Несколько советов по проектированию цифровых устройств на VHDL для ПЛИС

Алексей ДЕНИСОВ
maildenisov@gmail.com

В статье даются рекомендации, правила и руководства по созданию описания цифровых устройств на VHDL (Very high speed integrated circuits Hardware Description Language) для программируемых логических интегральных схем (ПЛИС).

Введение

Описание цифрового устройства на VHDL должно быть простым, структурированным для понимания и легко адаптируемым к различным платформам, чтобы другие разработчики, участвующие в проекте, имели возможность быстро вносить изменения и верифицировать описание, а также использовать его в своих разработках. Одна из главных трудностей в использовании «чужого» проекта на VHDL — отсутствие четких указаний по именованию и правилам составления текста описания цифрового устройства. Такие указания по именам и правилам «хорошего тона» должны быть созданы на начальной фазе проекта и предписаны для всего коллектива разработчиков. Далее приводятся основные руководящие требования для построения такого описания, которое можно легко синтезировать и верифицировать.

Именование

Выбор имени сигнала, переменной и т.д. является важной, но часто игнорируемой частью процесса проектирования. Для задания имен руководствуйтесь следующими правилами:

1. Используйте в именах для всех сигналов, портов, переменных, констант, определяемых пользователем типов и параметров только строчные буквы.
2. Используйте осмысленные имена, которые должны объяснять поведение переменной/сигнала. Выразительное имя дает понимание, а не путает того, кто читает описание цифрового устройства, и это верно также по отношению к самому автору. Удачное имя объясняет, что означает величина вектора или переменной. Без этой информации трудно читать описание аппаратуры, и маловероятно, что оно будет написано без ошибок. В таблице 1 представлены некоторые из имен.
3. Используйте "clk" для обозначения тактового сигнала блоков/модулей, а также в качестве префикса для всех тактовых сигналов, которые поступают с PLL, например clk_inv, clk_96MHz. Применение данного имени для всех тактовых сигналов информирует о том, что происходят они от одного внешнего источника. Если внешних источников тактовых сигналов два и более, тогда добавляйте порядковый номер к имени "clk", например clk2, clk3, clk4, clk5 и, соответственно, clk2_inv, clk2_100MHz.
4. Используйте последовательный порядок битов при объявлении многоразрядных векторов переменных/сигналов от высокого к низкому, то есть от (n–1) до 0 для VHDL. Ноль здесь используется как младший разряд, поэтому все многоразрядные векторы переменных/сигналов должны быть в форме от (n–1) до 0. Эти соответствия кодирования помогут избежать ошибок при связи многоразрядных переменных между собой.
5. Добавляйте _n для указания инверсной фазы сигнала. Например:

```
data_in_n, grand_n.
```

Если есть два имени, которые отличаются только символами _n, то убедитесь, что они действительно являются логической инверсией один другого.

6. Используйте поименованные константы вместо указания непосредственных значений в тексте модуля/блока.
7. Используйте значимые имена файлов, которые должны объяснять их назначение. В таблице 2 представлены некоторые суффиксы, отражающие тип описания на VHDL.

Таблица 1. Типовые имена сигналов и переменных

rst, rst_n reset, reset_n	Асинхронные сбросы «1» / «0»
clk	Тактовая частота модуля
sclr	Синхронный сброс
clear	Рабочий сброс (никакого отношения к цепи начального сброса не имеет!)
load	Загрузка
ena, enable	Разрешение
inc, dec	Инкремент, декремент
wr, write	Строб записи
rd, read	Строб чтения
rd_int, wr_int или rd_reg, wr_reg	Внутренние регистровые сигналы модуля/блока

Таблица 2. Типовые суффиксы для имени файла

_rtl	Описание RTL модели	Uart_rtl.vhd
_beh	Описание behavioural модели	Uart_beh.vhd
_tb	Описание Test Bench	Uart_tb.vhd
_pack	Описание Package	Uart_pack.vhd
_top	Описание Top Level	Uart_top.vhd

Текст описания цифровых устройств

Приведем основные правила написания текста описания цифровых устройств.

1. Приводите иерархию проекта в соответствие с иерархией физического разбиения. Модули должны быть понятными — достаточно короткими. Но слишком маленькие модули порождают много ветвей в структуре дерева, и иерархия становится сама по себе трудной для понимания. Соблюдение иерархий упрощает генерацию и извлечение тестовых векторов и делает проще трансляцию модели в физическую структуру.
2. Делайте декларацию портов логичной и непротиворечивой. Пишите один порт на строку, добавляя короткий комментарий, если что-либо неочевидно из его имени. Например, если модуль имеет 5 или меньше портов, их можно перечислять в виде списка в следующем порядке: входы, входы/выходы, выходы. Если портов более 5, группируйте сигналы по функциональному назначению. Пример декларации портов показан в листинге 1.

```
entity division is
GENERIC (nbits: integer:=9; mbits: integer:=5);
      port (
            X: in std_logic_vector (nbits-1 downto 0); -- divisible
            Y: in std_logic_vector (mbits-1 downto 0); -- divisor
            Q: out std_logic_vector (nbits-1 downto 0); -- quotient
            R: out std_logic_vector (mbits-1 downto 0) ); -- remainder
end division;
```

Листинг 1. Пример декларативной части описания

3. Используйте функции и процедуры вместо повторения кода.
4. Для уплотнения кода используйте векторы, массивы, операторы генерации и циклы.
5. Применяйте технологически независимое и совместимое с различными пакетами проектирования описание схем на VHDL. Исключением является использование блоков памяти, PLL, three-state buses, bi-directional buses и других. Избегайте таких элементов, так как могут возникнуть ограничения для повторного проектирования и применения из-за их зависимости от технологии и производителя. Работу с ними лучше изолировать в отдельных модулях/блоках проекта.
6. Старайтесь избегать вставок вентильных примитивов в RTL-код, поскольку такой проект на логическом уровне трудно будет читать и повторно использовать.
7. Старайтесь не встраивать директивы синтеза в RTL-код. Они различаются в разных пакетах проектирования, поэтому могут вызывать конфликты и ограничения для повторного проектирования и использования. Если же они необходимы, то в комментариях укажите, для какого пакета проектирования директивы синтеза предназначены и какую функцию они выполняют.
8. Используйте для процессов метки (label), которые кратко поясняют работу процесса. Например, метка label_counter поясняет, что процесс описывает схему счетчика. Кроме того, ставьте метки в начале и в конце процесса, что позволит быстро ориентироваться в описании на VHDL. Пример написания метки для процессов показан в листинге 2.

```
label_counter: process (clk, en, count, rst)
               ..............
end process label_counter;
```

Листинг 2. Пример написания метки

9. Пишите комментарии, соответствующие описанию на VHDL. Это означает, что необходимо проверить комментарий на соответствие тексту. Необходимо модифицировать комментарий, когда меняете текст описания цифрового устройства.
10. Включайте в файл комментарий к любому сделанному вами изменению. В нем следует обозначить дату, имя внесшего изменение, собственно изменение и причину, его вызвавшую. Это поможет другим, занимающимся поддержкой текста описания на VHDL, понять, что происходит, и разобраться в проблеме, если сделанные изменения повлекли за собой дополнительные ошибки.
11. Описывайте процесс создания готовых ядер при использовании соответствующих программ (например, CoreGenerator для Xilinx ISE). При отсутствии описаний могут возникнуть ограничения для повторного проектирования и применения из-за зависимости от технологии и производителя. Работу с такими ядрами лучше изолировать в отдельных модулях/блоках проекта.
12. Предоставляйте для функционально законченного блока, а также для проекта в целом Test bench файлы, в которых создаются внешние воздействия, максимально соответствующие работе схемы в реальных условиях, и предоставляйте описания работы с ними. Это помогает в дальнейшем быстрее понять и верифицировать работу модуля/блока и проекта в целом.
13. Используйте сигналы вместо переменных в VHDL.
14. Используйте графический схемотехнический редактор для создания схемы верхнего уровня проекта/модуля/блока, которую затем можно автоматически перевести в структурное описание на VHDL. Графическое представление позволит наглядно увидеть структуру проекта/модуля/блока в целом.
15. Делайте список чувствительности наиболее полным. Когда списки чувствительности являются неполными, моделирование может не сходиться между pre- и post-synthesis netlists. В комбинаторных процессах список чувствительности должен содержать каждый сигнал, входящий в этот процесс, а для последовательных блоков в него нужно включать тактовый, а также синхронные и асинхронные сигналы. Избегайте лишних сигналов в списке чувствительности, поскольку они замедляют процесс моделирования. Пример процесса с полным и неполным списком чувствительности показан, соответственно, в листингах 3 и 4.

```
process (clk, en, cnt, rst)
begin
if (rst = '1') then
cnt <= (others => '0');
elsif (clk'event and clk = '1') then
if (en = '1') then
cnt <= cnt + «00000001»;
end if;
end if;
count <= cnt;
end process;
```

Листинг 3. Пример процесса с полным списком чувствительности

```
process (clk)
begin
if (rst = '1') then
cnt <= (others => '0');
elsif (clk'event and clk = '1') then
if (en = '1') then
cnt <= cnt + «00000001»;
end if;
end if;
count <= cnt;
end process;
```

Листинг 4. Пример процесса с неполным списком чувствительности

Руководство для синтеза

При проектировании особенно важно понимать взаимодействие между стилем HDL-кодирования, различными архитектурами устройств FPGA и программным обеспечением для автоматизированного проектирования. Приложения, от которых требуется высокая производительность и плотность размещения логики, являются очень критичными к используемому разработчиком стилю кодирования. Для получения оптимальных результатов необходимо понимание архитектуры матриц FPGA, принципов работы инструментов синтеза схем и программного обеспечения конечной трассировки.

Цель RTL — создание проекта через процесс синтеза. Каждый инструмент имеет свои собственные конструкции описаний для VHDL; использование этих конструкций позволяет сделать эффективным процесс синтеза, а также упростить post-synthesis анализ. Некоторые обобщенные руководящие принципы для принятия RTL-синтеза:

1. Избегайте использования конструкций, создающих различные защелки (Latchs). Они могут появляться из-за неполного или двусмысленного RTL-кода. Поэтому каждое ***if*** должно иметь ***else***. Когда проектируется аппаратура, часто требуется выполнение одних действий при условии «истина» и других при условии «ложь». Даже если считаете, что условие «ложь» маловероятно (или невозможно), все равно ***else*** для случая по умолчанию должно быть в каждом ***if***-операторе. Нарушение пары ***if–else*** может вызвать синтезирование логики (появление защелки), отличной от RTL модели. Пример описания, создающего защелку, и регистра приведен в листингах 5 и 6.

```
process (clk, rst, counter, reg)
begin
if rst = '1' then
reg <= '0';
elsif clk'event and clk ='1' then
if counter = «1010» then
reg <= '1';
end if;
end if;
end process;
```

Листинг 5. Пример описания, создающего защелку

```
process (clk, rst, counter, reg)
begin
if rst = '1' then
reg <= '0';
elsif clk'event and clk ='1' then
if counter = «1010» then
reg <= '1';
else
reg <= '0';
end if;
end if;
end process;
```

Листинг 6. Пример описания, создающего регистр

2. Избегайте в RTL петли комбинаторной обратной связи и асинхронной логики.
3. Используйте регистры и триггеры для последовательной логики, которые обеспечивают синхронизацию и, следовательно, предпочтительнее защелок. Если необходим сброс модуля/блока, то вместо использования «инициализации» в декларации VHDL используйте цепи сброса сигнала для начальной инициализации регистров.

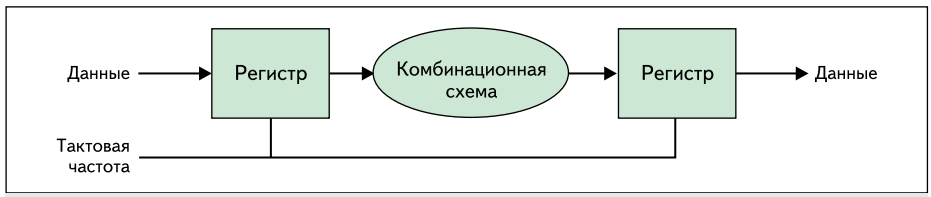

Рис. 1. Фиксация комбинационной схемы регистрами

4. Производите переключение последовательной логики по переднему (***clk'event and clk = '1'***) или по заднему фронту тактовой частоты (***clk'event and clk = '0'***) в модулях/блоках. Об этом следует задуматься еще в начальной стадии проектирования. По умолчанию принята работа схем по переднему фронту тактовой частоты (***clk'event and clk = '1'***).
5. Разделяйте RTL-код на процессы, чтобы процесс синтеза работал эффективнее, и могли быть легко проверены временные требования. Например, для конечных автоматов (finite-state machine) код может быть разделен на два процесса: один для комбинаторной логики, а второй — для последовательной.
6. Фиксируйте в ПЛИС данные/сигналы на регистрах — как на входах, так и на выходах (рис. 1). На выходах блоков/модулей рекомендуется иметь регистры — иначе больше вероятность того, что задержка в комбинационной схеме на выходе блока плюс задержка в межблочных соединениях и в комбинационной схеме на входе следующего блока превысят время между синхроимпульсами. Это также упрощает синтез, делает выход блока/ПЛИС надежным, а задержки более предсказуемыми. Фиксация на регистре делает технологию mapping легче.
7. В модуль верхнего уровня включайте только межсоединения (рис. 2). Структура, описанная на VHDL, может быть иерархичной и состоять из одного или более модулей, которые могут использовать любые другие модули. Модуль, который включает любой другой модуль, является узлом дерева, а не включающий никакого другого — его «листом». «Корень» — такой тип узла, который не входит ни в один другой модуль. Хотя VHDL и допускает проектирование «леса», но иерархия будет понятной, если используется только одно дерево. Конкретно это означает, что структура должна иметь только один корневой модуль, не содержащий чего-либо другого, кроме межсоединений и входящих модулей. Вся интерфейсная логика должна быть собрана в модулях нижнего уровня и в модулях, подсоединяемых к «корню».

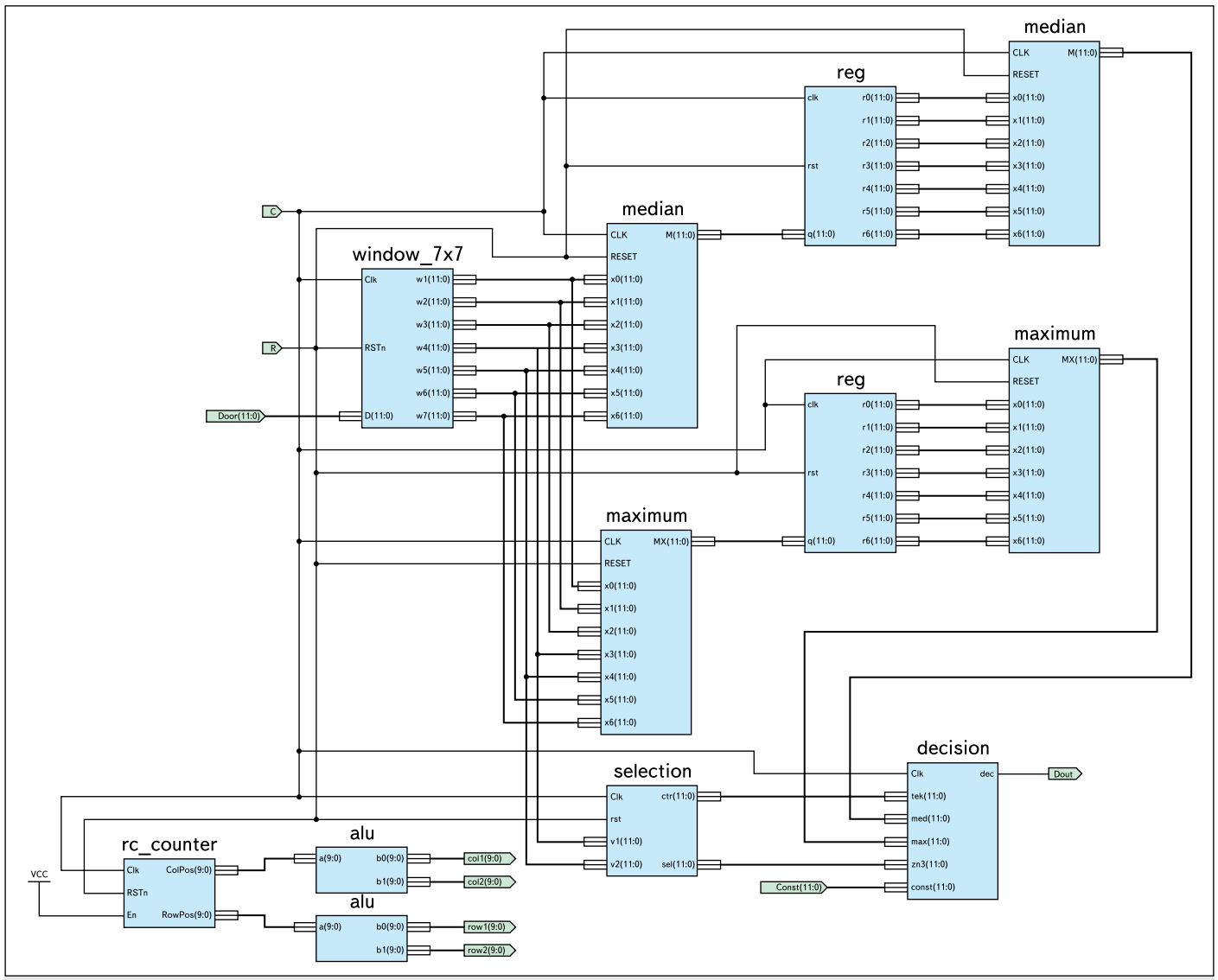

Рис. 2. Пример схемы верхнего уровня

Наиболее важной причиной нахождения логики вне корневого модуля является эффективный синтез. САПР не сможет оптимизировать логику в подмодулях, если в корневом модуле есть промежуточная логика. Исключение логики из корневого модуля делает структуру иерархически яснее и чище. Более радикальным стилем было бы исключить логику из всех узловых модулей, хотя это может вызвать избыточные уровни иерархии и не сделает описание на VHDL проще для чтения. И последней причиной исключения логики из корневого модуля является необходимость дать возможность программе-трассировщику создать хороший список межсоединений.

8. Не объединяйте в одном модуле/блоке последовательную логику, управляемую передним фронтом, и последовательную логику, управляемую задним фронтом тактового сигнала, а также модули/блоки, работающие на разных частотах. Каждый блок/модуль должен разрабатываться как «одноклоковый», в нем тактовая частота должна обозначаться “clk”, за исключением блоков стыков доменов. То есть подсоединение к сигналу вида clk_96MHz производите только на Top Level уровне.
9. Старайтесь избегать использования арифметических операторов, поскольку при реализации они требуют много логики и занимают большую площадь. Например, если требуется операция умножения или деления операнда на четную степень двойки, то можно использовать соответствующую операцию сдвига на регистре.
10. Используйте скобки для оптимизации логической структуры. Изменяйте структуру схемы, используя скобки для группирования логики таким образом, чтобы результат синтеза был ближе к оптимальному. Например, следующее выражение:

```
sum = a+b+c+d;
```

синтезатор транслирует в три последовательно соединенные сумматора. А такое выражение:

```
sum = (a+b) + (c+d);
```

транслируется в два параллельных сумматора для (a+b), (c+d) и один окончательный сумматор для сложения этих сумм.

11. Старайтесь использовать совмещение аппаратных ресурсов — это уменьшает затраты на оборудование. Примеры оптимизированного и неоптимизированного описания, соответственно, показан в листингах 7 и 8.

```
Signal sum: std_logic_vector (7 downto 0)
If (select = ‘1’) then sum <= a + b;
Else sum <= c + d;
End if;
```

Листинг 7. Пример VHDL-описания, порождающего в синтезируемой схеме два сумматора на входе мультиплексора

```
variable tmp1, tmp2: std_logic_vector (7 downto 0)
If (select = ‘1’) then tmp1:= a; tmp2:= b;
Else tmp1:= c; tmp2:= d;
End if;
sum <= tmp1 + tmp2;
```

Листинг 8. Пример оптимизированного VHDL-описания, порождающего два мультиплексора и один сумматор

Заключение

В статье дана попытка систематизировать правила создания текста описания на VHDL для повышения эффективной работы как отдельно взятого разработчика, так и коллектива в целом, а также выработать единые требования к синтезу и верификации описанных на VHDL цифровых устройств.

Литература

1. ALSE’s. VHDL Design Rules & Coding Style. 2005. **http://www.alse-fr.com/archive/VHDL_Coding_eng.pdf**
2. Cohen B. VHDL Coding Guide and Methodologies. Springer. August 31, 1995.
3. Rajsuman R. System-on-chip Design and test. Boston: Artech House, 2000.
4. Keating M., Bricaud P. Reuse Methodology Manual. Third Edition. Springer. June 30, 2002.
5. Поляков А. К. Языки VHDL и Verilog в проектировании цифровой аппаратуры. М.: СОЛОН-Пресс, 2003.
6. Каршенбойм И. Г. Краткий Курс HDL // Компоненты и технологии. 2008. № 3–4.