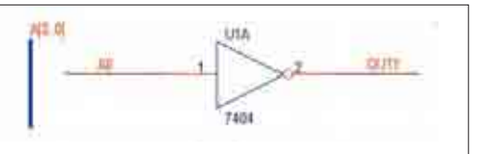


Рис. 1. Простая схема с шинной структурой

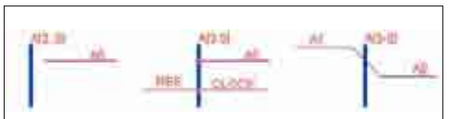
тите внимание, к входу инвертора U1A подключен проводник с именем A0, которое уже известно редактору. Именно так называется один из проводников, составляющих шину A[3..0].

Однако нет ли ошибки в том, что проводник A0 не соприкасается с шиной A[3..0]? Ответ на этот вопрос надежнее всего получить экспериментом. Промоделируем схему, описав в качестве внешнего воздействия только шинный сигнал A. Сигнал A[0], действующий непосредственно на проводнике A0, программировать не станем! Тем не менее, результаты моделирования показывают, что инвертор получил нужный сигнал (рис. 2) — ведь его выход переключается.

Рис. 2. Результаты моделирования инвертора, получающего входной сигнал с шины A[3..0]

Отсюда следует вывод, что шина и проводник *связаны общим именем*, так что совсем не обязательно добиваться их визуального (физического) контакта. Впрочем, подобное поведение редактора Capture можно было предсказать, если вспомнить (см. урок 2), что проводники с одинаковыми именами (алиасами) объединяются им в одну цепь. В данной ситуации разница лишь в том, что проводник цепи виден на схеме, а проводник шины «спрятан» в ней.

На рис. 3, кроме уже рассмотренного варианта, показаны еще два других способа «подключения» проводника к шине. Взятое в кавычки слово «подключение» говорит о том, что никакие манипуляции с графическими образами проводника и шины (рис. 3, б и в) в действительности не создают электрического контакта.



а) б) в)

Рис. 3. Возможные варианты «подключения» проводника к шине

На рис. 3, б в точке их соприкасания даже стоит junction-соединитель. Потяните за проводник и вы увидите, что он не отрывается от шины. Создается полная иллюзия, что соединение существует физически. Однако это не так. Посмотрите на проводник CLOCK. Графически он тоже связан с шиной, но такого проводника в ней точно нет, а следовательно, не может быть и электрической связи.

Мы уже знаем, что шина — это совокупность составляющих ее проводников. Так вот, проводники разрешается непосредственно «заводить» в шину, как это показано на рис. 3, б. Но лучше использовать для этих целей специальные графические объекты, называемые входами в шину (Bus Entry).

Они вызываются командой **Place/Bus Entry** (Shift+E) или щелчком на пиктограмме, показанной рядом .

И дело даже не в том, что чертеж приобретает более привлекательный вид (рис. 3, в). Входы в шину допускают соприкосновение друг с другом без образования электрического контакта, тогда как проводники в подобной ситуации сливаются в одну цепь, создавая дополнительные проблемы при их размещении. На рис. 3, б RES и CLOCK — это не две разных цепи, а одна цепь с двумя алиасными именами RES и CLOCK.

Рассмотрим более серьезный пример с использованием шин для передачи данных от одного объекта к другому. На рис. 4 для этих целей используется счетчик 74193 и регистр 74194 из библиотеки ttl.olb. Передачу данных с выходов счетчика на входы параллельной загрузки регистра осуществим с помощью шины bus_Q[3:0], а начальную загрузку счетчика произведем, используя четырехразрядную шину bus_D[3..0].

Командой **Place/Bus** (Shift+B) проведем две шины, так чтобы они не касались ни одного контакта. Названную команду можно также инициализировать, щелкнув по иконке из палитры инструментов Tool Palette. Присвоим шинам имена bus_D[3..0] и bus_Q[3:0]. Это делается точно так же, как и для обычных проводников (командой **Place/Net Alias…**). Разница заключается лишь в том, что в имени шины должен быть указан диапазон, определяющий ее разрядность (ширину шины) и список проводников, входящих в нее.

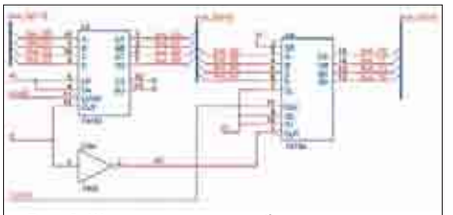


Рис. 4. Шинная организация обмена данными

При рисовании шин с большим числом проводников (например, 16- или 32-разрядных шин) можно ускорить процесс их создания, если воспользоваться услугами команды **Edit/Repeat Place** (или **Edit/Repeat Copy**). Эти команды «висят» на горячей клавише F4 и поэтому исполняются очень быстро.

Поместите на схеме первый вход в шину и нажмите клавишу Esc. Созданный объект должен сохранить выделение. А теперь жмите F4 столько раз, какова разрядность шины. Точно так же вы можете размещать и проводники шины. Легко и быстро, не правда ли?

Чуть сложнее заводить проводники в горизонтально расположенную шину, потому что заданные по умолчанию установки (шаг и направление перемещения) вам не подходят. В этой ситуации надо поступить так. Поместив на схему первый объект, например, проводник, надо нажать клавишу Ctrl и, удерживая ее в этом состоянии, отбуксировать копию размещаемого объекта в нужном направлении и на требуемое расстояние. Редактор запомнит ваши действия, и повторное копирование по F4 будет таким, как вам надо.

Добавлю, что работу по созданию шинных структур можно еще немного усовершенствовать, если перед копированием проводников первому из них присвоить алиасное имя. Тогда все последующие проводники будут автоматически получать пользовательские имена в возрастающем порядке.

Вооружившись полученными знаниями, проведем от шины bus_D[3..0] к входам параллельной загрузки счетчика проводники bus_D0, bus_D1, bus_D2 и bus_D3. Они выходят из шины и потому должны называться так же, как и проводники, составляющие шину. Аналогичным приемом подведем ко второй шине проводники bus_Q0, bus_Q1, bus_Q2 и bus_Q3, затем в другом месте (с правой стороны) выведем их из шины и соединим со входами параллельной загрузки регистра (рис. 4).

Рис. 5. Результаты моделирования шинной структуры

Выходы регистра также соберем в шину bus_Y[3-0]. Вероятно, вы уже обратили внимание, что, задавая ширину шины, мы использовали разные разделители: bus_D[3..0], bus_Q[3:0] и bus_Y[3-0]. Это делалось с единственной целью — познакомиться с возможными форматами описания шинных имен.

Сохраним свой проект в файле bus.opj и промоделируем его (рис. 5), акцентируя внимание на передаче сигналов по шинам bus_D[3..0], bus_Q[3:0] и bus_Y[3-0]. Другие режимы работы счетчика и регистра, например, счет и сдвиг, в данном эксперименте не рассматриваются.

Мы видим (рис. 5), что перед началом работы оба узла устанавливаются в ноль. Это происходит на отметке 100 ns, когда на вход R подается высокий уровень (положение визирной линейки). Затем, с появлением низкого уровня на входе LOAD, в счетчик загружаются данные с шины bus_D (положение первого маркера, 500 ns). И, наконец, по фронту сигнала CLOCK текущее содержимое счетчика (код 8) передается в регистр (положение второго маркера, 700 ns).

Назовем несколько простых правил, о которых следует помнить при проектировании схем с шинной организацией.

Самая первая операция, которую надо выполнить после того, как нарисована шина — определить ей шинное имя. Оно задается в формате:

basename[x..y]

Первая часть имени basename несет смысловую нагрузку, вторая — [x..y] определяет ее диапазон. Диапазон или ширина шины — это число проводников в ней. Его легко подсчитать по формуле: x – y + 1 (для убывающего диапазона).

а) б)

Рис. 6. Прямое подключение шин к иерархическим блокам

Между базовым именем и диапазоном можно включить один или несколько пробелов. Редактор OrCAD Capture игнорирует их и воспринимает такие модификации как одно имя. Например, шинные имена:

Data[7..0]	(пробела нет)
Data [7:0]	(один пробел)
DATA [7-0]	(два пробела)

описывают одну и ту же шину. Заодно в нижней строке показано, что имя нечувствительно и к регистру. Сказанное касается списка цепей. Как графические объекты они имеют разные коды идентификации.

Имена шин, не соответствующие описанному выше формату, редактор OrCAD Capture просто-напросто выбраковывает, отказываясь размещать их на схеме.

Сказанное, правда, не касается ошибок, сделанных в описании самого базового имени. Например, если имя начинается или (что еще хуже) заканчивается цифрой, то редактор «пропускает» такие имена, однако позднее могут появиться проблемы. Чаще всего это происходит во время автоматической генерации списка цепей (netlist).

Не рекомендуется также проставлять незначащие нули слева при задании границ диапазона, например, Data[00..16]. И совсем недопустимо указывать незначащие нули в именах членов шины. Например, будет ошибкой, если вместо имени Data6 вы укажете имя Data06. САПР OrCAD воспринимает их как совершенно разные имена.

Диапазон шины может быть убывающим [7..0] или возрастающим [0..7]. Однако в любом случае старшим будет крайний левый разряд. Мы уже знаем, что в качестве разделителя границ диапазона используется двоеточие «:», дефис «-» или две точки, идущие подряд «..».

Диапазон определяет не только ширину шины, но и имена всех проводников и сигналов, включенных в нее. Они называются членами (members) шины. Имя члена шины образуется добавлением к базовому имени числа, входящего в диапазон шины. Например, шина Data [0..7] содержит следующие проводники Data0, Data1, …, Data7 и действующие на них сигналы: Data[0], Data[1], …, Data[7].

Особенно эффективно применение шин в схемах с иерархическими блоками (рис. 6). В таких блоках допускается объявлять не только одиночные контакты (Scalar), но и так называемые многоразрядные (шинные) контакты (Bus). Их имена описываются в тех же соглашениях, что и имена шин.

К многоразрядным контактам шина подключается так же просто, как к одиночному контакту проводник. Имена шин и иерархических контактов могут совпадать (входная шина на рис. 6) или отличаться (выходная шина там же). Могут не совпадать даже диапазоны. Например, к контакту A[1..0] разрешается подключить шину B[7..0]. Правда, в этом случае надо помнить, что передача сигналов с шины на многоразрядный контакт будет осуществляться позиционным (поразрядным) способом:

$$B[7] \rightarrow A[1], B[6] \rightarrow A[0].$$

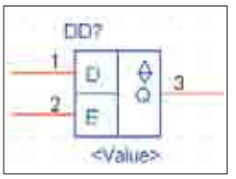
Заканчивая разговор о шинных контактах, заметим, что они будут работать только с VHDL-форматом списка цепей. Никакой другой формат их не понимает.

В принципе можно соединить на схеме и две шины разной ширины. Передача данных с одной шины на другую будет осуществляться по только что описанному правилу. Объединенная шина получит в наследство оба псевдонима, однако рабочим будет имя, доставшееся от шины с большей шириной.

Добавлю, что список цепей, сгенерированный по такой схеме, будет составлен некорректно и его придется редактировать вручную.

Создание компонентов, имеющих выходы с тремя состояниями, весьма актуально для схем с шинной организацией. Поэтому мы рассмотрим этот процесс более подробно. Предположим, нам требуется спроектировать тристабильный буфер, выход которого Q переводится в Z-состояние низким уровнем на разрешающем входе E.

Откроем библиотеку my_lib.olb и создадим графическое изображение этого элемента (показан справа). Назовем его TBUF1 и напишем для него VHDL-модель (рис. 7).




```
library ieee;
use ieee.std_logic_1164.all;

entity TBUF1 is
port (D, E : in std_logic; Q : out std_logic );
end TBUF1;

architecture model of TBUF1 is
begin
   process (D, E)
   begin
      if E = '1' then Q <= D;
      else Q <= 'Z';
   end if;
   end process;
end model;
```

Рис. 7. Текст VHDL-модели тристабильного буфера TBUF1

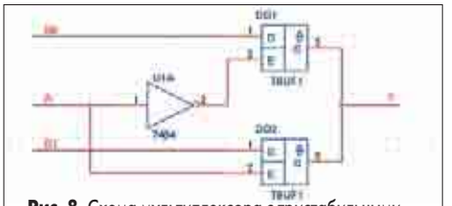


Рис. 8. Схема мультиплексора с тристабильными буферными элементами TBUF1

Для верификации разработанного элемента построим схему мультиплексора mux2_tri (рис. 8), в которой используется тристабильный элемент TBUF1, и промоделируем ее.

Результаты верификации (рис. 9) показывают, что разработанный элемент правильно выполняет свою функцию. Когда на адресном входе A присутствует низкий уровень, данные на выход Y передаются с входа D0. Тристабильный буфер DD2 находится в это время в Z-состоянии и не мешает работать элементу DD1. При A = 1 все наоборот: элемент DD1 переведен в Z-состояние, а на выход поступают данные с входа D1 через тристабильный буфер DD2.

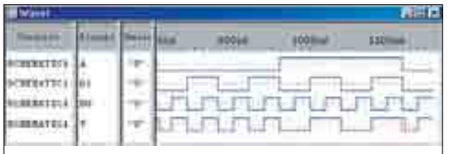
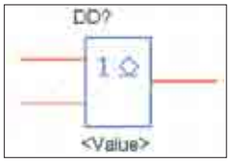


Рис. 9. Результаты верификации мультиплексора, построенного на тристабильных элементах

Для элемента *с открытым коллектором* проделаем аналогичную работу. В качестве объекта исследования спроектируем компонент OR2_OC, выполняющий функцию 2ИЛИ. Сначала нарисуем его графический образ (справа). Для выходного контакта укажем тип — Open Collector. Затем создадим его VHDL-модель.



Отмечу, что разработчики пакета OrCAD по неизвестным мне причинам не стали «закладывать» в модель особенности работы элемента с открытым коллектором. Сравните модели элементов 7400 (155LA3) и 7401 (155LA8). Первый элемент имеет активный выход, второй — выход с открытым коллектором.

Между тем, их VHDL-модели отличаются только задержкой (22 ns против 45 ns). А ведь при отсутствии «подтягивающего» резистора (pull-up resistor) на выходе выключенного элемента фактически образуется Z-состояние или, если говорить с натяжкой, — слабая единица.

В языке VHDL для сигналов типа std_logic есть соответствующие значения 'Z' и 'H'. Мы воспользуемся этой возможностью и построим более правдоподобную модель элемента с открытым коллектором (рис. 10).

Логический ноль представляется здесь полноценным сигналом низкого уровня (OUT1<='0'), логическая единица — только обозначена (OUT1<='H'). Это намек на то, что разработчик схемы должен побеспокоиться о том, чтобы обеспечить на выходе полноценный высокий уровень, подключив к нему «подтягивающий» резистор R_pullup (рис. 11).

```
library ieee;
use ieee.std_logic_1164.all;

entity OR2_OC is
port (IN1, IN2 : in std_logic; OUT1 : out std_logic );
end OR2_OC;

architecture model of OR2_OC is
begin
   process (IN1, IN2)
   begin
      if (IN1 or IN2) = '0' then OUT1 <= '0';
      elsif (IN1 or IN2) = '1' then OUT1 <= 'H';
   else OUT1 <= 'X';
   end if;
 end process;
 end model;
```

Рис. 10. Текст VHDL-модели логического элемента 2ИЛИ с открытым коллектором (в модели он имеет название OR2_OC)

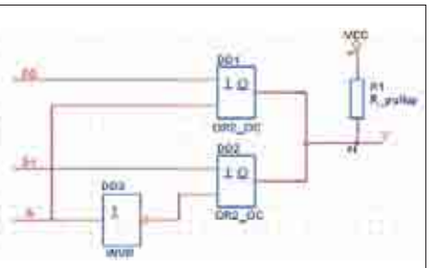


Рис. 11. Мультиплексор mux2_oc, выполненный на элементах 2ИЛИ с открытым коллектором

Для того чтобы схема мультиплексора моделировалась корректно, придется еще немного потрудиться, чтобы реализовать модель «подтягивающего» резистора. Ее VHDL-код показан на рис. 12.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity R_pullup is
      port (R1 : in std_logic; R2 : inout std_logic);
end R_pullup;

architecture model of R_pullup is
begin
   process (R2)
   variable int_R2 : std_logic:='Z';
   begin
      if (R2 = 'H') then int_R2 := '1';
      elsif (R2 = '0') or (R2 = 'X') then int_R2 := 'Z';
      end if;
      R2 <= int_R2 after 0ns;
   end process;
end model;
```

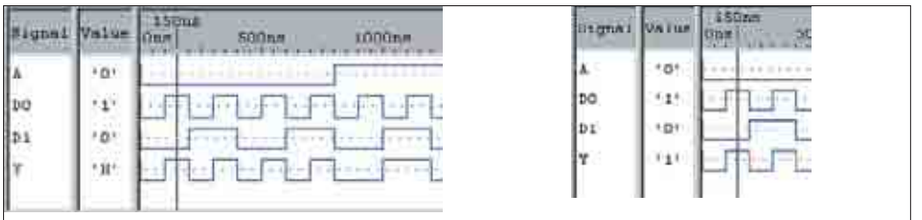
Рис. 12. Текст VHDL-модели «подтягивающего» резистора R_pullup

Что должна делать эта модель? Прежде всего, выяснить состояние выхода открытого коллектора. Если там появилась «слабая единица», то преобразовать ее в полноценный высокий уровень:

```
if (R2='H') then int_R2:='1';
```

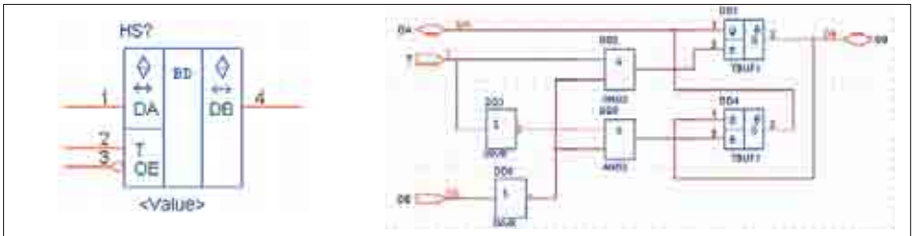
Если на выходе элемента обнаружен полноценный уровень логического нуля, то не мешать ему, переведя «выход» резистора в состояние высокого импеданса:

```
elsif (R2='0') or (R2='X') then int_R2:='Z';
```



а) б)

Рис. 13. Результаты моделирования мультиплексора mux2_oc, построенного на элементах с открытым коллектором без «подтягивающего» резистора а) и с ним б)



а) б)

Рис. 14. Двунаправленный шинный драйвер и его схема замещения

Так как модель резистора должна принимать данные и исправлять их, используя одну и ту же точку (нижний вывод R2 резистора), то этот контакт следует сделать двунаправленным:

```
R2 : inout std_logic;
```

Верхний вывод резистора R1 в функциональном смысле вообще не используется. Конечно, такая модель не выглядит особенно убедительной, но в защиту ее скажу, что нечто похожее можно обнаружить и в фирменной документации по пакету OrCAD.

Результаты моделирования показывают (рис. 13), что построенная модель имеет право на жизнь: без «подтягивающего» резистора на выходе выключенного элемента присутствует слабая единица **'H'** (рис. 13, а). «Подтягивающий» резистор исправляет положение — теперь там полноценный высокий уровень (рис. 13, б).

Последний элемент, без которого не обойтись при проектировании шинных структур, носит название двунаправленный шинный драйвер. Мы реализуем его в двух вариантах. Сначала построим иерархический символ, поддерживаемый схемой замещения, протестируем его, а затем напишем для него VHDL-модель.

Создавая графическое описание двунаправленного шинного драйвера (рис. 14, а), надо не «промахнуться» с указанием типа для выводов, которые могут быть либо входами, либо выходами. Это контакты DA и DB. Им следует задать тип — bidirectional (двунаправленный).

Затем нарисуем схему замещения проектируемого драйвера (рис. 14, б). Здесь тоже не забудем про двунаправленные порты DA и DB. Они называются соответственно PORTBOTH-R и PORTBOTH-L.

Чтобы убедиться, что разработанный шинный драйвер правильно выполняет свои функции, необходимо промоделировать его. Задача осложняется тем, что, создавая схему верификации символа BD_schema, придется позаботиться и о его «обвязке». Действительно, когда данные передаются через шинный драйвер в направлении «слева направо», то есть от входа DA к выходу DB, то их можно подать непосредственно на контакт DA.

Однако при передаче данных в обратном направлении, когда контакт DA станет выходом, ранее подаваемые в эту точку данные станут мешать работе. Возникнет конфликт, который можно разрешить, если отключить входные данные от контакта DA. Эту функцию проще всего выполнить с помощью тристабильного буфера TBUF1, переведя его выход в Z-состояние (DD1 на рис. 15).

Понятно, что такую же «обвязку» придется создать и по другую сторону шинного драйвера (со стороны контакта DB).

Прокомментируем в нескольких словах результаты верификации двунаправленного шинного драйвера (рис. 16). В начале моделирования на входе OE (Output Enable) установлен низкий уровень, который разрешает работу шинному драйверу. Высокий уровень на входе T определяет направление передачи данных: слева направо. Этот же сигнал разрешает работу тристабильному буферу DD1, а его инверсия (сигнал NOT_T на выходе DD3) переводит выход буфера DD2 в Z-состояние. Таким образом, обеспечивается передача данных по цепочке: вход DA — буфер DD1 — шинный драйвер HS1 на выходную цепь DB_INT.

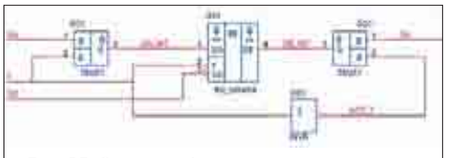


Рис. 15. Схема верификации двунаправленного шинного драйвера

В момент времени 750 ns сигнал OE переключается на высокий уровень и переводит шинный драйвер в Z-состояние. Однако на линию DA_INT продолжается передача данных, так как буфер DD1 остается открытым.

В момент времени 1000 ns меняется направление передачи данных. Теперь они следуют по цепочке справа налево: вход DB — буфер DD2 — шинный драйвер HS1 на выходную линию DA_INT. Но шинный драйвер еще выключен, и сигнал DB проходит только до линии DB_INT.

Наконец, в момент времени 1250 ns управляющий сигнал OE переключается на низкий уровень, открывается шинный драйвер, и данные поступают на линию DA_INT, которая в этом режиме является выходной.

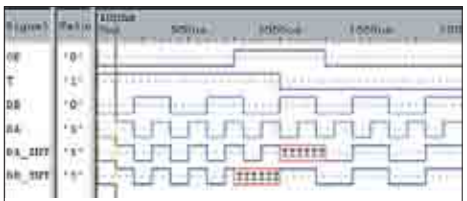


Рис. 16. Результаты верификации двунаправленного шинного драйвера

Нам осталось сделать совсем немного — написать VHDL-модель двунаправленного шинного драйвера. Эта работа не потребует значительных усилий, тем более что уже имеется опыт создания тристабильного буфера. Главное, не забыть про двунаправленный тип выводов DA и DB.

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity BD_schema is
port (
      DA : inout std_logic;
      T : in std_logic;
      OE : in std_logic;
      DB : inout std_logic);
end BD_schema;

architecture model of BD_schema is
begin
      process (T, OE, DA, DB)
      begin
            if (OE = '0') and (T = '1') then DB <= DA after 10ns;
            -- DA - input
            elsif (OE = '0') and (T = '0') then DA <= DB; -- DB - input
            else DA <= 'Z'; DB <= 'Z';
            end if;
      end process;
end model;
```

Рис. 17. VHDL-модель двунаправленного шинного драйвера

На рис. 17 показан один из возможных вариантов ее решения. Отключите от иерархического символа схему замещения, подключите VHDL-модель и повторите моделирование. Вы увидите практически те же результаты, что и на рис. 16. Разница заключается лишь в том, что сигнал DB_INT сдвинут относительно входного сигнала DA на 10 ns. Это отличие внесено в модель умышленно: иначе мы просто не заметим, что имеем дело не со схемой замещения, а с VHDL-моделью шинного драйвера.

В качестве заключительного примера исследуем схему, в которой передача данных осуществляется не по одному проводу, а по системной шине данных Y[8..1] в обоих направлениях (рис. 20). В схеме используются реальные двунаправленные шинные драйверы 74ls245 (отечественный аналог 555АП6).

Нам понадобятся генераторы логического нуля (LO) и логической единицы (HI). Создать их проще простого. Графические образы (рис. 18) поместим в библиотеку my_lib.olb, а VHDL-модели (рис. 19) — в библиотечный файл my_lib.vhd.

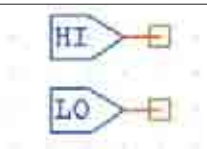


Рис. 18. Графические образы генераторов логической единицы и логического нуля

Исследуемая схема фактически повторяет предыдущую (рис. 15), с той лишь разницей, что здесь происходит обмен не одиночными, а шинными сигналами. Сначала данные передаются по маршруту: INP_A[8..1] — U2 — Q_A[8..1] — U1 — системная шина Y[8..1] — U4 — выходная шина Q_B[16..9]. Драйвер U3 выключен и его выходы B1… B8 находятся в Z-состоянии.

Затем направление передачи меняется на противоположное, и данные начинают поступать от входа INP_B[16..9] через драйверы U3, U4 и U1 на шину Q_A[8..1], которая в этом режиме становится выходной. Результаты моделирования рассмотренной схемы показаны на рис. 21.

```
library ieee;
use ieee.std_logic_1164.all;

entity HI is
      port (Y: out std_logic);
end HI;

architecture model of HI is
begin
      Y <= '1';
end model;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity LO is
      port (Y: out std_logic);
end LO;

architecture model of LO is
begin
      Y <= '0';
end model;
```

Рис. 19. VHDL-модели генераторов логической единицы и логического нуля

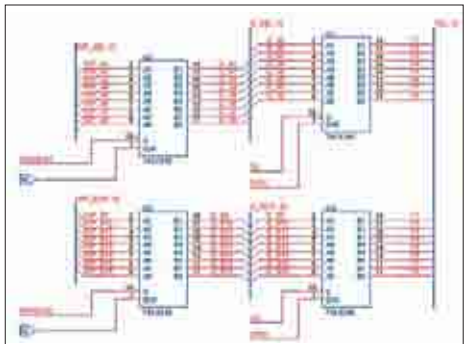


Рис. 20. Организация обмена данных через системную шину Y[8..1]

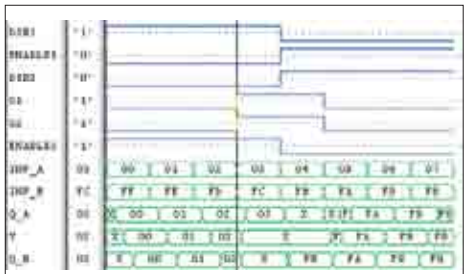


Рис. 21. Результаты моделирования шинной структуры