Соглашение по оформлению кода на языке Verilog

Организация файлов в проекте

- В ISE файлы исходного кода должны располагаться в папке src корневого каталога проекта.
- Папка src может содержать вложенные папки, в которых содержится исходный код больших функциональных блоков. Названия этих папок должны соответствовать названию модуля верхнего уровня функционального блока.
- Хорошей практикой является описание одного Verilog-модуля в одном файле. Исключением могут быть файлы библиотек, содержащие несколько модулей.
- Имя файла должно соответствовать имени модуля.
- Имя модуля должно отражать основную функцию, выполняемую модулем.
- Использование пробелов в именах файлов исходного кода, а также в именах папок, вложенных в src запрещено. Там, где это необходимо, нужно использовать символ подчеркивания.

Именование переменных

- Объявление портов должно быть выполнено в стиле Verilog 2001. Если это необходимо, можно использовать стиль Verilog 1995.

```
//Verilog 2001 Style
module Top (
  input Rst_i,
  input Clk_i,

  output Led_o
);
//Verilog 1995 Style
module Top (Rst_i, Clk_i, Led_i);
  input Rst_i;
  input Clk_i;
  output Led_o;
```

- Наименования переменных, портов, шин по возможности должны содержать в себе назначение сигнала и оформляться в соответствии с Camel-нотацией.
- Наименования портов, состояний автоматов, параметров модуля должны оформляться в UpperCamelCase.
- Наименования локальных переменных, шин, регистров должны оформляться в lowerCamelCase.
- Порты и сигналы имеющие стандартное назначение дополняются постфиксом через подчеркивание, который указывает на назначение порта или сигнала.

Список постфиксов для портов

_i - вход

_o - выход

_io - вход/выход

Список постфиксов для сигналов, регистров

_d1 - сигнал, задержанный на 1 такт

_d2 - сигнал, задержанный на 2 такта

ShReg - сдвиговый регистр, инициализируемый сигналом

```
reg dataValid;
reg dataValid_d1;
reg dataValid_d2;

always @ (posedge Clk_i)
begin
  dataValid_d1 <= dataValid;
  dataValid_d2 <= dataValid_d1;
end

reg [3:0] dataValidShReg;
always @ (posedge Clk_i)
  dataValidShReg <= {dataValidShReg[2:0], dataValid};
```

Имя сигнала сброса

Имя сигнала сброса должно содержать в себе "Rst". Помимо этого в названии может отображаться дополнительная информация о полярности сигнала сброса, является он синхронным или асинхронным, локальным или глобальным и т.п. Например:

```
  input Rst_i, // simple reset definition
  input RstGlbl_i, // global reset
  input RstAsync_i // asynchronous reset
```

Имя тактового сигнала

Имя тактового сигнала должно содержать в себе "Clk". Помимо этого в названии может отображаться дополнительная информация о тактовой частоте, является ли сигнал однополярным или дифференциальным и т.п.

```
  input Clk_i, // simple clock definition
  input Clk50Mhz_i, // 50 MHz clk
  input Clk_pi, Clk_ni, //differential clock
```

Группировка портов

Служебные входы (сигналы сброса и тактовые сигналы) обычно объявляются в начале списка портов модуля. Исключение составляют случаи, когда эти сигналы являются неотъемлемой частью функциональной или логической группы сигналов (например, шина AXI содержит в себе и тактовый сигнал, и сигнал сброса, которые следует объявить рядом с остальными сигналами этой шины).

Порты модуля могут образовывать функциональные или логические группы:

- служебные входы и выходы (сигналы сброса, тактовые сигналы, сигналы захвата частоты и т.п.)
- шины передачи данных
- блоки сигналов, относящихся к определенному интерфейсу
- ...

Такие группы следует:

- начинать с одного префикса
- отделять друг от друга пустой строчкой

Например:

```
module spi (
  input Rst_i,
  input Clk_i,

  input [7:0] DataToSpi_i,
  input DataToSpiNd_i,
  output [7:0] DataFromSpi_o,
  output DataFromSpiValid_o,

  output SpiClk_o,
  output SpiMosi_o,
  output SpiCs_o,
  input SpiMiso_i
);
```

Активный уровень логических сигналов

По умолчанию - активным уровнем сигнала считаем '1'. Исключением являются случаи использования стандартизированных интерфейсов, интерфейсов IP-ядер и т.п.

Следование правилам синхронного дизайна

Архитектура ПЛИС подразумевает работу преимущественно в рамках правил синхронного дизайна, которые включают следующее:

- избегание использования gated-clock (тактовый сигнал, порожденный логической схемой);
- избегание использования защелок(latch);
- использование входов триггеров clock-enable вместо большого количества тактовых частот;
- правильная синхронизация всех асинхронных сигналов;
- использование сигнала синхронного сброса.

Оформление блоков if

Во избежание некорректной привязки блока **else** условного оператора **if** всегда обрамляем блоки кода в **if**-блоке на уровнях выше самого низкого границами **begin end**. Например:

```
    if (condition1)
        begin
            if (condition2)
                doSomething;
        end
    else
        begin
            if (condition3)
                begin
                    if (condition4)
                        doSomething;
                    else
                        doSomething;
                end
            else
                doSomething;
        end
```

Создание платформонезависимого кода

Для упрощения переносимости/повторного использования кода между различными семействами ПЛИС одного производителя или между ПЛИС разных производителей рекомендуется там, где это допустимо, использовать синтезируемое соответствующим образом

описание примитивов. В основном это относится к аппаратным умножителям и блоковой памяти. Например, для описания двухпортовой памяти с одним портом на запись и одним на чтение с независимыми тактовыми сигналами вместо явного объявления BRAM можно использовать описание вида:

```
   parameter DataWidth = 36;
   parameter AddrWidth = 10;

   reg [RAM_WIDTH-1:0] ramBlock[(2**AddrWidth)-1:0];
   reg [RAM_WIDTH-1:0] dataOut;

   wire [AddrWidth-1:0] addrA, addrB;
   wire [DataWidth-1:0] inDataA;
   wire [DataWidth-1:0] outDataB;
   wire wrEnA;

   always @(posedge clkA)
      if (wrEnA)
         ramBlock[addrA] <= inDataA;

   always @(posedge clkB)
      outDataB <= ramBlock[addrB];
```

Также, вместо явного описания блока DSP, используемого для простого умножения двух 18-битных знаковых чисел, можно применить следующее описание:

```
   wire signed [17:0] aIn;
   wire signed [17:0] bIn;
   reg signed [35:0] p;

   always @(posedge <clock>)
      p <= aIn * bIn;
```

Замечание: При использовании описанного подхода необходимо удостовериться:

- в правильности описания поведения блока (т.е. оно должно соответствовать поведению аппаратного примитива; должна существовать возможность так сконфигурировать примитив, чтобы его поведение описывалось представленным кодом)
- в том что используемые средства разработки(в основном это касается средств синтеза) допускают неявное описание примитивов и корректно его интерпретируют

Замечание: В случае невозможности использования неявного описания примитивов допустимо использовать явное их описание. Для упрощения переносимости кода рекомендуется обернуть описание этого примитива промежуточным модулем-оберткой (wrapper) с интерфейсом, содержащим входы и выходы логически и функционально значимые для описываемого блока. Например, если мы описываем таким образом блок умножителя:

```
module multiplierWrapper (
   input Rst_i,
   input Clk_i,

   input signed [17:0] A_i,
   input signed [17:0] B_i,

   output signed [35:0] P_o
);

   DSP48A1 # (
     //...
     //Parameters definition
     //...
   ) dsp48Inst (
     //...
     //Port connections
     //...
   );

endmodule
```

Такой подход при переходе на другую архитектуру позволяет просто заменить блок с аппаратным примитивом на аналогичный для новой архитектуры ПЛИС без изменения интерфейса подключения.