# Школа схемотехнического проектирования устройств обработки сигналов

**Занятие 11.**
**Языки описания аппаратуры: синтезируемое подмножество VHDL**

**Владимир Стешенко, к.т.н.**

steshenk@sm.bmstu.ru

## Общие сведения

Как известно, изначально язык описания аппаратуры VHDL создавался как средство моделирования цифровых систем. Однако его популярность и определенные удобства, появившиеся у разработчика, привели к тому, что модели на языке VHDL стали средством описания алгоритмов, синтезируемых специальными программными средствами в файлы прошивки (межсоединений) ПЛИС. В то же время для каждого программного продукта набор поддерживаемых конструкций языка различается, и порой довольно существенно. В настоящее время сделана попытка определить синтезируемое подмножество языка VHDL в стандарте IEEE P1076.6. Определяются те элементы языка, которые могли бы быть синтезированы средствами синтеза (компиляторами) на уровне регистровых передач (register transfer level). В этом случае средства синтеза (synthesis tools), удовлетворяющие стандарту IEEE P1076.6, могли бы обеспечить подлинную переносимость проекта и возможность единообразного описания. Рассмотрим элементы синтезируемого подмножества языка VHDL.

## Переопределенные типы (redefined types)

Средства синтеза, которые соответствуют стандарту IEEE P1076.6, должны поддерживать следующие переопределенные типы:

- BIT, BOOLEAN и BIT_VECTOR в соответствии со стандартом IEEE Std 1076-1993;
- INTEGER в соответствии со стандартом IEEE Std 1076-1993;
- STD_ULOGIC, STD_ULOGIC_VECTOR, STD_LOGIC и STD_LOGIC_ в соответствии с пакетом (Package) STD_LOGIC_1164 (стандарт IEEE Std 1164-1993);
- SIGNED и UNSIGNED в соответствии с пакетом NUMERIC_BIT, являющимся частью стандарта IEEE Std 1076.3-1997;
- SIGNED и UNSIGNED в соответствии с пакетом (package) NUMERIC_STD, являющимся частью стандарта IEEE Std 1076.3-1997.

Кроме того, должна быть обеспечена поддержка средствами синтеза типов, определенных пользователем (user-defined types).

## Методика верификациии синтезируемого описания (Verification methodology)

Схемы, полученные в результате процедуры синтеза, могут быть последовательностными (sequential) и комбинационными (combinational). Как известно, последовательностные схемы содержат некоторые элементы памяти (internal storage), такие как защелки, регистры и т. п. (latch, register, memory), которые учитываются при определении выходного значения сигнала, в то время как выход комбинационной схемы зависит только от входных переменных.

Процесс верификации синтезированного проекта состоит в подаче тестовых воздействий как на поведенческую модель, так и на синтезированную, с последующим сравнением результатов их функционирования. Нетрудно понять, что такой подход к процессу проектирования позволяет по заранее обкатанной «чисто поведенческой» модели отработать методику тестирования системы.

Как правило, входной тестовый сигнал должен удовлетворять следующим критериям:

- входные данные не должны содержать неизвестных величин;
- при верификации последовательностных устройств следует принимать во внимание, что триггеры имеют время установления и тестовые сигналы должны иметь необходимую для завершения переходных процессов длительность;
- тактовые сигалы и входные данные должны изменяться спустя достаточное время после асинхронного сброса или установки;
- первичные входные сигналы для синхронных последовательностных устройств должны измениться значительно раньше, чем активный фронт синхроимпульса. При этом входные данные должны оставаться неизменными достаточно долго, чтобы быть удержанными относительно активного фронта синхроимпульса. Аналогичны требования ко времени удержания сигнала и на входах последовательностной схемы, управляемой потенциалами.

Таким образом, входные данные должны оставаться неизменными достаточно долго, чтобы обеспечить требуемое время удержания (hold times) сигнала на входе.

Средства синтеза могут определять неопределенные значения, появляющиеся на магистральных выходах

в одной модели как эквивалент к логическим значениям в другой модели. По этой причине следует запоминать входной задающий сигнал до того, как проведено сравнение логических значений на выходах обеих моделей.

### Верификация комбинационных устройств (Combinational verification)

При верификации комбинационной логической схемы после подачи входного задающего тестового сигнала должно пройти некоторое время, прежде чем начнется анализ сигнала на выходах схемы. Как правило, это происходит в цикле таким образом, что выходы анализируются непосредственно перед подачей следующего набора входных воздействий, то есть по завершении переходных процессов. Каждая итерация цикла должна включать достаточную задержку, превышающую задержки распространения (transient delays) и паузы (timeout clause delays). Если не придерживаться этих правил, то тестируемое устройство может никогда не достигнуть установившегося состояния (то есть демонстрировать колебательное поведение), например:

```
A <= not A after 5 ns;
```

### Верификация последовательностных устройств (Sequential verification)

При верификации последовательностных устройств обычно ко входам схемы прикладываются внешние воздействия, после этого оценивается состояние выходов. Затем процесс повторяется. Но поскольку последовательностное устройство управляется либо фронтом, либо уровнем тактового сигнала, то процесс верификации имеет некоторые особенности.

При верификации устройств, тактируемых фронтом (edge-sensitive models), подача входных данных и проверка выхода тактируются синхросигналом. Проверка выхода должна выполняться до подачи следующего активного фронта тактового сигнала. Таким образом, время переходных процессов в триггерах и период синхросигнала должны соответствовать друг другу.

При верификации устройств, тактируемых уровнем (level-sensitive models), следует учитывать, что предсказать их поведение гораздо сложнее, чем поведение устройств, тактируемых фронтом из-за асинхронной природы взаимодействий сигналов. Проверка результатов синтеза зависит от приложения. Общие правила: вводные данные должны установиться, пока активен сигнал разрешения; проверка выходов осуществляется при неактивном сигнале разрешения.

### Моделирование элементов аппаратуры (Modeling hardware elements)

Рассмотрим особенности описания таких устройств, как тактируемые фронтом последовательностные схемы (edge-sensitive storage elements), последовательностные схемы, тактируемые уровнем (level-sensitive storage elements), и схемы с тремя состояниями (three-state drivers).

### Синхронные последовательностные схемы (Edge-sensitive sequential logic)

### Типы тактового сигнала (Clock signal type)

Для задания тактового сигнала могут быть использованы типы BIT, STD_ULOGIC и их подтипы (например, STD_LOGIC) с минимальным набором значений «0» и «1». Рассмотрим пример описания тактового сигнала.

```
signal BUS8: std_logic_vector(7 downto 0);
...
process (BUS8(0))
begin
if BUS8(0) = '1' and BUS8(0)'EVENT then
...
...
-- BUS8(0) используется как тактовый сигнал.
```

### Определение фронта тактового сигнала

Функция RISING_EDGE представляет передний фронт, а функция FALLING_EDGE используется для описания заднего фронта тактового импульса. Эти функции объявляются в пакете STD_LOGIC_1164 (стандарт IEEE Std 1164-1993) или NUMERIC_BIT, определенном в стандарте IEEE Std 1076.3-1997. Синтаксис определения тактового сигнала имеет вид:

```
clock_edge::=
RISING_EDGE(clk_signal_name)
| FALLING_EDGE(clk_signal_name)
clock_level and event_expr
event_expr and clock_level
clock_level::=
clk_signal_name = '0' | clk_signal_name = '1'
event_expr::=
clk_signal_name'EVENT
| not clk_signal_name'STABLE
```

### Передний фронт

Ниже показаны способы задания переднего фронта тактового сигнала для использования в качестве условия в условном операторе (*if* statement):

```
(positive <clock_edge>):
RISING_EDGE(clk_signal_name)
clk_signal_name'EVENT and clk_signal_name = '1'
clk_signal_name = '1' and clk_signal_name'EVENT
not clk_signal_name'STABLE and clk_signal_name = '1'
clk_signal_name = '1' and not clk_signal_name'STABLE
```

Следующие конструкции для задания переднего фронта используются в качестве условия в операторе *wait until*:

```
RISING_EDGE(clk_signal_name)
clk_signal_name = '1'
clk_signal_name'EVENT and clk_signal_name = '1'
clk_signal_name = '1' and clk_signal_name'EVENT
not clk_signal_name'STABLE and clk_signal_name = '1'
clk_signal_name = '1' and not clk_signal_name'STABLE
```

### Задний фронт

Ниже показано задание заднего фронта тактового сигнала для использования в качестве условия в условном операторе (*if* statement):

```
FALLING_EDGE(clk_signal_name)
clk_signal_name'EVENT and clk_signal_name = '0'
not clk_signal_name'STABLE and clk_signal_name = '0'
clk_signal_name = '0' and clk_signal_name'EVENT
not clk_signal_name'STABLE and clk_signal_name = '0'
clk_signal_name = '0' and not clk_signal_name'STABLE
```

Для использования в качестве условия в операторе *wait until* используются следующие конструкции для задания заднего фронта:

```
FALLING_EDGE(clk_signal_name)
clk_signal_name = '0'
clk_signal_name'EVENT and clk_signal_name = '0'
clk_signal_name = '0' and clk_signal_name'EVENT
not clk_signal_name'STABLE and clk_signal_name = '0'
clk_signal_name = '0' and not clk_signal_name'STABLE
```

### Описание синхронных последовательностных устройств

При синхронном назначении (synchronous assignment) переменных или сигналов их изменение происходит по фронту тактового импульса (clock edge).

Сигнал, модифицированный в синхронном назначении, описывает один или несколько синхронных триггеров (edge-sensitive storage elements).

Переменная, модифицированная в синхронном назначении, может быть использована для описания синхронного триггера. В одном процессе может быть использован только один фронт тактового сигнала.

### Использование оператора if

Для описания синхронных последовательностных устройств удобно использовать условный оператор *if*, задавая фронт сигнала в качестве условия (см. выше). Приведем типовой шаблон (template) для описания синхронного последовательностного устройства:

```
process (<clock_signal>)
<declarations>
begin
if <clock_edge> then
<sequential_statements>
end if;
end process;
```

Тактовый сигнал в секции <clock_edge> должен задаваться в списке сигналов возбуждения процесса (process sensitivity list).

Последовательностные операторы, предшествующие условному оператору или следующие за ним, не поддерживаются средствами синтеза. Например:

```
DFF: process(CLOCK)
<declarations>
begin
if CLOCK'EVENT and CLOCK = '1' then
Q <= D; -- тактирование передним фронтом
end if;
end process;
```

### Использование конструкции *wait*

Синхронные последовательностные устройства (edge-sensitive storage element) могут быть описаны с использованием тактового сигнала (clock edge) как условия в операторе *wait until*.

Оператор ожидания (*wait until* statement) всегда является первым оператором процесса. В каждом процессе может быть использован только один оператор *wait until*. Ниже

приводится типовой шаблон для создания описания синхронных последовательностных устройств с помощью оператора *wait*:

```
process
<declarations>
begin
wait until <clock_edge>;
-- должен быть первым оператором в процессе
<sequential_statements>
end process;
```

Поскольку оператор *wait until* является первым оператором процесса, асинхронные сброс или установка не могут быть описаны с его помощью.

Сигналы, назначенные с помощью условного оператора или оператора выбора, не могут быть использованы для описания синхронных последовательностных устройств

Приведем пример D-триггера:

```
DFF: process
begin
wait until CLOCK = '0';
Q <= D; -- синхронизация задним фронтом
end process;
```

Еще один пример (счетчик):

```
process
variable VAR: UNSIGNED(3 downto 0);
begin
wait until CLOCK = '1';
VAR:= VAR + 1;
COUNT <= VAR;
end process;
```

Переменная VAR может описывать четыре триггера, синхронизируемых по переднему фронту.

## Асинхронные сброс и установка (asynchronous set-reset)

В схемах, тактируемых фронтом, возможны асинхронные сброс и установка (asynchronous set/reset). Ниже приводится шаблон асинхронного сброса и установки.

```
process (<clock_signal>, <asynchronous_signals>)
<declarations>
begin
if <condition1> then
<sequential_statements>
elsif <condition2> then
<sequential_statements>
elsif <condition3> then
...
elsif <clock_edge> then
<sequential_statements>
end if;
end process;
```

В ветвях оператора *if* проверяется условие сброса или установки. Фронт тактового сигнала может появиться только при выполнении условия в операторе *elsif*.

Последовательностные операторы (Sequential statements) не должны использовать тактовый импульс в условиях операторов *if*.

Список сигналов возбуждения процесса (sensitivity list) должен содержать:

- тактовый сигнал, задаваемый с помощью соответствующего выражения;
- все сигналы, которые проверяются в условиях оператора *if*, содержащегося в выражении для фронта синхроимпульса;
- все сигналы, использующиеся в последовательностных операторах, управляемых оператором *if*.

Последовательностные операторы, предшествующие условному оператору или следующие за ним, не поддерживаются средствами синтеза.

Следует помнить, что условия асинхронного сброса и установки не должны содержать выражений для фронта синхросигнала. Асинхронные сброс или установка имеют более высокий приоритет, чем синхросигнал. Очевидно, что необязательно описывать как сброс, так и установку. В 70 % случаев достаточно описания асинхронного сброса.

Приведем пример описания D-триггера с асинхронной установкой.

```
AS_DFF: process (CLOCK, RESET, SET, SET_OR_RESET, A)
begin
if RESET = '1' then
Q <= '0';
elsif SET = '1' then
Q <= '1';
elsif SET_OR_RESET = '1' then
Q <= A;
elsif CLOCK'EVENT and CLOCK = '1' then
Q <= D;
end if;
end process;
```

В этом примере сигнал Q сбрасывается по сигналу RESET и устанавливается по сигналу SET. Сигнал SET_OR_RESET может либо сбрасывать, либо устанавливать триггер в зависимости от величины A.

## Последовательностные узлы с потенциальным управлением (level-sensitive sequential logic)

Последовательностные узлы (storage element), управляемыми уровнями, моделируются с помощью сигналов или переменных, при этом внутри процесса не должно присутствовать конструкций, работающих по фронтам импульсов. Кроме того, в процессе не должно быть явных назначений сигналов с использованием оператора назначения (assignment statement). Все сигналы и переменные, участвующие в процессе, должны иметь четко определенные значения. Сигналы или переменные, назначаемые в процессе, не должны использовать фронты синхроимпульса. В списке сигналов возбуждения процесса (process sensitivity list) должны быть отражены все сигналы, участвующие в процессе.

Переменные в подпрограммах никогда не используются при описании последовательностных узлов с потенциальным управлением, потому что инициализируются при каждом запросе. Назначения сигнала с использованием операторов *if* или *case* не следует использовать при описании последовательностных узлов с потенциальным управлением. Рекомендуется избегать стиля моделирования, в котором значение сигнала или переменной читается перед его назначением.

Ниже приводится пример последовательностного узла с потенциальным управлением.

```
LEV_SENS:process (ENABLE, D)
begin
if ENABLE = '1' then
Q <= D;
end if;.
end process;
```

## Логика с третьим состоянием и моделирование шин (Three-state and bus modeling)

Трехстабильная шина моделируется, когда сигналу назначается значение «Z» в соответствии со стандартом IEEE Std 1164-1993. Назначение третьего состояния может быть условным (conditional) или безусловным (unconditional). Для сигналов, имеющих несколько источников (драйверов), должно соблюдаться условие: если хотя бы один драйвер имеет третье состояние, то остальные сигналы-драйверы должны иметь по крайней мере одно назначение в состояние «Z».

Следует заметить, что если объект назначен в состояние «Z» в процессе, инициализируемом фронтом или уровнем, средства синтеза могут ввести триггеры на всех входах устройства с третьим состоянием.

## Описание комбинационных логических схем (Modeling combinational logic)

Любой процесс, не содержащий тактовых импульсов (clock edge) или оператора ожидания (wait statement), описывается с использованием либо конструкций комбинационной логики, либо последовательностных схем, управляемыми уровнями.

Если назначение переменной или сигнала происходит во всех возможных вариантах выполнения процесса и все переменные и сигналы имеют определенные значения, то переменная или сигнал описывают комбинационную логику.

Если сигнал или переменная считываются до их изменения, то они могут быть использованы для описания комбинационной логики. Оператор параллельного назначения сигналов (concurrent signal assignment statements) всегда описывает комбинационную схему.

Список чувствительности (сигналов возбуждения) процесса (process sensitivity list) должен содержать все сигналы, влияющие на процесс.

## Директивы компилятора (псевдокомментарии, Pragmas)

Псевдокомментарии влияют на способ синтеза модели. В VHDL приняты следующие псевдокомментарии (Pragmas): атрибуты (Attributes) и метакомментарии (Metacomments).

## Атрибуты (Attributes)

Атрибуты компилятора не следует путать с атрибутами сигналов. Атрибуты компилятора используются для кодирования представлений перечислимых типов, а атрибуты сигналов определяют свойства сигнала — его изменение и т. п.

## Атрибут компилятора ENUM_ENCODING

Атрибут ENUM_ENCODING определяет способ кодирования значений перечисляемых типов. В определении этого атрибута задается представление литералов перечисляемых типов в форме строки. Строка состоит из символов, разделенных одним или несколькими пробелами. Число символов должно соответствовать числу литералов в перечислимом типе.

Каждый символ состоит из последовательности «0» и «1». Символ «0» означает низкий логический уровень, символ «1» — высокий.

Рассмотрим пример объявления атрибута.

```
type <enumeration_type> is (<enum_lit1>, <enum_lit2>,... <enum_litN>);
attribute ENUM_ENCODING: STRING;
```

Спецификация атрибута определяет кодирование для перечисления литералов.

```
attribute ENUM_ENCODING of <enumeration_type>: type is «<token1><space><token2><space>... <tokenN>»;
```

Символ <token1> задает представление <enum_lit1>, и т. д. Следует помнить, что применение этого атрибута приводит к несоответствиям моделирования при использовании, например, операторов отношения.

Пример:

```
attribute ENUM_ENCODING: string;
type COLOR is (RED, GREEN, BLUE, YELLOW, ORANGE);
attribute ENUM_ENCODING of COLOR: type is «10000 01000 00100 00010 00001»;
-- перечислимый литерал RED представляется как 10000,
-- GREEN как 01000, и т.д.
```

## Метакомментарии (Metacomments)

Метакомментарии используются для управления компиляцией. Существуют два типа метакомментариев:

```
-- RTL_SYNTHESIS OFF
-- RTL_SYNTHESIS ON
```

Средства синтеза игнорируют любое описание на VHDL после директивы RTL_SYNTHESIS OFF и возобновляют процесс синтеза по директиве RTL_SYNTHESIS ON.

В следующем занятии мы начнем рассмотрение синтезируемого подмножества языка Verilog.

## Литература

1. Стешенко В. ПЛИС фирмы ALTERA: проектирование устройств обработки сигналов — М.: Додэка, 2000.
2. Стешенко В. Школа схемотехнического проектирования устройств обработки сигналов // Компоненты и технологии, №3–8, 2000, № 1–4, 2001.
3. Стешенко В. Особенности проектирования аппаратуры цифровой обработки сигналов на ПЛИС с использованием языков описания аппаратуры // Сборник докладов 2-й Международной конференции «Цифровая обработка сигналов и ее применения» 21.09–24.09. — М.: МЦНТИ, 1999. Том 2. С. 307–314.
4. И. Потемкин. Функциональные узлы цифровой автоматики. — М.: Энергоатомиздат, 1986. С. 320, ил.
5. HDL Chip Design. Smith, Douglas J. Madison, AL: Doone Publications, 1996.
6. IEEE Standard VHDL Language Reference Manual. New York: Institute of Electrical and Electronics Engineers, Inc., 1994.
7. IEEE Std 1164-1993, IEEE Standard Multivalue Logic System for VHDL Model Interoperability (STD_LOGIC_1164).
8. IEEE Std 1076.3-1997, IEEE Standard Synthesis Packages (NUMERIC_BIT and NUMERIC_STD).