Семинар 2

Языки описания аппаратных устройств

Оглавление

Обзор языков описания аппаратных устройств.....1
Язык VHDL.....2
Язык Verilog.....3
Язык SystemC.....4
Основы языка VHDL.....5
Структура проекта.....5
Описание объектов проекта.....5
Иерархия в описании проекта.....9
Стили описания.....10
Примеры описания проектов с использованием интерфейсов (структурный стиль описания).....12
Пример 1.....12
Пример 2.....18

Обзор языков описания аппаратных устройств

Языки описания аппаратных средств – это языки, пригодные для представления поведения и взаимосвязи параллельно протекающих во времени процессов. Применяемые к проектированию электронных устройств, они позволяют ускорить процесс создания последних, а так же, значительно снизить их себестоимость.

Язык высокого уровня C++ используется в качестве основного средства разработки программного обеспечения и создания моделей высокого уровня абстракции (поведенческие модели «Систем на кристалле»). Основные достоинства C++ - это низкая стоимость средств программирования, простота в освоении и использовании. Главный недостаток связан с отсутствием специализированных библиотек системного уровня, поэтому поведенческую модель разработчику приходится создавать практически «с нуля».

Существует несколько языков описания аппаратных средств. Но в настоящее время наиболее распространенными и часто используемыми являются VHDL, Verilog, System Verilog и SystemC.

Языки описания аппаратуры (HDL-языки) имеют две основные разновидности – языки низкого уровня (аналоги языков программирования типа ассемблера) и высокого уровня [4]. Языки низкого уровня ближе к аппаратным средствам, вследствие чего представляют для компиляторов потенциальные возможности создания проектов с более выигрышными параметрами. Платой за это является обычно жесткая ориентация на определенную аппаратуру и призводящую ее фирму. Примерами таких языков могут служить языки PLDASM (фирма Intel), AHDL (Фирма Altera) и ABEL (Фирма Xilinx). Языки высокого уровня менее связаны с аппаратными платформами и поэтому более универсальны. Среди

них наиболее распространены языки VHDL и Verilog. Эти языки, как и другие алгоритмические языки высокого уровня, в принципе позволяют описать любой алгоритм в последовательной форме, т.е. через последовательность операторов присвоения и принятия решений. Основное их отличие в способности отражать также и параллельно исполняемые в аппаратуре действия, представляемые отдельными параллельно выполняемыми процессами с общим инициализирующим воздействием. Совсем недавно был создан язык HDL-уровня - System Verilog. Это новый язык описания аппаратуры, созданный на базе Verilog и C++. Он призван расширять возможности классического языка описания Verilog и дает возможность проводить моделирование как на функциональном, так и на системном уровне. SystemC, как и System Verilog, разрабатывался как единый язык проектирования, способный обеспечить выполнение архитектурного моделирования и возможность синтеза разработанной системы.

Язык VHDL

Язык VHDL появился в начале 80-х годов по заказу организации Министерства обороны США [4]. Первая его версия, предназначенная в основном для унификации описаний проектов в различных ведомствах, была принята в 1985 году. В 1987 году язык VHDL был принят Международным институтом IEEE как стандарт VHDL-87. Он использовался, главным образом, для описания (спецификации) уже спроектированных систем. Использование для задач синтеза устройств (работа с компиляторами) началось с 1991 года. В 1993 году IEEE принял новый расширенный стандарт VHDL-93.

Язык может быть использован для проектирования ЦА разных иерархических уровней – от вентильного до системы в целом. В 1999 году был утвержден стандарт языка IEEE Std 1076.1-1999, известного под названием VHDL-AMS. Это расширение языка VHDL ориентировано на описание аналоговых и смешанных (аналого-йифровых) устройств. В настоящее время язык VHDL является самым популярным среди проектировщиков цифровой аппаратуры.

Язык VHDL является проблемно-ориентированным, его основные прикладные аспекты связаны с использованием в качестве рабочего инструмента для задач описания структуры и (или) поведения широкого класса цифровых устройств. Описания могут использоваться для синтеза и (или) моделирования таких схем. В соответствии с назначением, язык приспособлен для описания систем как с точки зрения их структурной организации (из модулей с известным поведением), так с точки зрения поведения либо системы в целом, либо всех его составных частей.

Синтаксические конструкции языка содержат две составляющие – алгоритмическую и проблемно-ориентированную.

Алгоритмическая составляющая языка достаточно традиционна и содержит как традиционные операторы действия (присваивание :=), условия (IF), выбора (CASE), цикла (LOOP), вызова процедуры), так и традиционные типы данных: числовые, логические, символьные, перечислительные и агрегатированные (массивы, записи, файлы).

VHDL поддерживает три различных стиля для описания аппаратных архитектур. Первый из них — структурное описание (structural description), в котором архитектура представляется в виде иерархии связанных компонентов. Второй — потоковое описание (data-flow description), в котором архитектура представляется в виде множества параллельных регистровых операций, каждая из которых управляется вентильными сигналами. Потоковое описание соответствует стилю описания, используемому в языках регистровых передач. И, наконец, поведенческое описание (behavioral description), в котором преобразование описывается последовательными программными предложениями, которые похожи на имеющиеся в любом современном языке программирования высокого уровня. Все три стиля могут совместно использоваться в одной архитектуре.

Каждый объект проекта состоит, как минимум, из двух различных типов описаний: описания интерфейса и одного или более архитектурных тел. Интерфейс описывается в объявлении объекта (entity declaration) и определяет только входы и выходы объекта.

Для описания поведения объекта или его структуры служит архитектурное тело (architecture body). Чтобы задать, какие объекты использованы для создания полного проекта, используется объявление конфигурации (configuration declaration).

В языке VHDL предусмотрен механизм пакетов для часто используемых описаний, констант, типов, сигналов. Эти описания помещаются в объявление пакетов (package declaration). Если пользователь использует нестандартные операции или функции, их интерфейсы описываются в объявлении пакета, а тела содержатся в теле пакета (package body).

Язык VHDL является очень распространенным языком описания аппаратуры. Существует множество литературы, в том числе и на русском языке, посвященной этому высококлассному средству разработки аппаратуры, что делает его очень доступным для обучения.

Пример описания счетчика на языке VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Counter is
Port(clk : in STD_LOGIC;
 Reset : in STD_LOGIC;
 Count : out STD_LOGIC_VECTOR (3 downto 0);
 Carry : out STD_LOGIC);
end Counter;
architecture Behavioral of Counter is
```

```
signal count_int : std_logic_vector(3 downto 0);
-- define interal register
begin
process (reset, clk)
begin
 if reset = '1' then
 count_int <= "0000"; -- set counter, and
 carry <= '0'; -- carry to zero
 elsif clk'event and clk = '1' then
 if count_int <= "1000" then -- check count
 count_int <= count_int + "1"; --increment
 carry <= '0'; -- show still below 9
 else -- else we are at 9
 count_int <= "0000"; -- roll over count
 carry <= '1'; -- flag roll over
 end if;
 end if;
end process;
count <= count_int; -- send value to the outside
end Behavioral;
```

Пример TestBench

```
        LIBRARY ieee;
        USE ieee.std_logic_1164.ALL;
        use IEEE.STD_LOGIC_ARITH.ALL;
        use IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY Counter_TB IS
END Counter_TB;

ARCHITECTURE behavior OF Counter_TB IS
-- Component Declaration for UUT)
COMPONENT Counter
PORT(
        clk : IN std_logic;
        Reset : IN std_logic;
        Count : OUT std_logic_vector(3 downto 0);
        Carry : OUT std_logic );
 END COMPONENT;
--Inputs
        signal clk : std_logic := '0';
        signal Reset : std_logic := '0';
--Outputs
        signal Count : std_logic_vector(3 downto 0);
```

```
        signal Carry : std_logic;
-- Clock period definitions
constant clk_period : time := 50 us; -- = 20KHz
BEGIN
-- Instantiate the Unit Under Test (UUT)
         uut: Counter PORT MAP (
        clk => clk,
        Reset => Reset,
        Count => Count,
        Carry => Carry );
         clk_process :process -- Clock process definitions
         begin
         clk <= '0';
         wait for clk_period/2;
         clk <= '1';
         wait for clk_period/2;
         end process;
         setup : PROCESS -- generate short reset pulse
         BEGIN
         reset <= '0';
         wait for 3 ns;
         reset <= '1';
         wait for 3 ns;
         reset <= '0';
         wait;
 end PROCESS;
 END;
```

Язык Verilog

Язык Verilog был разработан фирмой Gateway Design Automaton как внутренний язык симуляции [8]. Cadence приобрела Gateway в 1989 г. и открыла Verilog для общественного использования. В 1995 г. был определен стандарт языка — Verilog LRM (Language Reference Manual), IEEE1364-1995. Таким образом, датой появления языка Verilog следует считать 1995 г. К этому времени уже успел получить широкое распространение другой язык высокого уровня для описания принципиальных схем — VHDL, появившийся еще в 1987 г. Несмотря на похожие названия, Verilog HDL и VHDL — различные языки. Verilog — достаточно простой язык, сходный с языком программирования С как по синтаксису, так и по «идеологии». Малое количество служебных слов и простота основных конструкций упрощают изучение и позволяют использовать Verilog в целях обучения. Но в то же время это эффективный и специализированный язык. VHDL обладает большей универсальностью и может быть использован не только для описания моделей цифровых электронных схем, но и для других моделей (например, модели экосистемы). Однако из-за своих расширенных возможностей VHDL проигрывает в эффективности и простоте, то есть на описание одной и той же конструкции в Verilog потребуется в 3–4 раза меньше символов (ASCII), чем в VHDL.

В Verilog существуют специфические объекты (UDP, Specify-блоки), не имеющие аналогов в VHDL. Также следует упомянуть стандарт PLI (Program Language Interface), который позволяет включать функции, написанные пользователем (например, на С), в код симулятора.

Основной структурной единицей Verilog описания является module. Модуль соответствует entity в VHDL. Модуль описывается ключевыми словами module — endmodule. В файле может быть описано несколько модулей. Другие модули могут подключаться к цепям модуля, образуя иерархическую структуру. При запуске Verilog симулятор строит иерархическое дерево из всех модулей, которые обнаружены в файлах, поданных на вход симулятора, и находит модуль верхнего уровня. Если таких модулей несколько, то происходит ошибка. Как правило, модуль содержит список портов — интерфейсных сигналов, которые служат для подключения его в других модулях. Порты бывают трех типов input — входы, output — выходы, inout — двунаправленные. Входы и двунаправленные порты должны иметь тип wire, а выходы могут быть как wire, так и reg. Построение иерархии (подключение модулей) возможно двумя способами: по имени (указываются имена портов, использованные при описании модуля) или по расположению (порядок сигналов такой же, как в описании модуля).

В настоящее время фактором, препятствующим быстрому изучению языка Verilog, является то, что этому языку посвящено не так много литературы, особенно на русском языке.

Пример описания счетчика на языке Verilog

```
module behav_counter( d, clk, clear, load, up_down, qd);

// Port Declaration

input   [7:0] d;
input   clk;
input   clear;
input   load;
input   up_down;
output  [7:0] qd;

reg     [7:0] cnt;


assign qd = cnt;


always @ (posedge clk)
begin
     if (!clear)
          cnt = 8'h00;
     else if (load)
          cnt = d;
     else if (up_down)
          cnt = cnt + 1;
     else
          cnt = cnt - 1;
end

endmodule
```

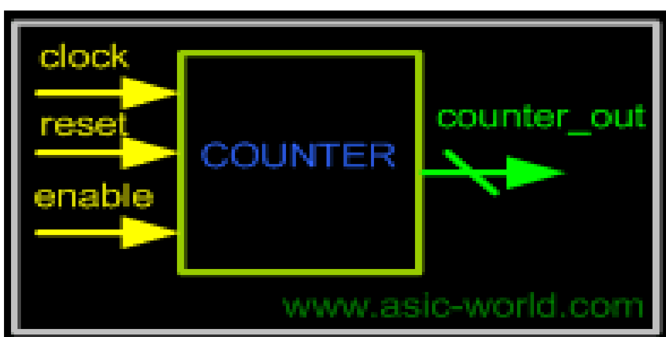
Язык SystemC

Необходимость создания нового языка описания аппаратуры возникает в том случае, если существующие языки не способны решать возникающие задачи. С одной из трудноразрешимых задач столкнулись самые популярные и распространенные языки VHDL и Verilog – это постоянно растущая сложность разрабатываемых проектов и необходимость в универсальном способе описания, пригодного для всех уровней проектирования и независимого от формы реализации проектов [4]. Попыткой решить возникшие трудности стало использование языка SystemC.

В 1999 году вышла в свет первая версия библиотеки SystemC 0.9. С этого времени можно считать появление языка SystemC. В настоящее время последняя версия данного продукта 2.2. Ассоциацией Стандартов IEEE одобрен стандарт библиотеки SystemC std. IEEE 1666 ™-2005 [1].

SystemC представляет собой надстройку стандартного языка программирования C++, реализованную в виде отдельных библиотек специальных классов. Данные библиотеки содержат в себе конструкции, позволяющие создавать эффективные и точные модели программных алгоритмов, аппаратных архитектур, интерфейсов и схем на системном уровне, т.е. практически всех компонентов встроенных систем [6]. Такой подход имеет значительный потенциал, так как основан на однородном описании C++ и легко позволяет моделировать, тестировать системы, рассматривать альтернативные архитектуры. Кроме того, команде проектировщиков может быть предложено развёрнутое описание процесса работы всей системы. Это описание представляет собой программу C++, которая при исполнении ведёт себя так же, как и система.
Основная цель в использовании данного языка – добиться сокращения времени разработки СнК [7]. Это достигается за счет исключения из процесса проектирования этапа перехода с одного языка описания на другой, который не требуется при использовании традиционной методики разработкЯзык VHDLи сложнофункциональных СБИС для перехода от поведенческой модели на языке C++ к синтезируемой RTL-модели на HDL-языке.

Пример описания счетчика на языке SystemC



- 4-bit synchronous up counter.

- active high, asynchronous reset.
- Active high enable.

```
 1 //-----------------------------------------------------
 2 // This is my second Systemc Example
 3 // Design Name : first_counter
 4 // File Name : first_counter.cpp
 5 // Function : This is a 4 bit up-counter with
 6 // Synchronous active high reset and
 7 // with active high enable signal
 8 //-----------------------------------------------------
 9 #include "systemc.h"
10
11 SC_MODULE (first_counter) {
12   sc_in_clk     clock ;      // Clock input of the design
13   sc_in<bool>   reset ;      // active high, synchronous Reset input
14   sc_in<bool>   enable;      // Active high enable signal for counter
15   sc_out<sc_uint<4> > counter_out; // 4 bit vector output of the counter
16
17   //------------Local Variables Here---------------------
18   sc_uint<4>        count;
19
20   //------------Code Starts Here-------------------------
21   // Below function implements actual counter logic
22   void incr_count () {
23     // At every rising edge of clock we check if reset is active
24     // If active, we load the counter output with 4'b0000
25     if (reset.read() == 1) {
26       count =  0;
27       counter_out.write(count);
28     // If enable is active, then we increment the counter
29     } else if (enable.read() == 1) {
30       count = count + 1;
31       counter_out.write(count);
32       cout<<"@" << sc_time_stamp() <<" :: Incremented Counter "
33         <<counter_out.read()<<endl;
34     }
35   } // End of function incr_count
36
37   // Constructor for the counter
38   // Since this counter is a positive edge trigged one,
39   // We trigger the below block with respect to positive
40   // edge of the clock and also when ever reset changes state
41   SC_CTOR(first_counter) {
42     cout<<"Executing new"<<endl;
43     SC_METHOD(incr_count);
44     sensitive << reset;
45     sensitive << clock.pos();
46   } // End of Constructor
47
48 }; // End of Module counter
```

Основы языка VHDL

Структура проекта

Проект в VHDL определяется как совокупность связанных проектных пакетов. Проектными пакетами (design unit) называются независимые (external) фрагменты описаний, которые можно независимо анализировать компилятором и помещать в рабочую библиотеку проекта (Work).

Проектными пакетами могут быть:
- объявление интерфейса объекта проекта (entity);
- объявление архитектуры (architecture);
- объявление конфигурации (configuration);
- объявление интерфейса пакета (package);
- объявление тела пакета (package body).

Можно выделить две категории модулей проекта: первичные и вторичные. К первичным относятся объявления пакета, объекта проекта, конфигурации, к вторичным — объявление архитектуры, тела пакета. Файл, в котором размещаются один или несколько модулей проекта, называется файлом проекта (design file).

Все проанализированные модули помещаются в библиотеку проекта (design library) и становятся библиотечными модулями (library unit). Существует два класса библиотек проекта: рабочие библиотеки и библиотеки ресурсов. Рабочая библиотека — это библиотека Work, с которой в данном сеансе работает пользователь и в которую помещается пакет, полученный в результате анализа пакета проекта. Библиотека ресурсов — это библиотека, содержащая библиотечные модули, используемые в анализируемом модуле проекта. В каждый момент времени пользователь работает с одной рабочей библиотекой и произвольным количеством библиотек ресурсов.

Описание объектов проекта

Полное описание модели объекта проекта состоит из следующих частей:

а) описание интерфейса объекта проекта (entity), включающее:
- Port (списки входных и выходных сигналов);
- Generic (настраиваемые параметры модели);

б) описание архитектуры объекта проекта (ARCHITECTURE), включающее:
- объявление переменных и дополнительных (внутренних) сигналов;
- операторную часть, представляющую собой описание объекта проекта на структурном или поведенческом уровне;

в) (только для структурной формы описания) описание конфигурации (configuration), задающей подключение библиотеки моделей элементов и выборку их в качестве компонентов структуры.

Интерфейс объектов проекта описывается при помощи ключевого слова entity:

```
entity entity_name is
 [generic (generic_list);]
 port (port_list);]
 [begin
 passive_statements;
 ...]
 end entity entity_name;
```

Объявление интерфейса может содержать необязательное тело, следующее за ключевым словом begin, в котором могут применяться только пассивные (т.е. не изменяющие значение сигналов) выражения и операторы. Пример описания интерфейса объекта проекта:

```
entity AND2_Checked is
 port (
 X1: in BIT;
 X2: in BIT;
 Y: out BIT;
 )
 begin
 assert (X1 = ’1′ and X2 = ’1′)
 report «X1 and X2 active» severity note;
 …
 end entity AND2_Checked;
```

Правила и примеры:

Список портов должен определять имя, режим (т.е. направление) и тип каждого порта в Entity.

```
entity HALFADD is
  port(A,B : in bit;
       SUM, CARRY : out bit);
end HALFADD;
entity COUNTER is
port (CLK  : in std_ulogic;
     RESET: in std_ulogic;
     Q    : out integer
             range 0 to 15);
end COUNTER;
```

Entity самого высокого уровня обычно пустое, т.е. не имеет портов. Архитектурой этого Entity является тестовая программа.

```
entity TB_DISPLAY is
end TB_DISPLAY;


architecture TEST of TB_DISPLAY is
-- signal declarations
```

```
-- component declaration(s)
begin
-- component instance(s)
-- test processes
end TEST;
```

Каждый порт подобен сигналу, который доступен в архитектуре(ах) данного Entity.
Режимом (т.е. направление) каждого порта определяется, доступен он для чтения или для записи в данной архитектуре.

Режим	Чтение	Запись
In	да	нет
Out	нет	да
InOut	да	да
Buffer	да	да

Если в Entity есть настраиваемые параметры (generic), то они должны быть объявлены перед портами. Они не имеют режима, т.к. с помощью них можно только пропускать информацию в Entity.

```
entity AN2_GENERIC is
  generic (DELAY: time := 1.0 ns);
  port (A,B : in std_ulogic;
      Z : out std_ulogic);
end AN2_GENERIC;
```

Entity может также содержать декларации, декларированные элементы видимы только в архитектуре(ах) данного Entity.

Особенности синтеза

Декларации Entity синтезируемы и обеспечивают доступность портов для синтезатора.
Обычно настраиваемые параметры поддерживаются только типа Integer. Значения настраиваемых параметров должны быть указаны пользователем во время синтеза.
Architecture (Архитектура)

Архитектура - это вторичная часть проекта.

Синтаксис:

```
architecture имя_architecture of имя_entity is
 -- декларации
 begin
 -- параллельные операторы
 end имя_architecture;
```

Правила и примеры:

Декларации могут быть любыми из следующих типов: type, subtype, signal, constant, file, alias, component, attribute, function, procedure, configuration specification:

```
architecture TB of TB_CPU_IF is
  component CPU_IF
   port -- список портов (port list) …
  end component;
  signal CPU_DATA_VALID : std_ulogic;
  signal CLK, RESET : std_ulogic := ‘0’;
  constant PERIOD : time := 10 ns;
  constant MAX_SIM : time := 50 * PERIOD;
 begin
   -- параллельные операторы
 end TB;
```

Порядок параллельных операторов можно не соблюдать:

```
architecture EX1 of CONC is
   signal Z, A, B, C, D : integer;
 begin
   D <= A + B;
   Z <= C + D;
 end EX1;
   или
 architecture EX2 of CONC is
   signal Z, A, B, C, D : integer;
 begin
   Z <= C + D;
   D <= A + B;
 end EX2;
```

Элементы, объявленные в архитектуре, доступны всем процессам в пределах архитектуры.

Архитектура может содержать любое сочетание компонентов, процессов или иных параллельных операторов:

```
architecture TEST of TB_DFF is
   component DFF
    port (CLK, D : in std_ulogic;
   Q : out std_ulogic);
  end component;
  signal CLK, D, Q : std_ulogic := '0';
 begin
  UUT: DFF port map (CLK, D, Q);
  CLK <= not (CLK) after 25 ns;
  STIMULUS: process
  begin
   wait for 50 ns;
   D <= '1';
```

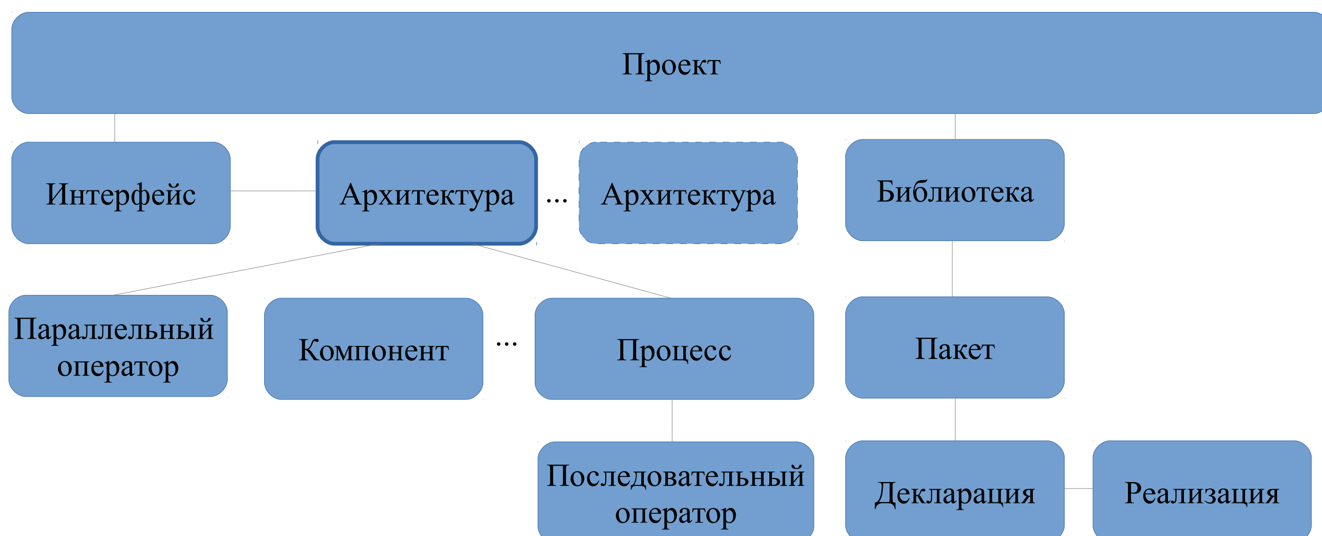
```
   wait for 100 ns;
   D <= '0';
   wait for 50 ns;
  end process STIMULUS;
 end TEST;
```

Для entity может быть одна или более архитектур. Которая из них будет использоваться – зависит от конфигурации (configuration).

Особенности синтеза:

Архитектуры поддерживаются системами логического синтеза.
Для некоторых систем синтеза, архитектура должна быть в том же файле проекта, что и entity.

Иерархия в описании проекта



Стили описания

В VHDL существуют три стиля описания архитектуры объектов — поведенческий, потоковый и структурный.

Поведенческий стиль

На поведенческом уровне описание объектов проекта представляется в VHDL в виде набора параллельных процессов. Организация процессов обеспечивается введением оператора процесса и оператора параллельного присваивания сигналов, также представляющего процессы.

Потоковый стиль

В потоковой форме описания объекта проекта его архитектура представлена в виде множества параллельных операций. Для сигналов вводятся специальные операторы параллельного присваивания "<=", являющиеся эквивалентами операторов присваивания ":=" для простых переменных, но имитирующие параллельные процессы с сигналами. Это реализуется искусственным приемом выполнения этих операторов на каждом шаге моделирования с бесконечно малой дельта-задержкой. Например, при выполнении операторов

```
Y <= X1 and X2; (1)
 Z <= not (X3 and Y); (2)
```

в правой части оператора (2) будет использоваться не полученное в операторе (1), а старое значение Y, определенное на предыдущем шаге моделирования.

Параллельное присваивание определено в трех различных формах:
- безусловное параллельное присваивание:

```
signal_name <= expression
```

- условное параллельное присваивание:

```
signal_name <= expression1 when boolean_expression1
 else expression2 when boolean_expression2
 ...
 else expressionN when boolean_expressionN
 else expression
```

- параллельное присваивание по выбору:

```
with expression select
 signal_name <= signal_value1 when choices1,
 signal_value2 when choices2,
 ...
 signal_valueN when choicesN;
```

Примеры различных форм параллельного присваивания приведены ниже:

```
signal A: BIT_VECTOR (3 downto 0);
 signal F1, F2, F3: BIT

– безусловное присваивание:
 F1 <= (A(1) and not A(0)) or (not A(1) and A(2) and not A(3));

-- условное параллельное присваивание
 F2 <= '1' when A = "0000" or A = "0010" or
```

```
 A = "0100" or A = "0110" or
 A = "1010" or A = "1100" or
 A = "1110"
 else '0';

-- параллельное присваивание по выбору
 with A select
 F3 <= '1' when "0000" | "0010" | "0100" | "0110" | "1010" | "1100" | "1110", '0' when others;
```

Условное параллельное присваивание удобно использовать при реализации различных схем выбора, например мультиплексоров:

```
– тип std_logic_vector определен в пакете std.logic
 entity mux is port (
 a, b, c, d: in std_logic_vector (3 downto 0);
 s: in std_logic_vector (1 downto 0);
 x: out std_logic_vector (3 downto 0)
 );
 end mux;

architecture archmux1 of mux is
 begin
 x <= a when (s = "00") else
 b when (s = "01") else
 c when (s = "10") else
 d;
 end archmux1;

architecture archmux2 of mux is
 begin
 with s select
 x <= a when "00",
 b when "01",
 c when "10",
 d when "11",
 (others => 'X') when others;
 end archmux2;
```

Примеры описания проектов с использованием интерфейсов (структурный стиль описания)

Пример 1

Устройство детектирования пакета на последовательной шине.

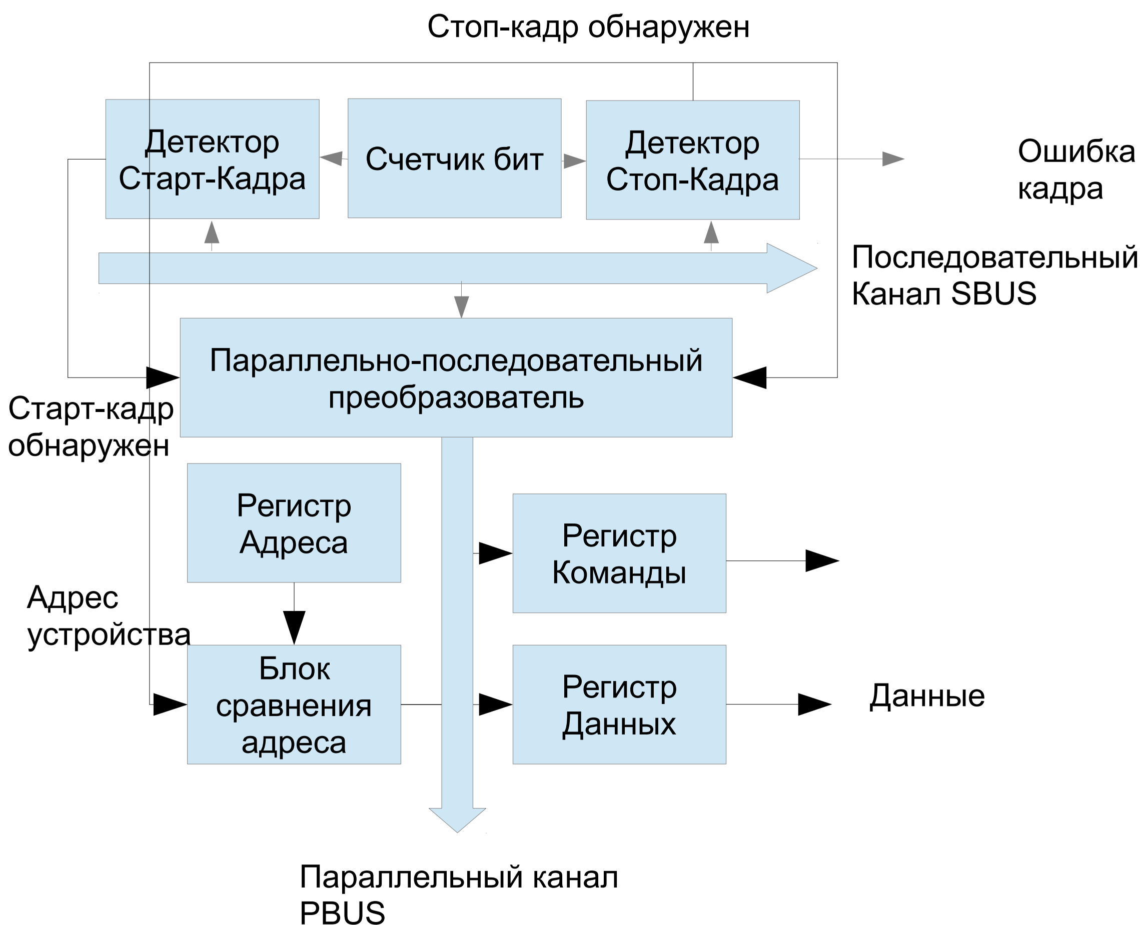
Пакет представляет собой последовательность

СтартКадр(10001)Адрес(12бит)Команда(4бита)Данные(64 Бита)СтопКадр(01110)

Необходимо распознать кадр, сравнить адрес с имеющимся и записать Адрес, Команду и Данные в регистры.

Детектор Старт Кадра определяет старт только если завершен предидущий кадр. Поэтому есть счетчик, который считает биты кадра и Детектор Стоп Кадра,Ю который после этого разрешает работу над следующим кадром. Сам счетчик также сбрасывается по сигналу от Детектора Стоп Кадра.

Параллельно-последовательный преобразователь начинает сдвигать данные по разрешению от Детектора Старт Кадра и сдвигает пока счетчик не переполнится. За этим произойдет детектирование стоп кадра (последовательность описана выше) и после этого последовательный преобразователь выдаст данные на PBUS. Схема сравнения адреса в этот момент сравнит адрес на шине PBUS.




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity EX1 is

GENERIC (
  ADR_W: integer:=12;                --Разрядность адреса
  CMD_W: integer:=4;                 --Разрядность команды
  DATA_W:  integer:=64;              --Разрядность данных
 ADR_CONST:  integer:=7;             --Адрес устройства на шине
);

PORT(
        CLK:IN std_logic; --Синхронизация
        RST:IN std_logic; --Сигнал сброса
        --Последовательный интерфейс
        SBUS:IN std_logic;
        --Данные на выход
        DATA_OUT:OUT std_logic_vector(DATA_W-1 downto 0);
        --Команда на выход
        CMD_OUT:OUT std_logic_vector(CMD_W-1 downto 0);
        --Команда и данные валидны
        VALID:OUT std_logic;
        --Ошибка пакета
        BUS_ERROR:OUT std_logic
);
end EX1;

architecture Behavioral of EX1 is


        --Параллельная шина
signal PBUS: std_logic_vector(ADR_W+CMD_W+DATA_W-1 downto 0);
        --Обнаружен Старт-кадр
signal START: std_logic;
        --Обнаружен Стоп-кадр
signal STOP: std_logic;
        --Адрес совпадает
signal ADR_EQ: std_logic;
```

```
	--Регистр адреса
signal ADR: std_logic_vector(ADR_W-1 downto 0);
	--Переполнение счетчика бит пакета
signal CNT_OVF: std_logic;
	--Разрешение записи регистров
signal REG_WRITE_ENABLE: std_logic;

component START_DETECTOR is
PORT(
	CLK:IN std_logic;
	RST:IN std_logic;
	--IN interface
	SBUS:IN std_logic; --SBUS
	CNT_OVF: IN std_logic;
	--OUT signals
	START_DETECTED:OUT std_logic
);
end component;

component STOP_DETECTOR is
PORT(
	CLK:IN std_logic;
	RST:IN std_logic;
	--IN interface
	SBUS:IN std_logic; --SBUS
	CNT_OVF: IN std_logic;
	--OUT signals
	STOP_DETECTED:OUT std_logic
);
end component;

component SBUS_COUNTER is
PORT(
	CLK:IN std_logic;
	RST:IN std_logic;
	RST_SYNC:IN std_logic; --Дополнительный сброс
	--IN interface
```

```
        CNT_OVF: OUT std_logic
);
end component;

component SBUS2PBUS is
GENERIC (
 DATA_WEIGHT: integer:=ADR_W+CMD_W+DATA_W;                --Разрядность регистра
);
PORT(
        CLK:IN std_logic;
        RST:IN std_logic;
        --IN interface
        SBUS:IN std_logic; --SBUS
        CNT_OVF: IN std_logic;
        START_DETECTED: IN std_logic;
        STOP_DETECTED: IN std_logic;
        --OUT signals
        PBUS: OUT std_logic_vector(DATA_WEIGHT-1 downto 0);

);
end component;

component ADR_CMP is
PORT(
        CLK:IN std_logic;
        RST:IN std_logic;
        --IN interface
        ADR_PBUS:IN std_logic_vector(ADR_W-1 downto 0);
        CNT_OVF: IN std_logic;
        ADR_EQ: OUT std_logic
);
end component;

component REG is
GENERIC (
 DATA_WEIGHT: integer:=32;           --Разрядность регистра
);
```

```
PORT(
	CLK:IN std_logic;
	RST:IN std_logic;
	--IN interface
	DATA_IN:IN std_logic_vector(DATA_WEIGHT-1 downto 0);
	WRITE_ENABLE: IN std_logic;
	DATA_OUT:OUT std_logic_vector(DATA_WEIGHT-1 downto 0)
);
end component;

begin

START_Inst: START_DETECTOR
PORT MAP(
	CLK=>CLK,
	RST=>RST,
	SBUS=>SBUS,
	CNT_OVF=>CNT_OVF,
	START_DETECTED=>START
);


STOP_Inst: STOP_DETECTOR
PORT MAP(
	CLK=>CLK,
	RST=>RST,
	SBUS=>SBUS,
	CNT_OVF=>CNT_OVF,
	STOP_DETECTED=>STOP
);


COUNTER_Inst: SBUS_COUNTER
PORT MAP(
	CLK=>CLK,
	RST=>RST,
	RST_SYNC=>STOP,
```

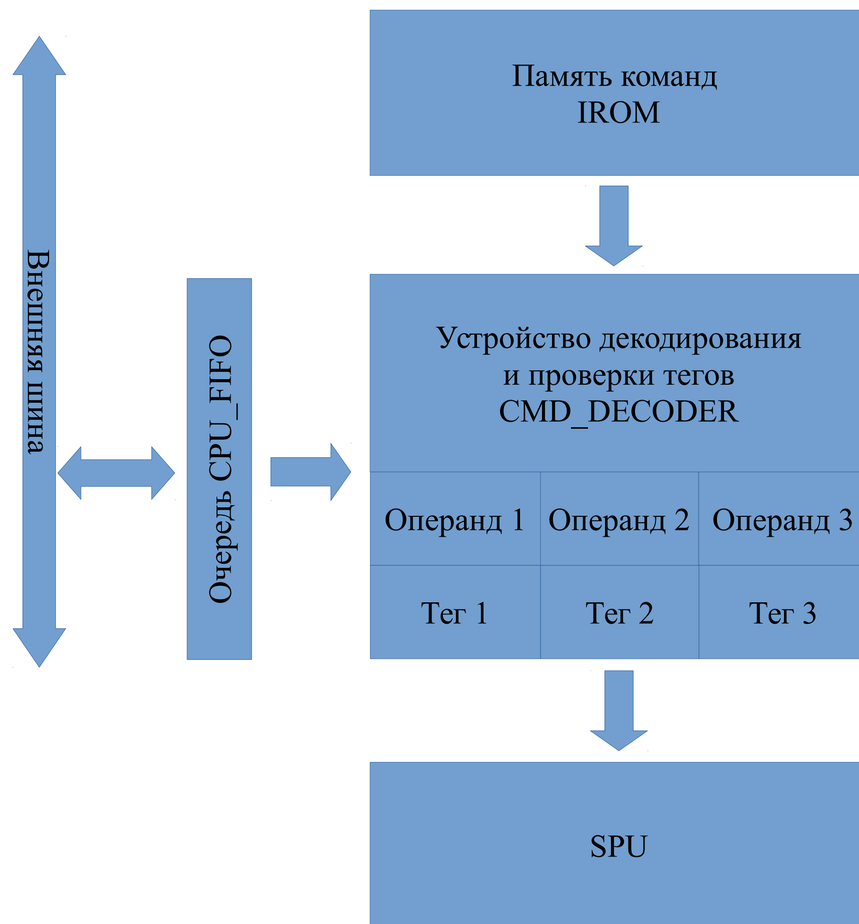
```
        CNT_OVF=> CNT_OVF,
);


S2P_Inst: SBUS2PBUS
GENERIC  MAP(
  DATA_WEIGHT=>ADR_W+CMD_W+DATA_W);
PORT MAP(
        CLK=>CLK,
        RST=>RST,
        SBUS=>SBUS,
        CNT_OVF=> CNT_OVF,
        START_DETECTED=>START,
        STOP_DETECTED=>STOP,
        PBUS=>PBUS
);

ADR_CMP_Inst: ADR_CMP
PORT MAP(
        CLK=>CLK,
        RST=>RST,
        ADR_PBUS=>PBUS(ADR_W+CMD_W+DATA_W-1 downto CMD_W+DATA_W),
        CNT_OVF=>CNT_OVF,
        ADR_EQ=>ADR_EQ,
);

REG_WRITE_ENABLE<='1' when CNT_OVF='1' and ADR_EQ='1' and STOP='1' else '0';

CMD_REG_Inst: REG
GENERIC MAP(
  DATA_WEIGHT=>CMD_W
);
PORT(
        CLK=>CLK,
        RST=>RST,
        DATA_IN=>PBUS(CMD_W+DATA_W-1 downto DATA_W),
        WRITE_ENABLE=>REG_WRITE_ENABLE,
```

```
        DATA_OUT=>CMD_OUT
);


DATA_REG_Inst: REG
GENERIC MAP(
  DATA_WEIGHT=>DATA_W
);
PORT(
        CLK=>CLK,
        RST=>RST,
        DATA_IN=>PBUS(DATA_W-1 downto 0),
        WRITE_ENABLE=>REG_WRITE_ENABLE,
        DATA_OUT=>DATA_OUT
);


END Behavioral
```

Пример 2

Устройство выборки и декодирования команд

Устройство извлекает из IROM памяти 80 разрядные команды, декодирует их и передает процессорному ядру (SPU). В каждой команде есть три тега. Если тег равен нулю, ожидается соответствующий операнд из внешней шины. Команда запускается только тогда, когда все необходимые операнды валидны.



Например:

Команда с кодом 000001 требует наличия двух валидных операндов: OP1 и OP2.

Если хотя бы один не валиден (ТЕГ =0), то команда не готова к исполнению (ждет операнда).

По шине такой операнд поступает вместе с адресом — номером тега. В каждом такте теги. Соответствующие команде проверяются и команда поступает дяльше (т. е. Сопроваждается сигналом CMD_SPU_START)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FIU is

GENERIC (
  CMD_CPU_W: integer:=32;            --Размер слова команды
  ADR_CPU_W: integer:=3;             --Размер слова адреса
```

```
  FIFO_CPU_W:  integer:=4;                    --Размер буфера
  IROM_ADR_W: integer:=8                      --Разрядность памяти команд

);

PORT(
        CLK:IN std_logic; --Синхронизация
        RST:IN std_logic; --Сигнал сброса
        RSTQ:IN std_logic; --Дополнительный сигнал сброса
        --Сигналы внешнего интерфейса
        CMD_CPU_VLD:IN std_logic; --Сигнал: «команда на шине данных валидна»
        ADR_CPU_IN:IN std_logic_vector(ADR_CPU_W-1 downto 0); --Адрес устройства на
шине
        CMD_CPU_IN:IN std_logic_vector(CMD_CPU_W-1 downto 0); --Код команды
        Q_CPU_FULL:OUT std_logic; --Очередь заполнена
        Q_CPU_EMPTY:OUT std_logic; --Очередь пуста
        --Интерфейс с процесосрным ядром
        CMD_SPU_START:OUT std_logic; --Сигнал запуска команды на исполнение
        SPU_RDY:IN std_logic; --Ядро готово к запуску команды
        CMD_SPU_CODE:out std_logic_vector(4 downto 0); --Код микрокоманды
        CMD_SPU_OP1:out std_logic_vector(31 downto 0); --Операны
        CMD_SPU_OP2:out std_logic_vector(31 downto 0); --
        CMD_SPU_OP3:out std_logic_vector(31 downto 0) --
);
end FIU;

architecture Behavioral of FIU is

        --CPU_FIFO_OUT interface
signal CMD_CPU_NXT: std_logic; --Impulse wdth must be 1 clock
signal ADR_CPU_OUT: std_logic_vector(ADR_CPU_W-1 downto 0);
signal CMD_CPU_OUT: std_logic_vector(CMD_CPU_W-1 downto 0);
signal Q_CPU_EMPTY_Int: std_logic;

        --SPU Instruction memory

constant DATA_WIDTH: integer:=80;
```

```
signal IROM_ADDR: std_logic_vector(IROM_ADR_W-1 downto 0);
signal IROM_DATA_OUT: std_logic_vector(DATA_WIDTH-1 downto 0);
signal IROM_DATA_OUT: std_logic_vector(DATA_WIDTH-1 downto 0);
signal DATA_FROM_FIFO,DATA_TO_FIFO: std_logic_vector(CMD_CPU_W+ADR_CPU_W-1 downto 0);

        --Регистр команды
constant CODE_WIDTH: integer:=5;

signal CMD_START,Q_CPU_EMPTYn: std_logic;

component FIFO_QUEUE is
GENERIC (
  DATA_W: integer;   --razrjadnost' slova komandi
  FIFO_W:      integer                    --Razmer FIFO
);
PORT(
        CLK:IN std_logic;
        RST:IN std_logic;
        RSTQ:IN std_logic;
        --IN interface
        DATA_VLD:IN std_logic; --Impulse wdth must be 1 clock
        DATA_IN:IN std_logic_vector(DATA_W-1 downto 0);
        Q_FULL:OUT std_logic;
        --OUT interface
        DATA_NXT:IN std_logic; --Impulse wdth must be 1 clock
        DATA_OUT:OUT std_logic_vector(DATA_W-1 downto 0);
        Q_EMPTY:OUT std_logic
);
end component;


component IROM is

    GENERIC (
      ADDR_WIDTH : integer;
      DATA_WIDTH : integer;
```

```
      FILENAME : string
     );
    PORT(
      CLK : IN std_logic;
      RST : IN std_logic;
      ADDR : IN std_logic_vector(ADDR_WIDTH-1 downto 0);
      DATA : OUT std_logic_vector(DATA_WIDTH-1 downto 0)
      );
end component;


component CMD_DECODER is

    GENERIC (
      DATA_WIDTH : integer;
                        ADR_WIDTH : integer
     );
    PORT(
      CLK : IN std_logic;
      RST : IN std_logic;
                        CMD_CPU_VLD : IN std_logic;
                        CMD_CPU_NXT : OUT std_logic;
                        CMD_CPU_IN : IN std_logic_vector(31 downto 0);
                        TAG_CPU_IN : IN std_logic_vector(2 downto 0);
                        SPU_RDY : IN std_logic;
                        CMD_SPU_START : OUT std_logic;
      CMD_SPU_IN : IN std_logic_vector(DATA_WIDTH-1 downto 0);
                        CMD_SPU_CODE:out std_logic_vector(4 downto 0);
                        CMD_SPU_OP1:out std_logic_vector(31 downto 0);
                        CMD_SPU_OP2:out std_logic_vector(31 downto 0);
                        CMD_SPU_OP3:out std_logic_vector(31 downto 0);
                        IROM_ADR:out std_logic_vector(ADR_WIDTH-1 downto 0)
      );

end component;
```

```
begin

--CPU command FIFO

DATA_TO_FIFO<=ADR_CPU_IN&CMD_CPU_IN;
ADR_CPU_OUT<=DATA_FROM_FIFO(CMD_CPU_W+ADR_CPU_W-1 downto
CMD_CPU_W);
CMD_CPU_OUT<=DATA_FROM_FIFO(CMD_CPU_W-1 downto 0);

CPU_FIFO: FIFO_QUEUE
GENERIC MAP(
  DATA_W => CMD_CPU_W+ADR_CPU_W,
  FIFO_W => FIFO_CPU_W
)
PORT MAP(
        CLK=>CLK,
        RST=>RST,
        RSTQ=>RSTQ,
        --IN interface
        DATA_VLD=>CMD_CPU_VLD,
        DATA_IN=>DATA_TO_FIFO,
        Q_FULL=>Q_CPU_FULL,
        --OUT interface
        DATA_NXT=>CMD_CPU_NXT,
        DATA_OUT=>DATA_FROM_FIFO,
        Q_EMPTY=>Q_CPU_EMPTY
);

--IROM
IROM_inst: IROM
GENERIC MAP(
        ADDR_WIDTH=>IROM_ADR_W,
        DATA_WIDTH=>DATA_WIDTH,
        FILENAME=>"irom.mem"
 )
PORT MAP(
```

```
            CLK=>CLK,
            RST=>RST,
            ADDR=>IROM_ADDR,
            DATA=>IROM_DATA_OUT
);



Q_CPU_EMPTYn<= not Q_CPU_EMPTY_Int;
Q_CPU_EMPTY<= Q_CPU_EMPTY_Int;

DECODER_Inst: CMD_DECODER

      GENERIC MAP (
        DATA_WIDTH=>DATA_WIDTH,
                              ADR_WIDTH=>IROM_ADR_W
       )
      PORT MAP(
        CLK=>CLK,
        RST=>RST,
                              CMD_CPU_VLD=>Q_CPU_EMPTYn,
                              CMD_CPU_NXT=>CMD_CPU_NXT,
                              CMD_CPU_IN=>CMD_CPU_OUT,
                              TAG_CPU_IN=>ADR_CPU_OUT(2 downto 0),
                              SPU_RDY=>SPU_RDY,
                              CMD_SPU_START=>CMD_SPU_START,
        CMD_SPU_IN=>IROM_DATA_OUT,
                              CMD_SPU_CODE=>CMD_SPU_CODE,
                              CMD_SPU_OP1=>CMD_SPU_OP1,
                              CMD_SPU_OP2=>CMD_SPU_OP2,
                              CMD_SPU_OP3=>CMD_SPU_OP3,
                              IROM_ADR=>IROM_ADDR(IROM_ADR_W-1 downto 0)
        );
end Behavioral;
```