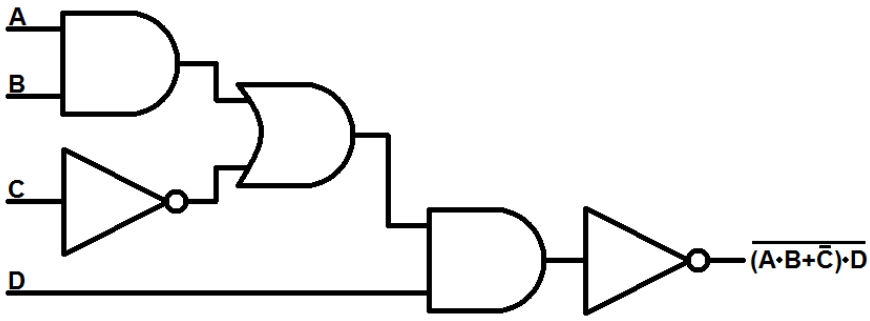
9 НОЯБРЯ 2015 Г.

```
MainProcess: PROCESS (clk, rst)
BEGIN
    IF rst ='1' THEN
        Asignal <= '0';
        Bsignal <= '1';
        Cport <= "0000";
        Dport <= x"00";
    ELSIF clk'EVEN and clk = '1' THEN
        -- There is should be your code
    END IF;
END PROCESS;
```



Практика по VHDL

Лабораторная работа №1

Юрий Михайлуц
ИММиКН ЮФУ

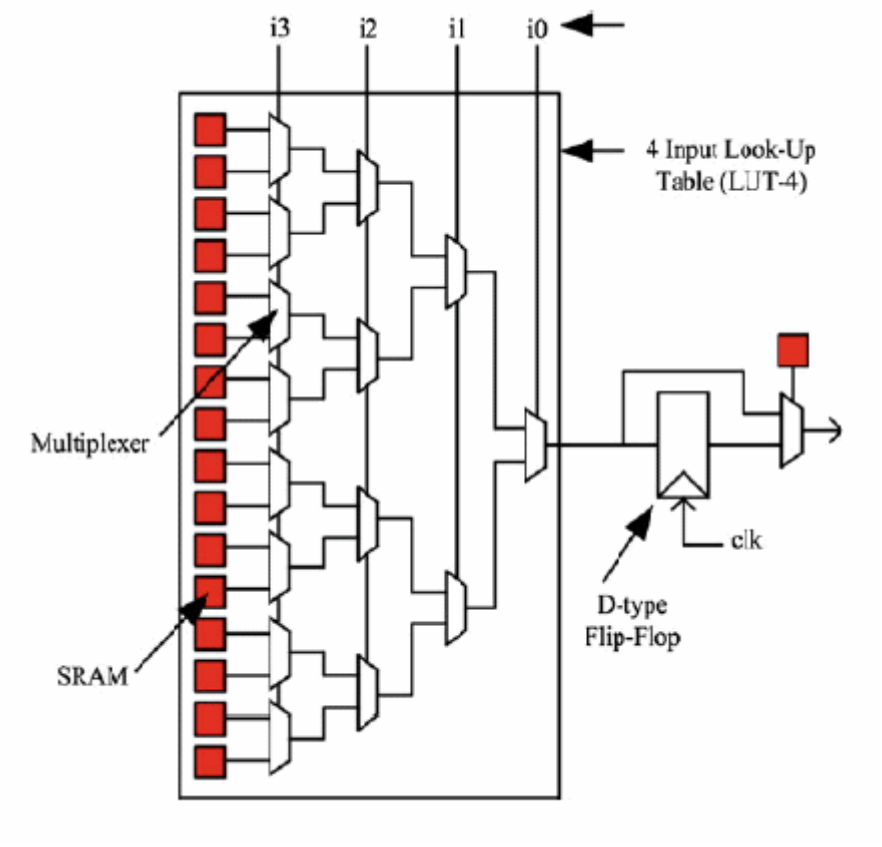
НЕОБХОДИМЫЕ ТЕОРЕТИЧЕСКИЕ ЗНАНИЯ

Элементы архитектуры ПЛИС

Конфигурируемые логические блоки. [1]

В документации компании Альтера встречается выражение Logic Array Block (LAB) – массив логики. У компании Xilinx в микросхемах ПЛИС есть примерно такие же блоки - Configurable Logic Block (CLB). Конфигурируемый логический блок – это базовый элемент в ПЛИС, в нем может быть выполнена какая-то простая логическая функция или реализовано хранение результата вычисления в регистрах (триггерах).

Конфигурируемый логический блок может состоять из одного или нескольких базовых логических элементов. В англоязычной литературе это Basic Logic Element (BLE) или просто Logic Element (LE). В ПЛИС обычно используются так называемые LUT-based базовые логические элементы. Что-то вроде этого:



LUT – это Look-Up Table, таблица преобразования. Например, на Рис.1 показан четырехбитный LUT в составе базового логического блока. Здесь четырехбитному числу на входе логической функции ставится в соответствие однобитный результат. Красные квадратики на Рис. 1 обозначают программируемый элемент, регистр – это та память, где хранится прошивка для ПЛИС. Видно, что для конфигурации 4-х битного LUT требуется 16 конфигурационных регистра. Содержимое этих регистров определяют логическую функцию, реализованную внутри базового логического элемента.

Еще один конфигурационный регистр (на Рис. 1 это одиночный красный квадратик справа) определяет нужно ли на выход базового логического элемента выдавать прямо значение с LUT

или нужно выдать зафиксированное в D-триггере значение с LUT. Фиксация и хранение данных в цифровых схемах нужна практически в любом проекте.

Рассматривая Рис. 1 как пример традиционного базового логического элемента понимаешь какая избыточность заложена внутрь современного кристалла ПЛИС (SRAM-based). Ведь в самом деле, конфигурационные регистры (красные квадратики) прямо не доступны для использования в цифровом проекте. Они только служат для формирования пользовательской функции. Для одного D-триггера в пользовательском проекте требуется более 16 (иногда много больше) триггеров для хранения конфигурации ПЛИС.

На самом деле базовый логический элемент в разных ПЛИС оказывается гораздо сложнее, чем показано на Рис. 1.

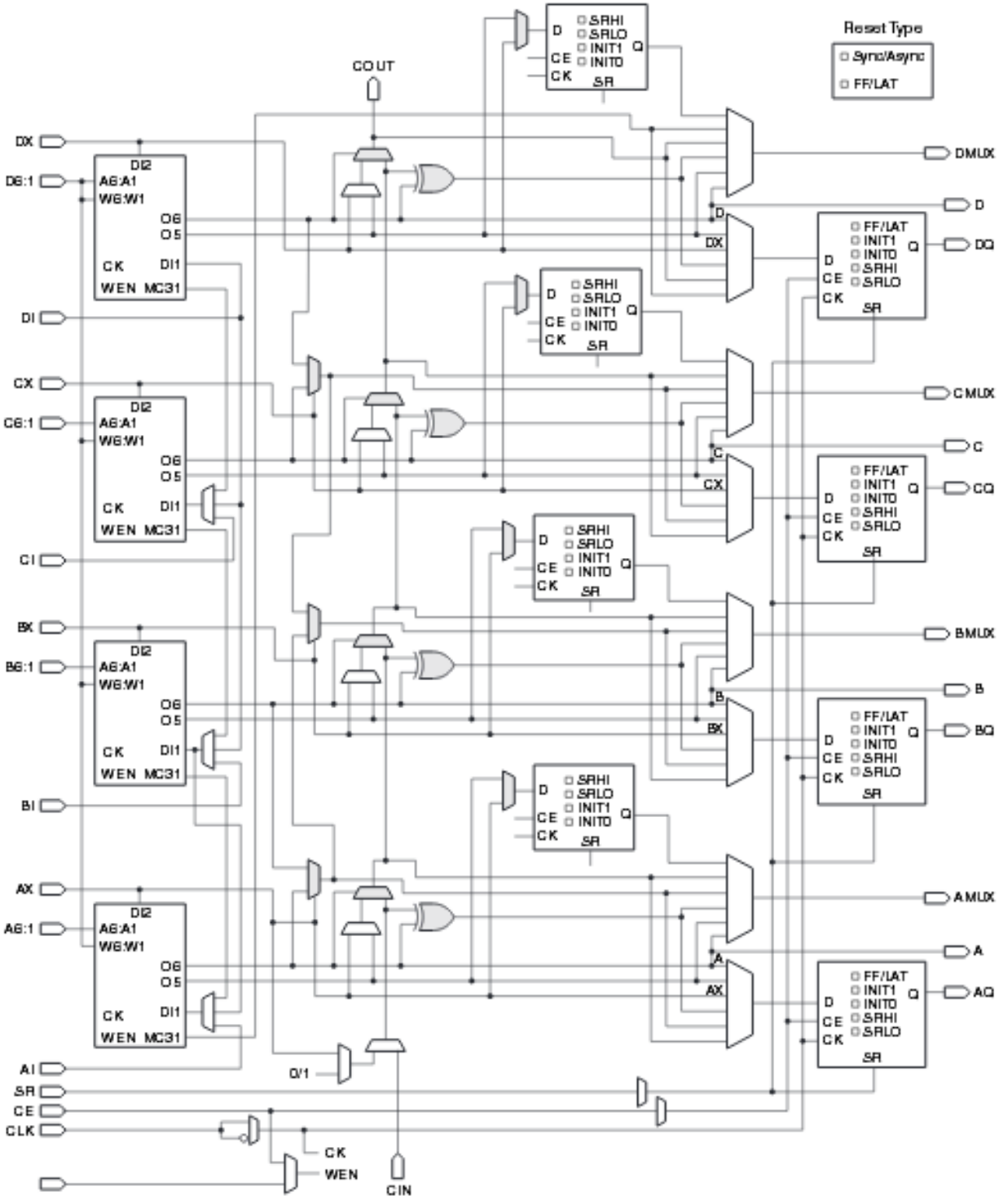


Иллюстрация 1: Базовый элемент Xilinx Virtex-6 Slice

Программируемые связи между логическими блоками.

Чтобы в ПЛИС заработала нужная нам цифровая схема мало того, что нужно сконфигурировать имеющиеся логические блоки особым образом, еще нужно создать, запрограммировать связи между логическими блоками.

Для этого в ПЛИС имеются специальные конфигурируемые коммутаторы.

В англоязычной документации встречаются следующие термины: FPGA Routing Architecture и Programmable Routing Interconnect. Это все об этом, о программируемых связях между логическими блоками.

Известно две основных методики построения ПЛИС по типу архитектуры связей: островная и иерархическая.

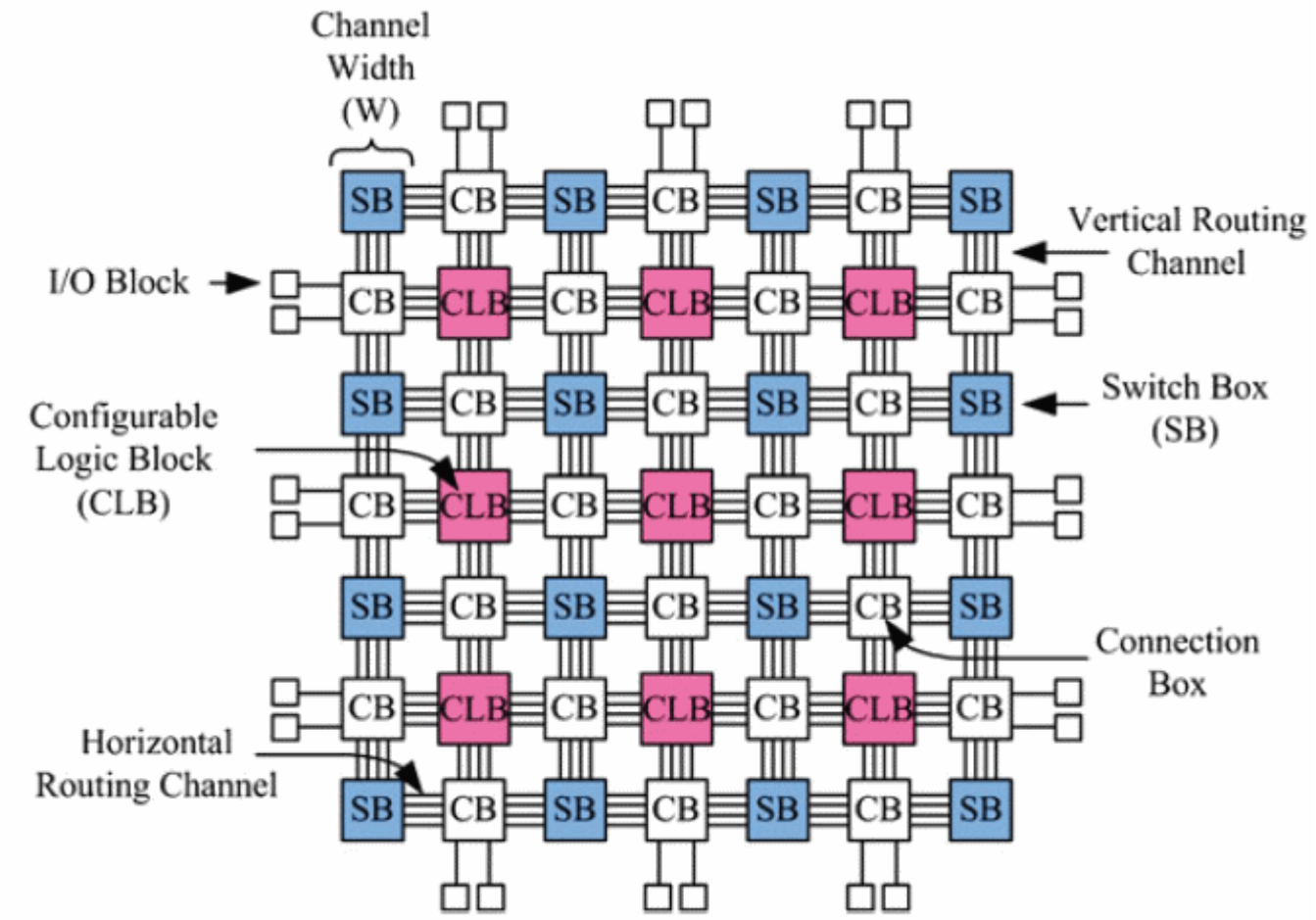


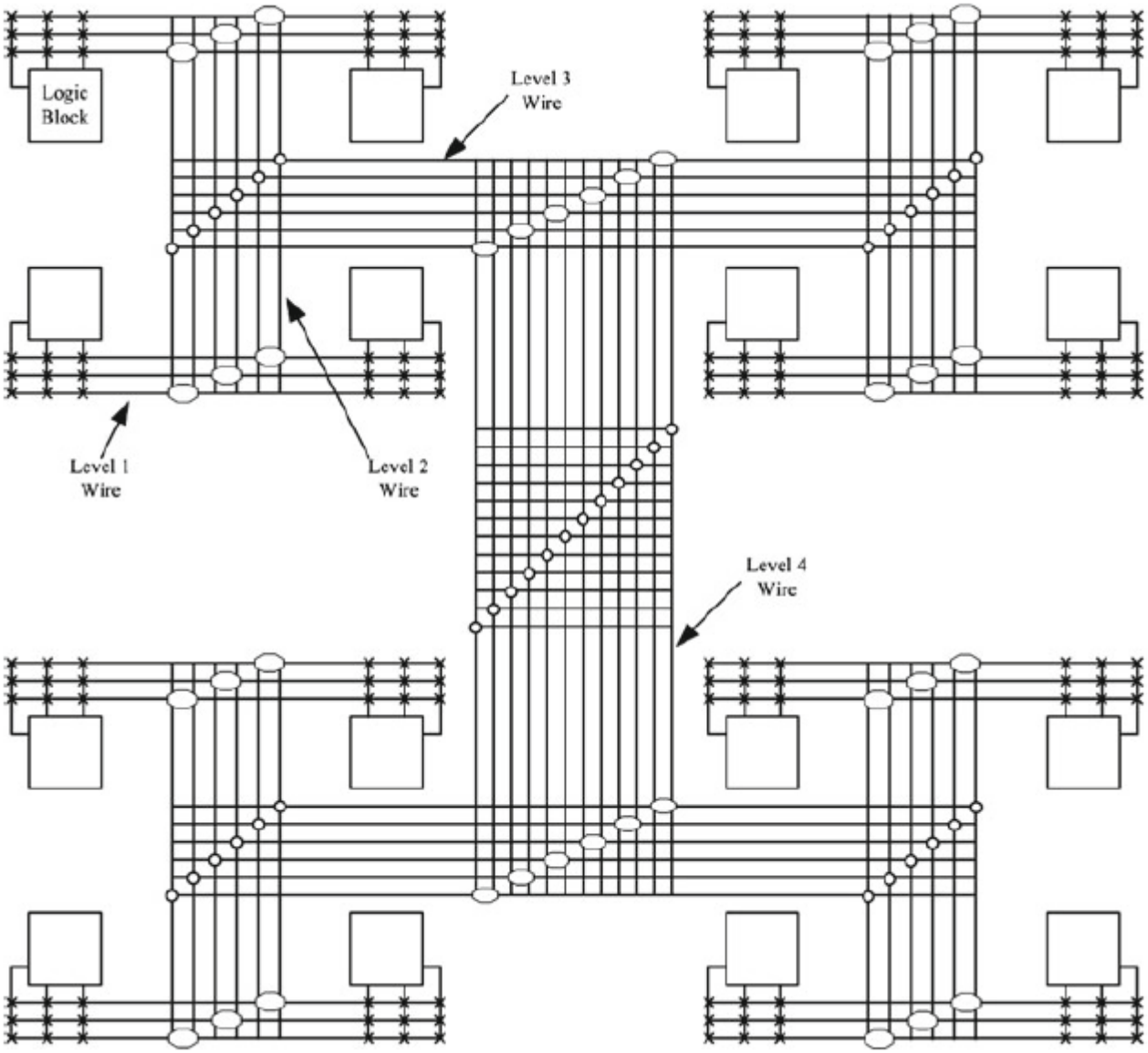
Иллюстрация 2: Островная ПЛИС

Островная ПЛИС называется так потому, что конфигурируемые блоки все равны между собой и находятся, как острова в океане, между узлами коммутации и линиями связи.

Здесь, на Рис. 6 обозначаются CB – Connection Box и SB – Switch Box. В сущности это программируемые мультиплексоры, подключающие тот или иной CLB к другому CLB через цепочки проводов в ПЛИС.

Это island-style FPGA или mesh-based FPGA. Типичный пример таких микросхем – это серии Altera Cyclone и Stratix.

Второй известный тип ПЛИС – это иерархические ПЛИС. Здесь идет расчет на то, что в схеме всегда есть участки которые взаимодействуют друг с другом более тесно, чем с отдаленными модулями проекта.



Здесь близлежащие CLB соединить довольно просто, нужно не много коммутаторов и получающиеся связи работают быстро. Вот если нужен более крупный блок вычислителей, то сигнал должен выйти на более высокий уровень иерархии и потом зайти вглубь в соседнюю «комнату».

Нельзя сказать, что это существенно хуже, чем island-style. Просто каждый метод имеет свои плюсы и минусы.

Структура программы на VHDL

Описание любого компонента в VHDL всегда начинается с подключения библиотек и пакетов. Даже если Вы поместите несколько компонентов в один файл, то перед каждым необходимо заново подключать библиотеки.

```
2
3   -- Следующая строка подключает стандартную библиотеку.
4   -- В VHDL она называется IEEE и в ней находятся все основные вкусняшки языка.
5   library IEEE;
6   -- Далее подключаются отдельные пакеты из стандартной библиотеки.
7   use IEEE.std_logic_1164.all;
8   use IEEE.std_logic_arith.all;
```

Далее идёт объявление прототипа. Тут объявляется только способ общения компонента с внешним миром, это, как бы, его интерфейс. Объявление прототипа делится ещё на две части: generic и port.

Generic служит для введения констант, наподобие тех, которые можно ввести в C++ при помощи шаблонов. При использовании компонента можно будет переопределить эти значения, но внутри компонента они будут восприниматься, как константа. Это очень удобный инструмент для построения компонентов с настраиваемой разрядностью или объёмом памяти.

Port объявляет имена переменных, которые могут быть видны другим компонентам, а так же способ доступа к ним (чтение / запись).

```
10
11  -- Объявление прототипа для компонента
12  entity simple is
13      -- generic в VHDL имеет схожий функционал с шаблонами в C++.
14      -- Через него в компонент можно передавать константы,
15      -- Значения которых будут определены позже.
16      generic(
17          n: integer := 16
18      );
19
20      -- Порт является основным окном во внешний мир для компонента.
21      port(
22          -- in порты служат для ввода информации и не могут её выводить
23          clk: in std_logic;
24          rst: std_logic;
25          a: in std_logic;
26          b: in std_logic(n downto 0);
27          -- out порты служат для вывода информации и не могут вводить
28          c: out std_logic
29      );
30  end simple;
31
```

Реализация компонента содержит объявления сигналов, выполняющих роль локальных переменных конкретно этого компонента, а также процессы, выполняющие основную работу в компоненте.

```
33  -- Реализация компонента
34  architecture simple_synth of simple is
35      signal e: std_logic;
36  begin
37
38      -- Это синхронный процесс
39      FirstProcessName: process (clk, rst) is
40      begin
41          if rst = '1' then
42          elsif clk'event and clk = '1' then
43          end if;
44      end process;
45
46      -- Это асинхронный процесс
47      SecondProcessName: process (a, b) is
48      begin
49          -- Просто пример из нескольких IF'ов работающих асинхронно.
50          -- Арифметика внутри примера клубинного смысла не несёт.
51          if a = '1' and b ='0' then
52              e <= '0';
53          elsif a = '0' or b ='0'then
54              e <= '1';
55          else
56              e <= '0';
57          end if;
58      end process;
59
60  end simple_synth;
```

Сигналы

Сигналы — это некий аналог переменных. В них можно сохранять значение и читать его оттуда. Но, поскольку все процессы в VHDL выполняются одновременно, то писать в сигнал может только один процесс, а вот читать могут все.

Запись в сигнал происходит в теле условного оператора, то он должен либо записываться синхронно, т. е. По фронту тактового сигнала, либо запись какого-либо значения сигнала должна присутствовать во всех ветвях условного оператора. В первом случае, после синтеза, сигнал превратится в регистр, а во втором в соединительную линию. Разница между этими двумя случаями в том, что будучи записанной в регистр, информация будет доступна для чтения только со следующего такта. Информация же записанная асинхронно доступна сразу после записи. Неправильная запись сигнала может привести к тому, что его значение будет отставать на один такт от ожидаемого.

```
1  signal firstSignal: std_logic;
2  signal secondSignal: std_logic_vector(15 downto 0);
3  signal integerSignal: integer;
4  signal signedSignal: signed(31 downto 0);
5  signal unsignedSignal: unsigned( 63 downto 0);
6
```

Иллюстрация 3: Пример объявления сигналов

Порты

Порты в VHDL ведут себя так же точно, как и сигналы, но могут быть записаны или прочитаны из другого компонента. Режим доступа (чтение / запись) задаётся ключевыми словами in, out и inout.

In-порты доступны только для чтения изнутри компонента и только для записи снаружи.

Out-порты, наоборот, снаружи только для чтения, а изнутри для записи.

Inout-порты доступны и для чтения и для записи с обеих сторон, но не стоит заблуждаться на их счёт, поскольку такой порт, на самом деле, представляет собой два порта с одним именем, противоположными направлениями и занимающими один и тот же физический контакт. Таким образом записав в inout-порт информацию вы прочтёте из него не её, а другую информацию, которую кто-то должен будет прислать «снаружи».

```
1   -- Порт является основным окном во внешний мир для компонента.
2   port(
3       -- in порты служат для ввода информации и не могут её выводить
4       clk: in std_logic;
5       rst: std_logic;
6       a: in std_logic;
7       b: in std_logic(n downto 0);
8       -- out порты служат для вывода информации и не могут вводить
9       c: out std_logic;
10      -- inout могут работать и как in порты и как out,
11      -- в зависимости от того читаете-ли Вы их или пишете.
12      d: inout std_logic
13  );
```

Процессы

Модель вычислителя без исполнителя

Во всех современных компьютерах имеется так называемый «исполнитель», который получает инструкции, исполняет их и возвращает результат. ПЛИС отличаются тем, что в ней исполнителя нет и в языке для их программирования VHDL, соответственно, тоже. Это означает, что нельзя просто послать на выполнение команду и получить результат. Все компоненты, процессы и все операторы в VHDL работают непрерывно и одновременно друг с другом. Каждый процесс и каждый компонент читает какие-то сигналы, в качестве входных данных, а в какие-то пишет результаты своей работы. За счёт такой организации вычислений ПЛИС получают огромное преимущество в вычислительной мощности, поскольку нет затрат на расшифровку команды, сохранение результата и переключение между командами. Но за такие преимущества приходится платить сложностью программирования.

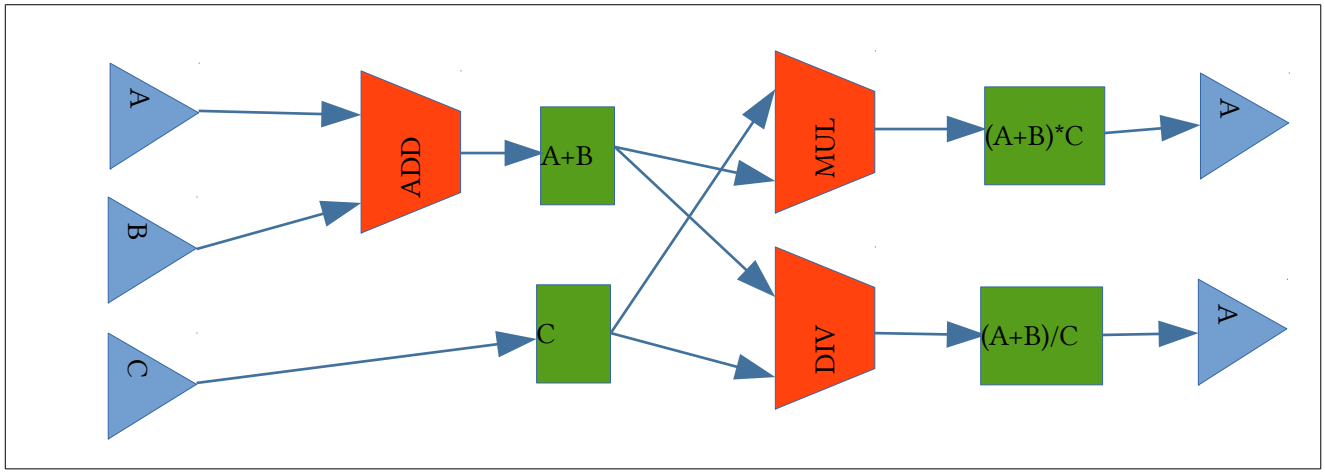


Иллюстрация 4: Пример-блоксхема компонента написанного на VHDL. Все вычислители работают одновременно друг с другом.

Синхронные и асинхронные процессы

Все процессы в VHDL делятся на синхронные и асинхронные. Синхронные процессы активируются при любом изменении входных данных и сразу же пересчитывают результат. Асинхронные же активируются только по фронту тактового сигнала и только сохраняют уже пересчитанный асинхронными процессами результат в регистры.

Пример асинхронного процесса

```
72      -- Вычисляет следующее состояние
73      NextStateProc: process (present_state, input_on, input_off)
74      begin
75          case present_state is
76              when Init =>
77                  if input_on = '1' then
78                      next_state <= Read1;
79                  else
80                      next_state <= Init;
81                  end if;
82              when Read1 =>
83                  next_state <= Read2;
84              when Read2 =>
85                  next_state <= Read3;
86              when Read3 =>
87                  next_state <= Read4;
88              when Read4 =>
89                  next_state <= Write1;
90              when Write1 =>
91                  if input_off = '1' then
92                      next_state <= Init;
93                  else
94                      next_state <= Read1;
95                  end if;
96              when others =>
97                  next_state <= Init;
98          end case;
99      end process;
```

Примеры синхронных процессов

```vhdl
-- Это синхронный процесс с асинхронным сбросом
FirstProcessName: process (clk, rst) is
begin
    if rst = '1' then
    elsif clk'event and clk = '1' then
    end if;
end process;

-- Это синхронный процесс с асинхронным сбросом
FirstProcessName: process (clk, rst) is
begin
    if rst = '1' then
    elsif Rising_edge(clk) then
    end if;
end process;

-- Это синхронный процесс с синхронным сбросом
process (clk, rst, Rg2_Ready) is
begin
  if Rising_edge(clk) then
    if (rst = '1' or Start = '1') then
    elsif Rg2_Ready = '1' then
    end if;
  end if;
end process;

-- Это синхронный процесс с синхронным сбросом
process (clk, rst, Rg2_Ready) is
begin
  if clk'event and clk = '1' then
    if (rst = '1' or Start = '1') then
    elsif Rg2_Ready = '1' then
    end if;
  end if;
end process;
```

ЗАДАЧА

Восстановить логическую таблицу по описанию задачи и разработать компонент реализующий данную логическую таблицу.

ВАРИАНТЫ

- А. Разработать машину для голосования.
- Б. Разработать схему, позволяющую несколькими разными переключателями включать и выключать свет независимо друг от друга.
- В. Разработать схему сумматора одноразрядных троичных чисел. Одним разрядом троичного числа считать два разряда двоичного: 00 – 0, 01 – 1, 10 – 2, 11 – NaN.

ЛИТЕРАТУРА

1. Архитектура ПЛИС (FPGA). МАРСОХОД. (http://marsohod.org/11-blog/265-fpga)
2.