АКАДЕМИЯ НАУК БЕЛАРУСИ
ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ
ИНСТИТУТ ТЕХНИЧЕСКОЙ КИБЕРНЕТИКИ

АЛГОРИТМЫ И ПОДСИСТЕМЫ АВТОМАТИЗИРОВАННОГО ЛОГИЧЕСКОГО ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ СБИС

Материалы по математическому обеспечению ЭВМ

Научный редактор доктор техн. наук,
профессор Р.Х. Садыхов

Минск 1994

УДК 658.512.22.011.56:621.382.049.771.14

Авторы-составители: А. А. Дудкин, В. А. Головко,
Г. Л. Муравьев, И. В. Качков, А. Г. Мачнев,
В. П. Махнист, В. Б. Гладыщук

Описываются алгоритмы и процедуры, предназначенные для логического проектирования цифровых схем в базисе элементов библиотек интегральных технологий. Приведенные процедуры позволяют автоматизировать процесс разработки принципиальных последовательностных и комбинационных схем, начиная с анализа и моделирования закона функционирования синтезируемых блоков. Процедуры синтеза дают возможность выполнить синтез схем логического управления как на жесткой, так и на микропрограммной логике, а также произвольных неоднородных отказоустойчивых структур для нейтрализации как эксплуатационных, так и производственных отказов. Входными данными для проектирования являются: описание проектируемой схемы на языке высокого уровня VHDL, требования к характеристикам схемы, тестовые наборы. Выходные данные представляют собой синтезированную схему и результаты моделирования. Предложены структуры подсистем САПР СБИС, которые являются частью системы высокоуровнего синтеза. Работа подсистем продемонстрирована на конкретных примерах.

Материал предназначен для специалистов, работающих в области логического проектирования цифровых СБИС, и для инженеров-разработчиков САПР.

Печатается по решению редакционно-издательского совета Института технической кибернетики АН Беларуси.

Рецензент доктор техн. наук П. Н. Бибило

© Институт технической кибернетики АН Беларуси, 1994

ВВЕДЕНИЕ

Современное состояние цифровой вычислительной техники и автоматики характеризуется широким применением больших и сверхбольших интегральных микросхем (БИС и СБИС), которое открывает новые возможности их совершенствования. Разработка СБИС невозможна без создания мощных систем автоматизированного проектирования (САПР), позволяющих использовать ЭВМ в качестве рабочего инструмента инженера-проектировщика цифровых схем на СБИС. Сегодня технические средства позволяют решить все этапы синтеза СБИС: алгоритмический, архитектурный, логический, схемотехнический и топологический. Это ставит на повестку дня создание сквозной САПР на рабочих станциях.

В материале по математическому обеспечению ЭВМ находит отражение этап логического проектирования цифровых СБИС, начиная с поведенческого описания на языке VHDL. Язык VHDL, предназначенный для описания сверхскоростных интегральных схем, получает все большее распространение в качестве входного языка САПР СБИС. Его отличительной особенностью является возможность описания на многих уровнях абстракции, что и обеспечивает его использование в САПР. Задание на проектирование содержит также требования к проектируемому устройству по быстродействию и сложности, и набор тестов. Результатом синтеза является принципиальная схема из элементов заданной интегральной технологии, оптимизированная по сложности и удовлетворяющая ограничению на быстродействие.

Первый раздел материала содержит описание конструкций языка VHDL, используемых в разрабатываемой САПР. Во втором разделе предложен способ преобразования исходного описания проекта в функционально-адекватный текст на языке высокого уровня, и приведен алгоритм событийного моделирования такого описания. Следующие три раздела содержат описания алгоритмов синтеза устройств управления в базисе элементов интегральных библиотек, контролепригодного синтеза произвольных цифровых схем, и также синтеза систолических вычислителей.

1. ИСПОЛЬЗОВАНИЕ ЯЗЫКА VHDL В ПРОЕКТИРОВАНИИ

Данный раздел содержит упрощенное описание основных конструкций языка VHDL, используемых в разрабатываемой САПР. Для более полного знакомства с языком необходимо обратиться к описанию стандарта языка VHDL [1] и учебному пособию [2].

1.1. Основные конструкции языка VHDL

Из четырех конструкций верхнего уровня языка VHDL: Entity, архитектуры (architecture body), пакета и конфигурации на любой стадии проектирования обязательно будут использоваться две: Entity и архитектура. Entity содержит описание интерфейса между проектируемым устройством и внешним миром. Архитектура содержит описание алгоритма работы проектируемого устройства. Так как для выполнения одной и той же функции могут быть предложены различные алгоритмы, а кроме того один алгоритм может быть записан различными способами, одному Entity может соответствовать несколько архитектур.

Формально конструкция Entity записывается следующим образом:

```
entity <имя_entity> is
    generic (<generic_list>);
    port    (<port_list>);
    <entity_declarative_part>
end <имя_entity>;
```

Конструкция generic_list используется для задания препроцессорных констант и другой статической информации, определяемой окружением. Она используется не всегда.

Конструкция port_list является основной. Она содержит описание портов, т.е. входов и выходов схемы.

Декларативная часть entity содержит объявления сигналов, констант, введенных пользователем новых атрибутов и типов данных, а также тексты подпрограмм и функций.

Все эти объекты могут использоваться в любой архитектуре,

относящейся к этому entity.

Кроме того, если некоторые из этих объектов используются в различных entity, их объявления можно выделить в отдельную конструкцию, называемую пакетом, и в декларативной части entity вместо объявлений поместить ссылку на этот пакет.

В первой версии САПР предполагается ограничиться использованием конструкций из некоторого подмножества языка VHDL. Примеры использования основных конструкций языка VHDL приведены ниже.

Здесь отметим только, что в списке port_list будут присутствовать порты типа in, out, inout. Будут использоваться такие объекты как константы, переменные, сигналы, атрибуты.

1.2. Стили описания архитектур

При описании архитектуры средствами языка VHDL могут использоваться 3 стиля, которые обеспечиваются разными конструкциями языка:

1) поведенческий;
2) потоковый; (не допускается в первой версии САПР)
3) структурный.

При поведенческом описании алгоритм записывается последовательными операторами (аналогичные которым имеются в большинстве языков программирования). Этот стиль используется для задания описания, поступающего на вход САПР.

При потоковом описании архитектура представляется в виде множества параллельных регистровых операций.

При структурном описании проект представляется в виде множества компонент, каждому из которых соответствует своя пара Entity-архитектуры. При этом должны быть указаны все связи между компонентами, т.е. для любого входа любого компонента (Entity) должно быть указано, с какого выхода какого компонента поступает сигнал на этот вход.

При этом для описания архитектуры компонента (проекта нижнего уровня) может использоваться любой из трех вышеперечисленных стилей и их комбинация. Таким образом может быть построено целое дерево компонент, на нижнем уровне которого используются уже известные (библиотечные) элементы. Как прави-

ло, так выглядит результат работы САПР. (Пример такого описания приведен ниже).

Поведенческий стиль описания является основным для пользователей САПР, так как именно он используется для создания VHDL-описания, поступающего на вход САПР.

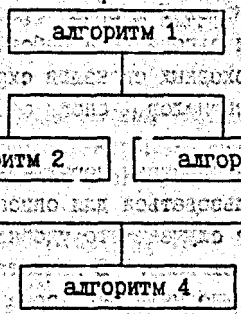
В общем случае поведенческое описание выглядит следующим образом:

```
architecture <имя_архитектуры> of <имя_entity> is
<architecture_declarative_part>
  begin
    process (sensitivity_list_1)
    <process_declarative_part>
      begin
      <sequential_statements_1>
      end process;
  .....
  process (sensitivity_list_n)
    <process_declarative_part>
      begin
      <sequential_statements_n>
      end process;
end <имя_архитектуры>;
```

Декларативная часть архитектуры может содержать такие же конструкции как и декларативная часть Entity. Процессы, как и другие операторы, допустимые в теле архитектуры, выполняются параллельно. В то же время в теле процесса, как и в теле функции или подпрограммы могут присутствовать только последовательные операторы. В основном эти операторы аналогичны операторам таких языков как Си и Паскаль. Кроме того в языке VHDL имеются последовательный оператор назначения сигнала и оператор ожидания (wait), которые позволяют описывать алгоритмы, работающие в реальном масштабе времени. Так как в языке VHDL нельзя смешивать последовательные и параллельные операторы (как скажем в языке МС-3), встает проблема описания алгоритмов, содержащих параллельно выполняемые действия. Эта проблема имеет следующие решения. Во-первых, так как последовательные операторы (кроме оператора wait) выполняются за нулевое время,

группа расположенных рядом операторов назначения сигнала стартует одновременно, т.е. эти назначения сигналов можно считать выполняющимися параллельно. Во-вторых, в более сложных случаях можно воспользоваться оператором wait.

Рассмотрим следующий алгоритм с параллельным выполнением алгоритмов 2 и 3.



На языке VHDl его можно описать следующим образом:

```
architecture  PAR of <имя_entity> is
   signal IND1,IND2     : bit:='0';
    begin
     M1:  process
                begin
                  <алгоритм 1>
                   IND1<= not IND1;
                  <алгоритм 2>
                   wait on IND2;
                  <алгоритм 4>
                   wait on <входные сигналы схемы>;
                   end process M1;
       M2:  process (IND1)
                begin
                  <алгоритм 3>
                   IND2<= not IND2;
                  end process M2;
                end PAR;
```

Процесс M1 не содержит списка чувствительности и он стартует первым. После выполнения алгоритма 1 изменится значение сигнала IND1. Так как этот сигнал входит в список чувствительности процесса M2, то после изменения его значения стартует процесс M2. В процессе M1 в это время будет выполняться алгоритм 2. После его выполнения процесс M1 остановится до изменения значения сигнала IND2. Но этот сигнал изменится только после выполнения алгоритма 3 в процессе M2. Таким образом после выполнения алгоритмов 2 и 3 начнется выполнение алгоритма 4. После выполнения алгоритма 4 работа приостанавливается до изменения значений входных сигналов схемы. Когда изменится какой-либо из входных сигналов, снова стартует процесс M1 и весь цикл повторится.

Отметим, что кроме описания параллельных алгоритмов, оператор wait может использоваться для описания генератора синхросигналов, хотя в ряде случаев это удобнее сделать при помощи оператора назначения сигнала. В качестве примера рассмотрим схему, которая должна выполнять некоторые действия с периодом в 1 с.:

```
entity PERIOD_ALG is
    generic (period: time:=1 sec);
    port (<port_list>);
end PERIOD_ALG;
architecture AR of PERIOD_ALG is
    signal   IND: bit:='0';
begin
P1:  process (ind)
        begin
          < алгоритм >
        end process P1;
P2:  process
        begin
          IND <= not IND after period;
        end process P2;
      end AR;
```

В этом примере через время, равное периоду 1 с., будет изменяться значение сигнала IND, а следовательно, стартовать процесс P1.

В качестве иллюстрации различных стилей описания архитектур рассмотрим схему из [2]. Эта схема производит подсчет числа единиц во входном битовом векторе A, имеющем длину, равную 3.

Описание ее интерфейса имеет вид:

```
entity ONES_CNT is
port (A: in bit_vector (0 to 2);
      C: out bit_vector (0 to 1));
end ONES_CNT;
```

Потоковое описание этой схемы использует параллельные операторы назначения сигнала, которые, как и остальные операторы в теле архитектуры, выполняются параллельно:

```
architecture POTOK of ONES_CNT is
  begin
    C(1)<=( A(1) and A(0) ) or (A(2) and A(0) )
                            or (A(2) and A(1) );
    C(0)<=( A(2) and not A(1) and not A(0) )
         or (not A(2) and not A(1)  and A(0) )
         or (A(2) and  A(1) and A(0) )
         or (not A(2) and A(1)  and not A(0) );
end POTOK;
```

Поведенческое описание является основным способом задания входной информации для САПР. В этом случае архитектура содержит один процесс или несколько процессов, выполняемых параллельно. В свою очередь каждый процесс содержит операторы, выполняемые последовательно. Поведенческое описание данной схемы может иметь, в частности, следующий вид:

```
architecture ALG1 of ONES_CNT is
  begin
    process (A)
      variable NUM: integer range 0 to 3;
      begin
        NUM:=0;
        for I in 0 to 2 loop
```

```
            if A(I)='1' then NUM:=NUM+1; end if;
        end loop;
    case NUM is
      when 0 => C<=B"00";
      when 1 => C<=B"01";
      when 2 => C<=B"10";
      when 3 => C<=B"11";
    end case;
    end process;
end ALG1;
```

Естественно существует много различных поведенческих описаний одной и той же схемы. В частности по потоковому описанию можно построить аналогичное поведенческое:

```
architecture ALG2 of ONES_CNT is
  begin
      process (A)
        begin
C(1)<=( A(1) and A(0) ) or (A(2) and A(0) )
                        or (A(2) and A(1) );
    C(0)<=( A(2) and not A(1)  and not A(0) )
                or (not A(2) and not A(1)  and A(0) )
                or (A(2) and  A(1)  and A(0) )
                or (not A(2) and A(1)  and not A(0) );
    end process;
    end ALG2;
```

Могут существовать различные структурные описания одного и того же алгоритма, так как при построении схемы могут использоваться различные элементы. Структурное описание нашей схемы может иметь следующий вид:

```
architecture STRUCT of ONES_CNT is
  component MAJ3
    port (X: in bit_vector (0 to 2); Z: out bit);
  component OPAR3
    port (X: in bit_vector (0 to 2); Z: out bit);
```

```
    begin
        COMP_1 : MAJ3 port map (A,C(1));
        COMP_2 : OPAR3 port map (A,C(0));
    end STRUCT;
```

Согласно этому описанию схема ONES_CNT состоит из двух: OPAR3 и MAJ3, первая из которых вычисляет нулевой бит результата, а вторая - первый. Чтобы полностью описать схему ONES_CNT, нужно описать схемы OPAR3 и MAJ3. Описание может быть любого типа, мы приведем структурное.

```
entity MAJ3 is
    port (X: in bit_vector (0 to 2)); Z: out bit);
end MAJ3;
  architecture AND_OR of MAJ3 is
     component AND2
         port (I1,I2: in bit; O out bit);
     component OR3
         port (I1,I2,I3: in bit; O out bit);
     signal A1,A2,A3: bit;
    begin
     G1: AND2 port map (X(0),X(1),A1);
     G2: AND2 port map (X(0),X(2),A2);
     G3: AND2 port map (X(1),X(2),A3);
     G4: OR3 port map (A1,A2,A3,Z);
    end AND_OR;
```

Схема MAJ3 состоит из трех экземпляров компонент типа AND2 и одного компонента типа OR3, связи между которым задаются картой портов. В свою очередь для компонент AND2 и OR3 приведем потоковое описание:

```
entity AND2
    port (I1,I2: in bit; O out bit);
end AND2;
    architecture PT of AND2 is
        begin
          O<=I1 and I2;
        end PT;
```

```
entity OR3
    port (I1,I2,I3: in bit; O out bit);
end OR3;
    architecture PT of OR3 is
        begin
            O<=I1 or I2 or I3;
        end PT;
```

Структурное описание схемы OPAR3 имеет аналогичный вид, но несколько сложнее:

```
entity OPAR3 is
    port (X: in bit_vector(0 to 2);Z out bit);
end OPAR3;
    architecture AND_OR of OPAR3 is
      component AND3
        port (I1,I2,I3: in bit; O out bit);
      component OR4
        port (I1,I2,I3,I4: in bit; O out bit);
      component INV
        port (I1: in bit; O out bit);
      signal A1,A2,A3,A4,B0,B1,B2 bit;
  begin
    G1: INV port map(X(0),B0);
    G1: INV port map(X(1),B1);
    G2: INV port map(X(2),B2);
    G3: AND3 port map(X(2),B1,B0,A1);
    G4: AND3 port map(B2,B1,X(0),A2);
    G5: AND3 port map(X(2),X(1),X(0),A3);
    G6: AND3 port map(B2,X(1),B0,A4);
    G7: OR4 port map(A1,A2,A3,A4,Z);
  end AND_OR;

entity INV
    port(I: in bit; O out bit);
end INV;
    architecture PT of INV is
        begin  O<=not I;
        end PT;
```

I2

```
entity AND3
      port( I1, I2, I3 : in bit; O out bit);
  end AND3;
  architecture  PT of AND3 is
     begin
       O<=I1 and I2 and I3;
     end PT;

entity OR4
      port( I1, I2, I3, I4 : in bit; O out bit);
  end OR4;
  architecture  PT of OR4 is
     begin
       O<=I1 and I2 or I3 or I4;
     end PT;
```

1.3. Описание автоматов

Описания автоматов на языке VHDL покажем на примерах.

Поведенческое описание автомата Мили, заданного таблицами переходов и выходов

	z1	z2	z3
x1	z2	z2	z1
x2	z3	z3	z3

	z1	z2	z3
x1	y1	y2	y1
x2	y2	y2	y1

имеет следующий вид:

```
entity MEALY is
 port (x : in integer range 1 to 2;     --входные состояния
       y : out integer range 1 to 2 ); --выходные состояния
end MEALY;
architecture AR of MEALY is
  constant period : time:=10ns; --тактовое время
  signal clk : bit:='0';
   signal z : integer range 1 to 3 :=1;   -- z - внутреннее
                                          --     состояние
```

```
begin
  CLOCK : process  -- описание синхронизации
          begin   clk <= not clk after period;
          end process CLOCK;
  MAIN : process (clk) --описание переходов
         begin
         case z is
           when 1 => case x is  -- 1 - начальное состояние
                       when 1 =>  z <= 2;
                                  y <= 1;
                       when 2 =>  z <= 3;
                                  y <= 2;
                     end case;
           when 2 => case x is
                       when 1 =>  z <= 2;
                                  y <= 2;
                       when 2 =>  z <= 3;
                                  y <= 2;
                     end case;
           when 3 => case x is
                       when 1 =>  z <= 1;
                                  y <= 2;
                       when 2 =>  z <= 3;
                                  y <= 1;
                     end case;
         end case;
       end process MAIN;
end AR;
```

Поведенческое описание этого же автомата с использованием пакетов и определяемых пользователем типов имеет следующий вид:

```
package M_PAC is
  type STATE is (Z1, Z2, Z3); --внутренние состояния
  type INPUT is (X1, X2);     --входные состояния
  type OUTPUT is (Y1, Y2);    --выходные состояния
end M_PAC;
```

```
entity MEALY is
 port (x : in INPUT;
       y : out OUTPUT);
 use M_PAC.all; --подключение объявлений из пакета M_PAC
end MEALY;
architecture AR of MEALY is
  constant period : time:=10ns; --тактовое время
  signal clk : bit:='0';
  signal z : STATE:= --текущее внутреннее состояние автомата
                ST1;--начальное состояние
begin
  CLOCK : process  -- описание синхронизации
          begin
          clk <= not clk after period;
          end process CLOCK;
  MAIN : process (clk) --описание переходов
          begin
          case z is
            when Z1 => case x is
                         when X1 =>  z <= Z2;
                                     y <= Y1;
                         when X2 =>  z <= Z3;
                                     y <= Y2;
                       end case;
            when Z2 => case x is
                         when X1 =>  z <= Z2;
                                     y <= Y2;
                         when X2 =>  z <= Z3;
                                     y <= Y2;
                       end case;
            when Z3 => case x is
                         when X1 =>  z <= Z1;
                                     y <= Y2;
                         when X2 =>  z <= Z3;
                                     y <= Y1;
                       end case;
          end case;
        end process MAIN;
      end AR;
```

Поведенческое описание автомата Мура заданного таблицами переходов и выходов

	z1	z2	z3
x1	z2	z2	z1
x2	z3	z3	z3

z1	z2	z3
y1	y2	y1

имеет вид

```
entity MUR is
 port (x : in integer range 1 to 2;      --входные состояния
       y : out integer range 1 to 2 );   --выходные состояния
end MUR;

architecture AR of MUR is
  constant period : time:=10ns;
  signal clk : bit:='0';
  signal z : integer range 1 to 3 :=1;
 begin
  CLOCK : process
          begin
          clk <='not clk after period;
          end process CLOCK;
  MAIN : process (clk)
          begin
          case z is
            when 1 => case x is
                        when 1 =>  z <= 2;
                        when 2 =>  z <= 3;
                      end case;
                      y <= 1;
            when 2 => case x is
                        when 1 =>  z <= 2;
                        when 2 =>  z <= 3;
                      end case;
                      y <= 2;
            when 3 => case x is
```

```
                    when 1 =>  z <= 1;
                    when 2 =>  z <= 3;
                  end case;
                  y <= 1;
        end case;
      end process MAIN;
    end AR;
```

Поведенческое описание этого же автомата с использованием пакетов и определяемых пользователем типов имеет следующий вид:

```
entity MUR is
 port (x : in INPUT;
       y : out OUTPUT);
 use M_PAC.all;
end MUR;

architecture AR of MUR is
 constant period : time:=10ns;
 signal clk : bit:='0';
 signal z : STATE:=ST1;
begin
 CLOCK : process
         begin
         clk <= not clk after period;
         end process CLOCK;
 MAIN : process (clk)
         begin
         case z is
           when Z1 => case x is
                        when X1 =>  z <= Z2;
                        when X2 =>  z <= Z3;
                      end case;
                      y <= Y1;
           when Z2 => case x is
                        when X1 =>  z <= Z2;
                        when X2 =>  z <= Z3;
                      end case;
```

```
                    y <= Y2;
            when Z3 => case x is
                         when X1 =>  z <= Z1;
                         when X2 =>  z <= Z3;
                       end case;
                       y <= Y1;
          end case;
        end process MAIN;
      end AR;
```

Возможны и другие представления конечных автоматов [2,с.66-68]. Выбранное нами представление обусловлено текущей рабочей версией транслятора VHDL-описания.

1.4. Подсистема лингвистического обеспечения

Подсистема лингвистического обеспечения включает в себя процедуры анализа, редактирования и трансляции VHDL-описания во внутреннее представление проекта.

Первоначальный анализ VHDL-описания осуществляется транслятором [4]. В отличие от трансляторов с языков программирования, настоящий транслятор переводит текстовое описание не в объектный код, а во внутреннее представление (ВП) проекта, которое является входным для программ САПР. Перевод VHDL-описания во внутреннее представление осуществляется в несколько этапов программами транслятора. На первом этапе синтаксический анализатор осуществляет синтаксическую проверку VHDL-описания. При отсутствии ошибок строится дерево разбора, а при их наличии - выдается сообщение об ошибке и вызывается текстовый редактор для ее исправления. На втором этапе семантический анализатор осуществляет семантическую проверку (используя дерево разбора) и решает проблему видимости, т.е. строит таблицу имен, в которой для каждого идентификатора указывается, в какой декларации он объявлен, а следовательно, к какому объекту или типу он относится. На третьем этапе конвертор осуществляет перевод дерева разбора во внутреннее представление. Полученное ВП может либо непосредственно использоваться проектными процедурами, либо записываться в ба-

зу данных. Внутреннее представление схемы используется на всех этапах ее проектирования. Для визуализации промежуточных результатов используется перевод ВП на язык VHDL, осуществляемый реверсивным анализатором. Все программные модули транслятора реализованы на языке программирования С.

Язык ВП разрабатывался с учетом следующих требований:

- он должен быть ориентирован на методы его обработки, преобразования и хранения в БД;
- поскольку входным языком САПР является VHDL, язык ВП должен обеспечивать однозначное представление конструкций VHDL, используемых в САПР, а в перспективе и всего VHDL;
- он должен предусматривать рациональное использование машинной памяти;
- он не должен накладывать дополнительные ограничения на разрабатываемые проекты;
- он не должен ограничивать переносимость САПР на другие ЭВМ и в другие операционные среды.

С учетом вышесказанного внутреннее представление проекта реализуется в виде сети структур языка С, размещаемых в оперативной памяти.

В настоящее время во внутреннем представлении используются С-структуры девяти типов (STRE, STRA, STRC, STRPAC, STRP, STRO, STRN, STRL, STR2). Структуры STRE, STRA, STRC, STRPAC описывают объекты верхнего уровня языка VHDL, т.е. Entity, архитектуру, конфигурацию и пакет, соответственно. Эти структуры содержат те же поля, что и соответствующие им конструкции языка VHDL, т.е. в эти структуры записываются имена объектов (а в STRA и STRC кроме того имя Entity). Остальные поля этих структур содержат адреса первых элементов списков структур нижнего уровня (STRP, STRO, STRN, STRL, STR2), описывающих декларативные части и тела объектов. Каждый элемент такого списка в последнем поле содержит адрес следующего элемента этого списка, а при его отсутствии - null.

Структура STRP предназначена для описания подпрограмм, функций, блоков, процессов, STRO - для описания операторов и некоторых других объектов, STRN - для описания простых имен, STRL - литералов, STR2 - выражений, диапазонов, сложных имен и других вспомогательных конструкций.

Все структуры внутреннего представления содержат указате-

ли на другие структуры, поэтому внутреннее представление любого VHDL-описания имеет вид сети (а если не принимать во внимание ссылки на имена и литералы, то - дерева). Поскольку имя появляется в списке имен один раз, а использоваться может неоднократно, древовидность нарушается. Длина идентификаторов во внутреннем представлении предполагается ограниченной, а длина строковых литералов - нет. Литералы типа bit_vector в настоящей версии ЕЛ хранятся как символьные строки, т.е. каждый бит строки занимает один байт. В будущем возможно использование какого-либо упакованного формата.

2. МОДЕЛИРОВАНИЕ ПОВЕДЕНЧЕСКОГО ОПИСАНИЯ СБИС

Задачи моделирования цифровой аппаратуры (ЦА) в САПР СБИС наиболее полно исследованы на уровнях от логического до схемно-конструкторского проектирования. В тоже время в последнее десятилетие основные усилия по разработке САПР СБИС переместились в область высокоуровневого проектирования [5-8], где объект проекта описывается заданием входов-выходов и соответствующего алгоритма преобразования входных значений в выходные. При этом переменные и операторы алгоритмического описания не обязательно соответствуют внутренней структуре проекта - регистрам и направлениям регистровых передач.

Высокоуровневый синтез обеспечивает [8] возможность полной автоматизации процесса проектирования ЦА и потенциальное сокращение цикла проектирования. Кроме того, разработчик имеет дело со спецификациями проекта существенно меньшей размерности и располагает большими возможностями по их верификации.

Для описания проектов чаще используются процедурные языки [8] со стандартными управляющими конструкциями. Модульность достигается иерархически организованными процедурами. Это языки высокого уровня Simscript, Фортран, Ада, Оккам, расширение языков Паскаль и Си [8,9]. Реже применяются функциональные языки как Signal и непроцедурные языки типа Lisp, Пролог, описывающие связи между сигналами. Сигналы имеют уникальные имена и задаются с помощью уравнений, определяющих их значения через значения входных сигналов. Третья группа - специализированные языки спецификации проектов верхнего уровня, обладающие

развитыми алгоритмическими средствами и средствами описания параллельных процессов: ISPS, ELLA, стандарт промышленного использования - язык моделирования сверхскоростных СБИС VHDL [10], версия языка - 2.X фирмы ZyCad [11], расширение языка VHDL - системы VAL [12], языки регистровых передач, язык иерархического функционально-логического моделирования МЗЗ и другие, образующие основу поведенческой спецификации проектов [13] и промышленно используемые в системах типа ViewLogic [14], Utile[15].

Таким образом, в связи с развитием средств синтеза и наметившейся тенденции автоматизации проектирования ЦА, начиная с верхнего, системного уровня, особую роль приобретает моделирование устройств, закон функционирования которых задается алгоритмически (в частном случае формулой, системой логических уравнений) либо описывается структурой и алгоритмом. Все сказанное обусловливает актуальность разработки принципов и методов организации поведенческого моделирования БИС, СБИС, описанных на алгоритмическом уровне.

При выполнении моделирования приняты следующие концепции: отказ как от интерпретации VHDL-описания проекта, так и от генерации загрузочного кода, и использование промежуточной трансляции описания проекта СБИС в адекватный в функциональном отношении текст на языке высокого уровня с последующим получением его загрузочного модуля; выбор в качестве средства внутреннего представления языка Си с развитыми вычислительными средствами и эффективным транслятором.

2.1. Требования, предъявляемые к средствам поведенческого моделирования СБИС

Поведенческое моделирование проектов СБИС должно обеспечивать: 1) на предпроектном этапе входной контроль спецификации проектируемого устройства на заданных разработчиком тестовых данных, возможность ее отладки и коррекции; 2) в ходе реализации проектных процедур САПР прогнозирование функциональных характеристик проекта и оценку степени их соответствия критериям оптимизации.

Анализ промышленных САПР и их проектов [5,14,16-22] пока-

зывает, что для этого системы моделирования СБИС алгоритмического уровня должны располагать: 1) средствами формальной спецификации проектов ЦА верхнего уровня; 2) анализаторами для лексического, синтаксического и семантического контроля спецификации проектов; 3) средствами контроля корректности проектов ЦА, в том числе, средствами описания утверждений для автоматического доказательства правильности проектов, трансляции исходных спецификаций на языках высокого уровня в формальные спецификации, пригодные для доказательства корректности проектов; 4) средствами преобразования исходного формального описания проекта в некоторый внутренний стандарт, пригодный для осуществления моделирования, и средствами "прокрутки" - выполнения описания проекта в статическом или динамическом (трассовом) режиме.

Язык, применяемый для описания закона функционирования устройства, помимо обычных требований надежности, модифицируемости и т.д., должен предоставлять пользователю адекватные изобразительные средства и обеспечивать создание эффективного моделирующего кода. Для этого необходимо наличие в языке средств, поддерживающих: 1) особые типы данных, как сигналы, битовые векторы произвольной длины, векторы многозначных сигналов; 2) операции над битовыми векторами, их частями, элементами и векторами в специфических алфавитах и таблицах истинности многозначных сигналов, а также особые операции типа размножения разрядов; 3) возможность спецификации параллельных процессов с помощью специальных операторов, обеспечивающих их запуск, тактирование и синхронизацию; 4) иерархию программных компонентов: процессов, блоков, пакетов, архитектур, конфигураций, интерфейсов и т.д.; 5) иерархическое проектирование СБИС с возможностью параллельного ведения нескольких проектов, построения иерархий проектов, библиотек моделей различной степени детализации, доступ к базам данных, автоматическое накопление и анализ результатов моделирования.

Анализ упомянутых выше языков позволяет остановиться на минимальном наборе изобразительных средств, необходимом и достаточном для моделирования проектов устройств от поведенческого до регистрового уровня.

Основной программный компонент такого набора - блок B1, соответствующий функционально-законченному описанию устройства

или его части. Блок представляется интерфейсом и множеством процессов {Pi| i=1..N}. Процессы задают фундаментальную часть алгоритма, в них определяются действия, выполняемые над переменными. Каждый процесс представляет комбинацию множества операторов, соединенных параллельно-последовательно. Процесс может быть представлен ориентированным графом G=(S,D) с вершинами, составляющими множество операторов S={Si}, где Si - подмножество операторов, выполняемых в одном такте, и дугами D={dij}, отображающими направление передачи управления в графе. Любой блок может раскрываться через вложенные блоки, что обеспечивает иерархию, композицию структурного и функционального описаний.

Основные функции, реализуемые операциями и операторами языка, составляют множество G={A,B,W,F,I,U,C}, где операторы присваивания A, инициирования процессов F, вызова блоков B, ожидания W для описания асинхронных процессов относятся к операторам, выполняемым за один такт, а условный оператор I к многотактным операторам; тип такта определяется операцией следования U. Множество допустимых операций обозначено C. Кроме операторов типа A и F, выполняемых как в одиночном, так и в групповом режиме (параллельно в одном такте), все остальные однотактные операторы могут выполнятся только в одиночном режиме. Базовые функции, легко отображаемые на языках высокого уровня и потому служащие основой для интерпретации других операторов в эквивалентных преобразованиях, реализуются одиночными операторами типа A, W, B.

При построении сложных функциональных описаний, содержащих параллельно-последовательные процессы, используются операторы типа F. При выполнении оператора F запускается новый процесс, при завершении которого возобновляется прерванный процесс с оператора, следующего за оператором вызова процесса.

Ниже описаны типовые конструкции языка в нотации Бэкуса-Наура

```
<Описание_блока> ::= <Заголовок_блока> <Декларации_блока>
                     <Процессы_блока>
<Заголовок_блока> ::= ИДЕНТИФИКАТОР_блока
                      (<Интерфейс_блока>)
<Интерфейс_блока> ::= <Список_формальных_параметров>
```

```
<Список_формальных_параметров> ::= <Тип_параметра>
                                   ИДЕНТИФИКАТОР_параметра
                          {,<Тип_параметра> ИДЕНТИФИКАТОР_параметра}
<Тип_параметра> ::= ВХОДНОЙ| ВЫХОДНОЙ| ВХОДНОЙ-ВЫХОДНОЙ
<Декларации_блока> ::= {<Строка_описания>}
<Строка_описания> ::= <Описание_массива>
                        |<Описание_константы>
                        |<Описание_переменной>
<Описание_массива> ::= {ИДЕНТИФИКАТОР_массива
                          [ЦЕЛОЕ_индекс_1: ЦЕЛОЕ_индекс_2]}
<Процессы_блока> ::= {<Описание_процесса>}
<Описание_процесса> ::= ИДЕНТИФИКАТОР_процесса
                          <Операторная_часть_процесса>
<Операторная_часть_процесса> ::= <Оператор>
                  |<Оператор> {<Операция_следования> <Оператор>}
<Операция_следования> ::= ПОСЛЕДОВАТЕЛЬНО| ПАРАЛЛЕЛЬНО
<Оператор> ::= <Оператор_присваивания>|<Условный_оператор>
                 |<Оператор_ожидания>
                 |<Оператор_инициирования_процесса>
                 |<Оператор_вызова_блока>
<Оператор_присваивания> ::= ИДЕНТИФИКАТОР := <Выражение>
<Выражение> ::= <Терм> {ОПЕРАЦИЯ <Терм>}
<Оператор_ожидания> ::= ЖДАТЬ(<Параметры_ожидания>)
<Параметры_ожидания> ::= ЦЕЛОЕ_число_тактов
                           |<Список_процессов>
                           |<Условие_ожидания>
<Список_процессов> ::=
          ИДЕНТИФИКАТОР_процесса[{,ИДЕНТИФИКАТОР_процесса}]
<Условие_ожидания> ::= Булево_выражение
<Оператор_инициирования_процесса> ::=
                           ИДЕНТИФИКАТОР_процесса.
```

2.2. Принципы организации поведенческого моделирования СБИС

Наличие в изобразительных средствах алгоритмического уровня инструментария для описания и управления параллельными процессами требует для его поддержки соответствующих аппарат-

ных средств с возможностью реализации многозадачных режимов либо программных средств, имитирующих параллельные процессы путем их последовательного моделирования на однопроцессорной ЭВМ. В связи с распространением в промышленных САПР СБИС ПЭВМ и мощных станций типа VAX, APOLLO, SUN актуальной представляется разработка принципов построения соответствующих программных средств.

Двумя крайними подходами являются - построение системы интерпретации либо компилирующей системы. Необходимая эффективность моделирующего кода при относительной простоте системы моделирования может быть достигнута: 1) отказом как от интерпретации спецификации проекта, так и прямой генерации загрузочного кода и использованием метода промежуточной трансляции описания закона функционирования устройства в адекватный в функциональном отношении текст на некотором внутреннем языке с последующим получением загрузочного модуля; 2) выбором в качестве средства внутреннего представления языка высокого уровня с развитыми вычислительными средствами и эффективным транслятором, как, например Си, Паскаль, Ада.

Таким образом, функционально система моделирования должна содержать: 1) входной препроцессор для трансляции исходного описания; 2) генератор промежуточного текста на языке высокого уровня; 3) монитор моделирования, составляющий ядро системы моделирования.

При этом само моделирование выполняется в два этапа: на первом по результатам лексического, синтаксического и семантического разбора описания блока, представленным в терминах структур данных языка высокого уровня, создается текст программного компонента, адекватный исходному описанию. При наличии в описании вложенных блоков этот этап повторяется для каждого блока; на втором этапе, выполняемом многократно, производится собственно имитационное моделирование блоков, загрузочные коды которых получают предварительно обработкой транслятором языка высокого уровня. Моделирование производится с использованием функциональных тестов и сводится к порождению порядка инициирования блоков и процессов в блоках в соответствии с признаками их активности и алгоритмом функционирования устройства.

2.3. Способ генерации функционально-эквивалентного программного компонента

После обработки исходной спецификации препроцессором каждый процесс синтаксически представляется деревом термов Gt =(St,Dt), где терм - константа, переменная, некоторая функция из набора базовых функций языка G либо ссылка на другие термы. Поэтому основная операция, используемая при работе с процессом, просмотре дерева термов может быть описана рекурсивно как операция N(t) - анализировать терм t:

если терм t<-G и t^=F, то перейти на левый терм t=tл, выполнить N(t); перейти на правый терм t=tп, выполнить N(t);

иначе, если t=F, то обработать терм-"вызов процесса";

иначе обработать терм-операнд.

Перед использованием результатов разбора блока для генерации исполнимого текста выполняется его преобразование с целью слияния с каждым вызывающим процессом тех дочерних, которые используются только для декомпозиции сложного функционального описания (т.е. не ссылаются в операторах ЖДАТЬ и не применяются для организации циклических вычислений). Алгоритм преобразования дерева термов приведен ниже:

1) просмотреть термы tj (j=1,2,..) деревьев каждого из процессов P={Pi}. Зафиксировать количество {ni|i=1..N} вызовов каждого процесса в соответствии с правилом:

ni= ni+1, если tj=Fi;

значение>1, если tj=ЖДАТЬ(Fi);

2) сформировать базовое множество процессов Po={Pi|Vni>1} +P1 и множество P'=P\Po, где P1 - корневой процесс;

3) просмотреть множество деревьев процессов Po и выполнить преобразование Po->Po' в соответствием с правилом: если терм tj=Fi, где Pi<-P', то заменить терм tj деревом термов процесса Pi и если P'\Pi^=o, то продолжить п.3;

4) разметить линейные участки - ветви графов процессов множества Po'. Для этого введем признаки а - начала процесса, б - начала ветвления, г - завершения ветвления (слияния): Начальные значения а=TRUE, б,г=FALSE. Обозначим G1={A,B,W}, G2={U,F}. Необходимо просмотреть графы процессов множества Po', используя рекурсивную операцию N'(t):

если ti<-G1 и а!б!г, то сбросить соответствующий признак

в FALSE, установить признак ветви;

если ti<-G2, то перейти на левый терм ti:=tл, выполнить N'(ti); перейти на правый терм ti:=tп, выполнить N'(ti);

если ti=I, то г,а:=FALSE, б:=TRUE, перейти на левый терм ti:=tл, выполнить N'(ti); г:=FALSE, б:=TRUE, перейти на правый терм ti:=tп, выполнить N'(ti), г:= TRUE, б:=FALSE.

Преобразованные графы используются для генерации текста. В процессе генерации блок B1 заменяется системой <P*,M>, где M - процедура, управляющая моделированием блока, а P* - множество процедур, функционально-адекватных соответствующим процессам блока (рис.1).

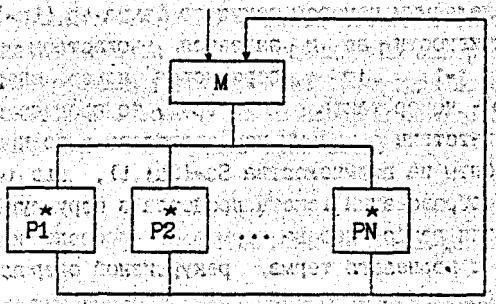


Рис.1. Структура генерируемого текста для блока B1

При моделировании блока предполагается периодическая инициация со стороны процедуры M процедур процессов P* и поочередное выполнение текущих тактов активных процессов. При этом, в зависимости от результатов выполнения P* выставляются соответствующие признаки, учитываемые процедурой M. Каждый процесс может быть описан программно в терминах управляющей структуры типа ВЫБОР, присутствующей в большинстве языков программирования, поскольку при каждом запуске продвижение вычислительного процесса выполняется в рамках его текущего такта. Операторы каждого варианта ВЫБОРа относятся к одному такту процесса; а выбор группы исполняемых операторов процесса определяется номером его активного такта. Процедура, соответствующая процессу, выглядит следующим образом:

```
ВЫБОР по номеру такта t
 СЛУЧАЙ t=1
   {операторы такта 1}
   Модификация t
 СЛУЧАЙ t=2
   {операторы такта 2}
   Модификация t
   ...
 СЛУЧАЙ   t=m
   ...
ВСЕ-ВЫБОР.
```

Пусть каждый p-й процесс характеризуется состоянием, однозначно определяемым номером текущего такта tp (tp=1,...,tm), признаком активности ap и задается множеством операторов Sp={Sj,t}, где j=1,...,jm - порядковый номер оператора, а t=0,1,..., tm - номер такта. Если tp=0, то процесс пассивен и ap=0. В соответствии с типами все операторы p-го процесса могут быть разбиты на подмножества Sa={Sai,t}, где i=1,...,am, a<-G. При преобразовании текста процесса в структуру типа ВЫБОР, выполняется разбор дерева термов Gt посредством применения, начиная с корневого терма, рекурсивной операции N(t) по следующим правилам:

1) одиночный оператор Sai,t (ai<-G1) преобразуется в однотактный оператор ВЫБОРА

СЛУЧАЙ t: (gai,t;tp:=tp+1),

где gai,t - эквивалент оператора Sai,t на языке высокого уровня, например, ПРИСВОИТЬ (для оператора типа A), ВЫЗВАТЬ ПОДПРОГРАММУ (для оператора типа B), ЕСЛИ (f) ТО tp:=tp+1 (для оператора W ожидания условия f);

2) одиночный оператор Sai,t(k) (ai=F) вызова k-го процесса либо множества процессов K преобразуется в последовательность двух тактов оператора ВЫБОР

СЛУЧАЙ t: ({g'ai,t(k)};tp:=tp+1,

СЛУЧАЙ t+1: (g"ai,t;tp:=tp+1),

где первому множеству операторов соответствует множество операторов {tk:=1} активизации процессов K, а g"ai,t соответствует оператор gw(K) ожидания завершения процессов K;

3) условный оператор Sai,t (ai=I) преобразуется в группу тактов оператора ВЫБОР, первым из которых является оператор

ЕСЛИ (f) ТО tp:=tp+1

ИНАЧЕ tp:=ta,

где ta - номер такта, соответствующего альтернативной ветви оператора, f - условие;

4) группа из n одиночных операторов (Sai,t"Sa2,t"..." San,t), выполняемых параллельно (ai<-{A,F}) отображается двумя тактами оператора ВЫБОР

СЛУЧАЙ t: ({gai,t|ai=A};{g'ai,t|ai=F};tp:=tp+1),

СЛУЧАЙ t+1: (g"ai,t;tp:=tp+1),

в частном случае, если Sf=o, одним тактом оператора ВЫБОР

СЛУЧАЙ t: ({gai,t|ai=A};tp:=tp+1).

Каждый процесс завершается дополнительным тактом, в котором выставляется признак его завершения. По завершении текущего такта p-го процесса управление из процедуры процесса передается в монитор M вместе с новым номером текущего такта tp и множеством номеров инициируемых процессов.

2.4. Реализация поведенческого моделирования

При моделировании блока (блоков) используется событийный подход. В качестве событий рассматриваются: инициирование нового процесса; завершение обработки активного процесса; завершение обработки текущего такта процесса.

Состояние каждого блока определяется как очередью Q (множеством Ри) инициированных (активных и ждущих) процессов, так и состояниями самих процессов, однозначно определяемыми номерами текущих тактов, подлежащих исполнению. Очередь инициированных процессов представляет множество троек Q = {(p,t,a)}, характеризующих процесс, где p = 1,...,pm - номер процесса, t = 1,...,tm - номер текущего такта и ap = {0,1} - признак активности процесса p. Вершина очереди определяется указателем q.

Множество Ри={PiVi(ai=1)} строится следующим образом:

1) вновь инициированный процесс помещается в конец очереди;

2) при завершении обработки процесс исключается из очереди;

3) просмотр очереди выполняется с конца, что обеспечивает своевременный учет в ждущих процессах в том же активном такте события - завершение дочерних процессов.

Укрупненный алгоритм моделирования приведен ниже:

1) активизировать корневой процесс блока (aa:=1), внести его в множество инициированных процессов Ри=Ро и поместить в очередь Q1 :=(0,1,1); q:=1; i:=1;

2) выбрать процесс Pn с параметрами (tn,an), стоящий в очереди Q под номером i (Qi=(n,tn,an));

3) если в текущем такте процесс Pn пассивен (an=0), то перейти к п.5;

иначе (an=1) выполнить операторы текущего такта tn процесса Pn, модифицировать номер такта процесса tn, сбросить признак активности процесса в текущем такте an:=0;

4) анализировать результаты выполнения процесса Pn в текущем такте:

если процесс завершился (tn>tm), то удалить его из множества активных процессов Ри:=Ри\Pn и очереди Q, q:=q-1;

иначе, если инициировались новые процессы и зафиксирован нерекурсивный вызов процесса Pk^<-Ри, то активизировать каждый новый процесс k (ak:=1; tk:=1), внести процесс в множество активных процессов Ри:=Ри U Pk и в очередь q:=q+1; Qq:= (k,1,1);

иначе активизировать новые процессы k (ak:=1; tk:=1);

5) если просмотр очереди Q не завершен (i^=1), то перейти к следующему элементу очереди i:=i+1 и выполнить п.2;

иначе, если очередь Q просмотрена (i=1) и в текущем такте моделирования инициировались новые процессы, т.е. существует Qj с aj=1, то возобновить цикл просмотра с конца очереди Q i:=q и перейти к п.2;

иначе, если i=1, нет других активных процессов в рассматриваемом такте и Q^=o, то восстановить признаки активности всех процессов в очереди Q (aj:=1Vj(Pj<-Ри)), i:=q и перейти к п.2;

иначе (Q=o) завершить моделирование.

При моделировании в статическом режиме в п.1 указанного алгоритма задаются значения входных параметров блока через его

интерфейс, а по завершении обработки значения выходных переменных выводятся через интерфейс блока пользователю для дальнейшего анализа. В динамическом режиме ввод значений параметров, отображающих временную диаграмму входа, производится на каждом такте в п.2 указанного алгоритма, там же выводятся и фиксируются значения выходов, образующие временную трассу результатов. Полученная диаграмма сличается с эталонной.

2.5. Обработка битовых векторов

При описании цифровых устройств на поведенческом уровне в терминах высокоуровневых языковых средств, таких как VHDL, широко используются атрибуты типа битовый вектор. Эффективность поведенческого моделирования во многом определяется скоростью обработки битовых векторов [23], особенно при описании проектов на регистровом уровне [24,25]. Поскольку в большинстве случаев классические языки программирования высокого уровня, например, Си, Паскаль, применяемые для поддержки языка моделирования, не обладают средствами обработки битовых векторов произвольной длины, а изобразительные средства VHDL предусматривают ограниченный спектр операций над ними, то существует необходимость разработки эффективных операций над этим типом данных.

Требования к обработке битовых векторов

Тип данных bit_vector, составляющий вместе с типами: bit, boolean основу двоичной арифметики VHDL, относится к предопределенным индексированным типам, описание которых дано в пакете STANDARD [3]. Значениями типа bit_vector являются одномерные массивы значений типа bit, индексированные значениями типа natural:

```
subtype natural is integer range 0 to integer'high;
type bit is ('0','1');
type bit_vector is array (natural range <>) of bit.
```

В VHDL над переменными типа bit_vector, bit допустимы следующие предопределенные операции: а) логические and, or,

nand, nor, xor, not; б) отношения =, /=, <, <=, >, >=, возвращающие результат типа boolean; в) аддитивная операция конкатенации &; г) вырезка (slise) - получение непрерывного подмножества (сечения) переменной типа bit_vector; д) присваивание, когда в левой части операции может содержаться переменная типа bit_vector или ее сечение.

Остальные операции VHDL, допустимые для числовых типов, в том числе арифметические, не являются предопределенными для типа bit и bit_vector, т.е. не поддерживаются транслятором VHDL и реализуются самостоятельно пользователем с помощью аппарата функций и процедур VHDL, что универсально, но может отрицательно сказаться на эффективности моделирования при неоптимальных алгоритмах имитации отсутствующих операций. Кроме того, возможны варианты синтеза СБИС, когда исходный проект специфицируется в терминах операций над битовыми векторами, которые в свою очередь основываются на соответствующих библиотечных компонентах и не подлежат перепроектированию. Поэтому представляется целесообразным расширить состав предопределенных операций битовой арифметики VHDL до типового набора операций регистрового уровня [26].

Представление переменных типа bit_vector в модели должно обеспечивать: а) корректное с точки зрения синтаксиса VHDL использование величин типа bit_vector во всех допустимых конструкциях языка, в том числе корректное взаимодействие объектов типа bit_vector с объектами других типов (bit, boolean, array of bit, array of boolean); б) минимальное по времени выполнение базовых операций двоичной арифметики, что зависит от числа преобразований типов и возможности использования для выполнения операций соответствующих машинных команд; в) реализацию формул, где могут быть использованы смеси битовых векторов разной длины, битовые и литеральные значения. Для этого термы формул и результаты выполнения операций в формулах должны приводиться к одному и тому же типу; г) возможность контроля результатов выполнения операций - контроль изменения диапазона значения в результате операций, контроль ситуации потери значимости, переполнения промежуточных и конечного результата; д) поддержку аппарата атрибутов переменных индексированного типа [23].

Внутреннее представление битовых векторов

Переменные типа bit_vector описываются в VHDL как

имя_переменной : bit_vector [i1 to i2]

или

имя_переменной : bit_vector [i1 downto i2],

где i1, i2 - границы индексов.

В качестве базового формата хранения переменной типа bit_vector предлагается упакованный формат, когда для представления отдельного бита переменной используется один бит в массиве элементов базового типа языка высокого уровня, что обеспечивает необходимую эффективность реализации. Тип базового элемента определяется аппаратной средой, а именно, размером слова, являющегося единицей обработки при выполнении машинных команд. Для IBM PC в среде Си таким элементом является слово длиной 16 бит (тип unsigned int). Размер памяти, выделяемой под переменную типа bit_vector в элементах базового типа,

$$AL = \begin{cases} A1 / T1 & \text{, при делении нацело;} \\ A1 / T1 + 1 & \text{, в остальных случаях.} \end{cases}$$

Здесь T1 - длина базового элемента в битах, A1=(|i1-i2|+1) - длина информационной части переменной в битах. Соответствие полей бит вектора длиной A1 и базовых элементов хранения представлено на рис. 2. При этом справедливо соотношение AL * T1 = A1 + Ad , где Ad -смещение битового вектора (количество бит) от начала буфера, выделенного под него. Начало буфера определяется адресом первого базового элемента хранения Ab.

Таким образом, само значение типа bit_vector хранится в массиве базовых элементов Си, а формат хранения полностью определяется данными, содержащимися в служебной структуре.

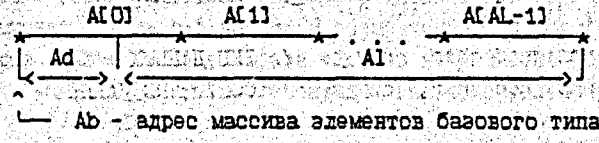


Рис. 2. Соответствие полей бит вектора и базовых элементов хранения

Работа с переменной типа bit_vector включает следующие аспекты: 1) описание базовых типов; 2) объявление переменной и инициализацию ее служебной информации; 3) инициализацию значения битовой переменной; 4) описание формул.

Такое внутреннее представление обеспечивает эффективную поддержку данных типа битовый вектор и классических типов данных - integer (ЦЕЛЫЙ), bit (БИТ), boolean (ЛОГИЧЕСКИЙ) в части использования памяти и скорости выполнения арифметических операций.

Физическая реализация внутреннего представления битовых векторов

При использовании данных типа bit_vector следует учитывать следующее: а) данные описываются с помощью структуры define_bit_vector (dbv), приведенной ниже. При вызове подпрограмм передаётся (возвращается) адрес этой структуры; б) данные самоопределяемые, define_bit_vector содержит поле ТИП_ДАННЫХ, обеспечивающее совместимость с типом данных boolean. Обработка данных строится на использовании средств (подпрограмм), обеспечивающих автоматическое преобразование их типов; в) структура define_bit_ vector содержит поле СОСТОЯНИЕ_ДАННЫХ, указывающее на корректность данных; г) в качестве поля ответа используется рабочий буфер, который сбрасывается по завершении оператора; д) БАЗА_ДАННЫХ служит для указания на начало поля, предназначенного для размещения данных. СМЕЩЕНИЕ_ДАННЫХ доопределяет место расположения битового вектора. ДЛИНА_ДАННЫХ указывает количество разрядов битового вектора.

```
#define define_bit_vector dbv
#define bit_vector dbv
struct dbv      /* ОПИСАНИЕ_БИТ_ВЕКТОРА */
  {
  unsigned char   dbv_t;  /* ТИП_ДАННЫХ */
  unsigned char   dbv_s;  /* СОСТОЯНИЕ_ДАННЫХ */
  union           dbv_b   /* БАЗА_ДАННЫХ */
    {
     char         *p;     /* указатель на начало
                             ПОЛЕ_ДАННЫХ типа a */
```

```
    unsigned int  w;        /* ПОЛЕ_ДАННЫХ типа w */
    unsigned long d;        /* ПОЛЕ_ДАННЫХ типа d */
    unsigned char c[4];     /* ПОЛЕ_ДАННЫХ типа c */
    } ;
  unsigned int    dbv_d;  /* СМЕЩЕНИЕ_ДАННЫХ */
  unsigned int    dbv_l;  /* ДЛИНА_ДАННЫХ */
  } .
```

При использовании данных типа boolean следует учитывать: а) тип boolean совместим с типом bit_vector длиной 1 для работы с ним как со строкой бит при наличии соответствующего описания (dbv); б) для размещения данных типа boolean требуется один байт (true - 00000001B, false - 00000000B).

При работе с битовыми данными используется указатель на структуру define_bit_vector, которая содержит всю информацию о битовой переменной и обеспечивает мобильность, самоопределение данных и реализацию операций пересылки путем изменения описательных структур. Сама переменная типа bit_vector расположена в некоторой области памяти, выделенной для нее транслятором либо в рабочей области.

Указанная структура обладает следующими свойствами: 1) общая_длина_поля_данных = (СМЕЩЕНИЕ+ДЛИНА)/16; 2) (СМЕЩЕНИЕ + ДЛИНА) MOD 16 = 0; 3) нумерация битов в строке бит начинается с 0; 4) битовая строка (сечение) выравнивается по правому краю на границу байта.

Заметим, что при определении базовых элементов хранения имеет место ориентация на конкретную машинную архитектуру и с целью эффективной реализации арифметических операций выполняется выравнивание сечения.

Массив bit_vector представляет из себя массив структур define_bit_vector (dbv). В случае большой разрядности bit_vector (более 32) дополнительно выделяются переменные под информационную часть bit_vector с обеспечением соответствующих указателей в dbv. Номер элемента массива вычисляется исходя из соглашений Си, т.е. первый элемент массива нумеруется нулем.

Правила работы с битовыми векторами

Ниже иллюстрируются следующие виды работ с битовыми векто-

рами: а) выделение памяти для информационной части битовой переменной

```
unsigned int _имя_переменной [ AL ];
```

б) инициализация структуры ОПИСАНИЕ_БИТ_ВЕКТОРА

```
struct define_bit_vector имя_переменной =
                                { <тип>, 0, null, Ad, Al };
имя_переменной.dbv_b.p = _имя_переменной;
```

в) определение указателя на структуру ОПИСАНИЕ_БИТ_ВЕКТОРА

```
struct define_bit_vector *имя_переменной_;
```

г) инициализация указателя на структуру ОПИСАНИЕ_БИТ_ВЕКТОРА

```
имя_переменной_ = & имя_переменной .
```

Пример работы с битовыми векторами на физическом уровне приведен ниже: а) определение переменной типа bit_vector

```
int имя_переменной [количество_базовых_элементов];
```

б) инициализация структуры define_bit_vector

```
struct dbv _имя_переменной =
    { dbv_tbv, dbv_sok, null, 0, 0 };
_имя_переменной.dbv_b = имя_переменной;
_имя_переменной.dbv_d = смещение;
_имя_переменной.dbv_l = количество_разрядов;
```

При размещении переменной bit_vector в памяти нумерация битов выполняется слева направо, начинается с нуля, при этом происходит выравнивание по правому краю переменной. Пример определения битовой строки для битового вектора 111 01011101 01111010 10000011 длиной 27 бит приведен на рис. 3.

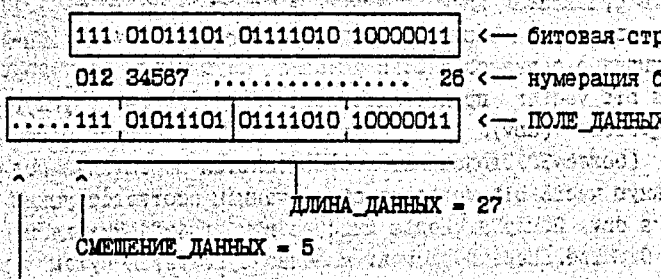


Рис. 3. Определение и размещение в памяти бит-вектора

Обработка битовых векторов в моделирующей программе

При обработке данных типа bit_vector в качестве параметров, передаваемых функции, используются адреса структур define_bit_vector. Функция возвращает адрес структуры define_bit_vector или данные типа boolean. Для размещения данных поля ответа используется рабочий пул. Выделение памяти под рабочий пул выполняется посредством вызова процедуры

BIT_INIT_WORK_BUFFER (РАЗМЕР_РАБОЧЕГО_ПУЛА).

По завершению выполнения моделирования оператора или по завершению всего процесса моделирования выполняется сброс рабочего пула посредством вызова процедуры

BIT_RESET_WORK_BUFFER ( ).

Сведения о рабочем пуле содержатся в структуре ОПИСАНИЕ_РАБОЧЕГО_ПУЛА. Доступ к ОПИСАНИЮ_РАБОЧЕГО_ПУЛА из моделирующей программы осуществляется посредством связывания на этапе редактирования (т.е. переменные описаны как EXTERNAL).

Функция, возвращающая bit_vector, выделяет в рабочем пуле место под структуру define_bit_vector и при необходимости место под переменную bit_vector и адрес define_bit_vector. Пример вызова функции

BIT_NOT (АДРЕС_dbv);
BIT_ADD (АДРЕС_dbv1, АДРЕС_dbv2).

Для функций, возвращающих boolean, место под возвращаемую переменную в рабочем пуле не выделяется. Пример вызова функции

BIT_EQU (АДРЕС_dbv1, АДРЕС_dbv2).

Отдельно выделены подпрограммы преобразования и присвоения полученных значений битовой переменной или ее сечения. ПОЛЕ_ДАННЫХ изменяется только после выполнения операции присвоения. Поле назначения bit_vector задается с точностью до позиции как и при задании операции вырезки (первая_позиция, последняя_позиция). Если указанные значения равны -1, то функция обрабатывает весь битовый вектор. Например, для операции присвоения X = A

BIT_ASSIGN (X, -1, -1, A).

При реализации формул на этапе создания текста моделирующей программы используются библиотечные вызовы пакета битовой арифметики. При этом формулы, описывающие поведение системы,

приводятся к польской записи, что задает последовательность вызовов соответствующих функций. Затем выражение преобразуется в некоторый текст на языке высокого уровня. Например, формула

```
X (1 to 3) = (A (1 to 3) + B) / C (2)
```

преобразуется к виду

```
BIT_ASSIGN (
            X,1,3,
            BIT_DVD (
                     BIT_ADD (
                              BIT_SLICE (A,1,3),
                              B
                              ),
                     BIT_SLICE (C,2,2)
                     )
           ).
```

При обработке битовых векторов имеют место следующие особенности: а) bit_vector представим ТОЛЬКО В ПРЯМОМ КОДЕ. В силу этого операции сравнения определяются флагом ЗНАК_ДАННОГО структуры define_bit_vector и арифметические операции строятся как последовательность операции сравнения и операции сложения или вычитания из большего меньшего; б) обработка bit_vector производится ТОЛЬКО ПОРЦИЯМИ БАЗОВЫХ ЭЛЕМЕНТОВ; в) при согласовании bit_vector с данными типа bit и boolean используется совместимость по внутреннему представлению данных; г) при представлении индексов в моделирующей программе в качестве значений используются значения из описания на VHDL без преобразований; д) при реализации операции вырезки в качестве передаваемых параметров используется адрес define_bit_vector, номер_первой_позиции, номер_последней_позиции; е) использование в описании указания направления нумерации бит приводит к формированию соответствующего флага dbv. Обработка такого bit_vector происходит справа налево ТОЛЬКО при выполнении операции инициализации bit_vector (т.е. подача элементов при заполнении происходит как a[n], a[n-1], a[n-2], ...).

Пользовательская среда при работе с библиотекой битовых векторов

Программное окружение определяется использованием языка Си, а именно: подпрограммы находятся в библиотеке LIB_BIT.LIB;

прототипы функций находятся в LIB_BIT.H. При реализации сопряжения предполагается следующее: а) используется "классический" интерфейс Си (без объектно-ориентированных структур C++ или каких-либо других нестандартных возможностей); б) для обеспечения системных функций, определяемых аппаратурой, используются собственные вызовы поддержки интерфейса; с) включение подпрограмм выполняется автоматически при задании соответствующей библиотеки в файле ПРОЕКТ_ЗАДАНИЯ.

Одноопернадная библиотечная функция с полем ответа в рабочей области имеет прототип

```
АДРЕС_ОПИСАНИЯ_СТРОКИ_БИТ_ОТВЕТ
        ВЫЗОВ
            (АДРЕС_ОПИСАНИЯ_СТРОКИ_БИТ_ОПЕРАНД)
```

Многооперандная библиотечная функция с полем ответа в рабочей области имеет прототип

```
АДРЕС_ОПИСАНИЯ_СТРОКИ_БИТ_ОТВЕТ
        ВЫЗОВ
            (АДРЕС_ОПИСАНИЯ_СТРОКИ_БИТ_ОПЕРАНД_1,
             ...,
             АДРЕС_ОПИСАНИЯ_СТРОКИ_БИТ_ОПЕРАНД_N)
```

При возврате функцией данных типа boolean рабочий пул не используется и прототип библиотечной функции имеет вид

```
ДАННОЕ_ТИПА_ЛОГИЧЕСКИЙ
        ВЫЗОВ
            (АДРЕС_ОПИСАНИЯ_СТРОКИ_БИТ_ОПЕРАНД_1,
             ...,
             АДРЕС_ОПИСАНИЯ_СТРОКИ_БИТ_ОПЕРАНД_N)
```

Ниже приведен перечень библиотечных операций над битовыми векторами с описанием их прототипов: 1) бинарные логические операции BIT_AND, BIT_OR, BIT_NAND, BIT_NOR, BIT_XOR. Прототип - bit_vector * ФУНКЦИЯ (bit_vector * X1, bit_vector * X2); 2) операция конкатенации - bit_vector * BIT_CON (bit_vector * X1, bit_vector * X2); 3) бинарные арифметические операции BIT_ADD, BIT_SUB, BIT_MLT, BIT_DVD. Прототип - bit_vector * ФУНКЦИЯ (bit_vector * X1, bit_vector * X2); 4) бинарные операции отношения BIT_EQ, BIT_NEQ, BIT_LT, BIT_LE, BIT_GT, BIT_GE. Прототип - char ФУНКЦИЯ (bit_vector * X1, bit_vector * X2); 5) бинарные операции сдвига BIT_SHL, BIT_SHR. Прототип

- bit_vector * ФУНКЦИЯ (bit_vector * X1, int N); 6) бинарная операция присваивания - BIT_ASSIGN (bit_vector * OUT, unsigned int X1, unsigned int X2, bit_vector * IN); 7) унарная операция НЕ - bit_vector * BIT_NOT (bit_vector * X); 8) унарная операция вырезки - bit_vector * BIT_SLICE (bit_vector * X, unsigned int X1, unsigned int X2).

Библиотеки подключаются с помощью указания их в проекте с последующим редактированием связей на этапе внешнего редактирования.

Пакет может быть использован как автономно, так и в рамках САПР СБИС.

2.6. Подсистема моделирования

Объектом моделирования является проект СБИС, описанный заказчиком на поведенческом уровне средствами языка VHDL. Моделирование выполняется на тестовых наборах - описаниях входных воздействий, имитирующих реальное окружение моделируемой СБИС, и выходных эталонных реакций проекта СБИС на входные воздействия, подготовленных заказчиком.

На поведенческом уровне проект задается входными-выходными сигналами и алгоритмами преобразования входных сигналов в выходные. Описание выполняется с использованием модулей entity и architecture и механизма процессов VHDL, что позволяет специфицировать функции, реализуемые проектом, и характер их взаимодействия. При этом закон функционирования проекта, предусматривающий реализацию ряда подфункций, отображается адекватной совокупностью процессов.

В качестве первого приближения описания проекта без ориентации на конкретные аппаратные компоненты используется модуль architecture в виде блока с одним процессом. Сам процесс описывается множеством последовательных операторов, последним из которых является оператор назначения тех сигналов, которые заданы в entity в режиме out, inout.

Тесты задаются в виде набора тестовых векторов путем перечисления статических либо динамических последовательностей значений сигналов на входах и выходах проекта, различающихся способом их интерпретации при моделировании, и необходимой

точностью воспроизведения проектом выходных реакций. В первом случае соседние значения каждого сигнала не связаны друг с другом и обрабатываются независимо. Каждый тестовый вектор используется для организации одного полного цикла моделирования. Во втором случае значения каждого сигнала представляют собой временные диаграммы и анализ входных и выходных значений сигналов осуществляется на каждом такте моделирования проекта с учетом предыдущих значений сигналов. Входные и выходные значения задаются значениями соответствующих типов, определяемых VHDL.

Система обеспечивает имитацию функционирования проекта СБИС на языке VHDL, управляемую трассой входных сигналов, соответствующей временной диаграмме работы проектируемого СБИС. Она функционирует в режимах:

1) отладки на этапе формирования текста описания закона функционирования проекта на VHDL с целью его синтаксической и семантической коррекции;

2) собственно моделирования с целью верификации проекта - входного контроля на заданных пользователем тестовых наборах путем сличения эталонных и реальных результатов;

3) оценки характеристик проекта (точностных свойств спецификации) и степени их соответствия критериям оптимизации;

4) генерации тестовых наборов для последующего моделирования проекта на более детальных уровнях описания.

По завершении моделирования автоматически формируется файл результатов. В статическом режиме для каждого тестового вектора отображаются соответствующие пары эталонных и модельных значений с указанием величины их расхождения. В динамическом режиме фиксируются расхождения эталонных и модельных временных диаграмм выходных сигналов. В отладочном режиме в выходной файл помимо результатов выполнения выводится потактная трасса моделирования.

Само моделирование выполняется в два этапа: на первом по результатам лексического и синтаксического разбора описания проекта создается текст программного компонента, адекватного исходному описанию; на втором этапе, выполняемом многократно, производится имитационное, событийное моделирование описания проекта, загрузочный код которого получают транслятором СИ. Соответственно этому проектная процедура моделирования стро-

ится как совокупность подсистем ТРАНСЛЯЦИИ, КОНВЕРТИРОВАНИЯ, ТЕСТОВЫЕ НАБОРЫ, ГЕНЕРАЦИИ СИ-ТЕКСТОВ и МОДЕЛИРОВАНИЯ (рис. 4).

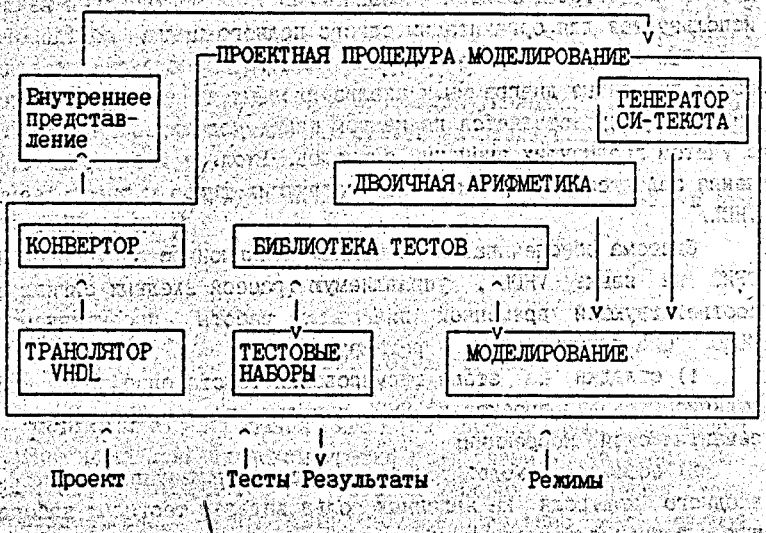


Рис. 4. Структура проектной процедуры МОДЕЛИРОВАНИЕ

Подсистемы ТРАНСЛЯТОР и КОНВЕРТОР обеспечивают: лексический, семантический и синтаксический анализ проекта и формирование его внутреннего представления в виде информационных таблиц - структур данных СИ.

Подсистема ТЕСТОВЫЕ НАБОРЫ обеспечивает настройку модулей процедуры на интерфейс моделируемого проекта, подготовку входных воздействий для проектируемой СБИС, в ходе моделирования обеспечивает накопление и анализ реакций проекта с выдачей результатов. Выполняется автономно.

Подсистема ГЕНЕРАЦИИ обеспечивает получение адекватного проекту текста на языке СИ, готового для последующего исполнения. В процессе генерации к СИ-тексту подключается пакет ДВОИЧНАЯ АРИФМЕТИКА и программа МОНИТОР , используемая для управления моделированием на уровне описанных в блоке процессов путем их запуска и синхронизации.

Подсистема ДВОИЧНАЯ АРИФМЕТИКА обеспечивает эффективную поддержку операций над объектами типа bit-vector, составляющи-

ми вместе с типами bit, boolean основу предопределенной арифметики VHDL.

Подсистема МОДЕЛИРОВАНИЕ осуществляет "прокрутку" СИ-текста проекта. Предварительно подсистема настраивается на интерфейс моделируемого проекта для обеспечения просмотра тестового набора, выделения тестовых векторов и передачи соответствующих значений в моделируемый блок. Затем транслятором СИ создается единый загрузочный модуль, представляющий сборку подсистемы МОДЕЛИРОВАНИЕ и СИ-текста проекта. При его запуске инициируется МОНИТОР и осуществляется обработка информации рядом модулей, реализующих элементарные функциональные преобразования.

Схема функционирования подсистемы приведена на рис. 5 и предусматривает два основных этапа. На первом этапе, проводимом однократно, генерируется СИ-текст, который передается на моделирование. На втором этапе, выполняемом многократно, производится имитационное моделирование. Этапы выполняются друг за другом либо порознь, что обеспечивает иерархический подход к моделированию и нисходящий-восходящий подходы к проектированию.

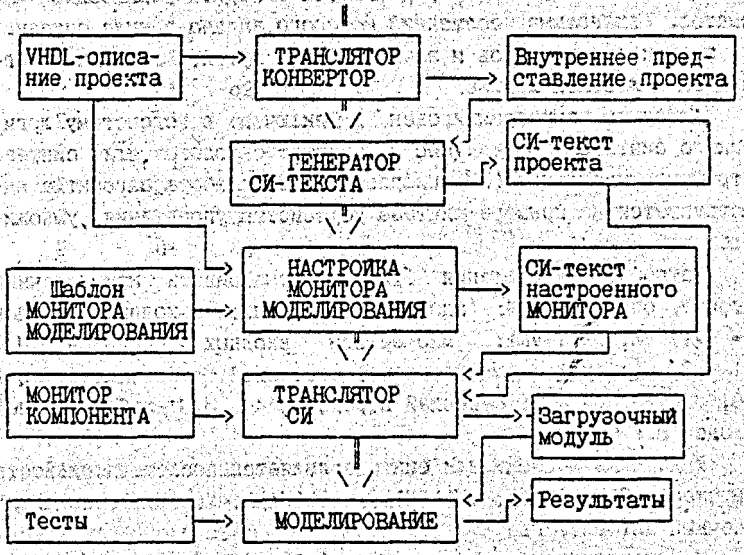


Рис. 5. Порядок функционирования процедуры

3. СИНТЕЗ УСТРОЙСТВ УПРАВЛЕНИЯ СБИС

3.1. Исходные данные и результат синтеза

При автоматизированном проектировании цифровых дискретных устройств широкое применение находят автоматные модели описания функционирования синтезируемых схем (автоматы Мили, Мура и их обобщения), а также алгоритмические модели (граф-схемы алгоритмов) [27-31]. При этом форма представления играет существенную роль и определяет, зачастую существенно, эффективность реализации и целесообразность использования проектировщиками.

Процесс разработки алгоритма управления является итерационным с использованием процедур моделирования. На первых шагах при разработке проектировщик пытается просматривать различные последовательности формирования подмножеств управляющих сигналов (смоделировать процесс управления блоками выбранной архитектуры). Для этого удобной формой представления является описание в виде последовательности циклов управления и линейных участков, их связывающих.

В настоящем разделе предлагается алгоритм реализации автоматов, диаграмма состояний которого задана в виде описания его элементарных циклов и элементарных путей, на микропрограммной и жесткой логике.

Алгоритм запрограммирован и включен в подсистему логического синтеза с языка VHDL в качестве базового для синтеза устройств управления (УУ) цифровых СБИС. Работа алгоритма иллюстрируется на примере синтеза устройства управления умножителем.

Устройство управления будем рассматривать как конечный автомат, определяемый: множеством двоичных выходных сигналов V = {v1, v2, ..., vm}; множеством входных сигналов U = {u1, u2, ..., un}; множеством состояний автомата Q = {q0, q1, ..., qp}; функцией переходов A : QxU -> Q; функцией выходов B : QxU -> V.

Исходными данными для синтеза является последовательность S множеств S(N)={t(N), U(t(N)), q(t(N))}, V(t(N))}, задающая конечный автомат, где

N - номер кванта времени, N = 1,2,...;

t(N) - время пребывания УУ в текущем состоянии;

U(N) - множество входных сигналов, поступающих на УУ к моменту t(N);

q(N) - состояние УУ, в которое он переключается в момент времени t(N);

V(N) - множество выходных сигналов, вырабатываемое УУ начиная с момента времени t(N).

Например, диаграмма автомата, показанная на рис. 6,

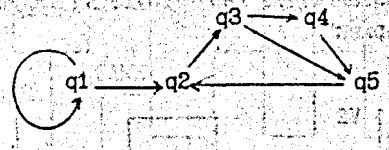


Рис. 6. Диаграмма автомата

представляется в виде массива, который удобно представить в виде таблицы

N	t(N)	U(N)	Q(N)	V(N)
1	3	^u1	q1	-
2	2	u1	q2	v1
3	2	u1	q3	v2
4	2	u2	q5	v1
5	2	-	q2	-
6	2	u1	q3	v1
7	2	u1	q4	-
8	2	-	q5	v2
9	2	-	q2	-

Строка 1 описывает цикл (1, 1); строка 2 - линейный участок (1, 2), строки 3-5 - цикл (2, 3, 5, 2), строки 6-9 - цикл (2, 3, 4, 5, 2).

Результатом синтеза является УУ на жесткой или гибкой логике, выполняющее заданный алгоритм за время ТО и обладающее минимальной сложностью.

В качестве базовой структуры на жесткой логике примем схему, состоящую из памяти и двух комбинационных схем (рис. 7).

Одна схема предназначена для формирования выходных сигналов V1 типа 1 (выходные сигналы автомата Мили) и входных сигналов памяти. Другая схема предназначена для формирования выходных сигналов V2 типа 2 (выходные сигналы автомата Мура).

Поставив во главу угла надежность работы устройства, для борьбы с гонками будем использовать дублирование памяти [32]. Недостатком этого способа является увеличение объема оборудования и временного цикла автомата. Однако другие способы (рациональный выбор длительности синхроимпульса; развязывание пар переходов; соседнее кодирование) не гарантируют надежную работу в случае "нулевых задержек".

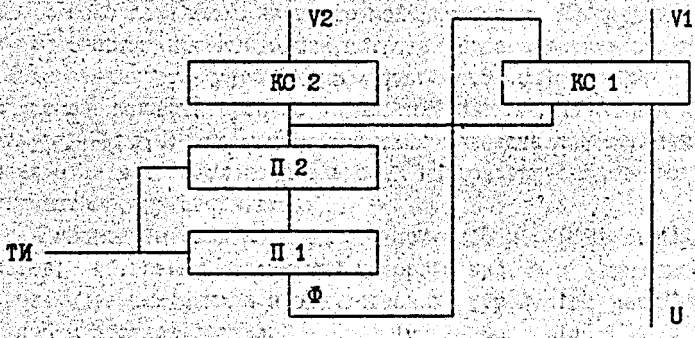


Рис. 7. Типовая схема УУ на жесткой логике

Эта структура представляет собой С-автомат [31], т.е. обобщение автоматов Мили и Мура.

Основную память будем реализовывать на JK-триггерах, так как они дают больше возможностей при логическом синтезе, чем, например, D-триггеры, потому как допускают при переключениях безразличное значение сигнала на одном из входов. Дополнительная память предназначена только для переписывания состояния основной памяти, поэтому ее целесообразно реализовать на D-триггерах, так как они занимают меньше места на кристалле.

При построении схемы с микропрограммной логикой будем использовать статическое микропрограммирование (использование ПЗУ в качестве памяти микропрограмм), горизонтально-вертикальную структуру микрокоманды (как наиболее гибкую и общую) и

прямое кодирование (косвенное кодирование позволяет несколько сократить операционную часть микрокоманды, но существенно усложняет алгоритм синтеза).

Типовая схема УУ с микропрограммной логикой изображена на рис. 8.

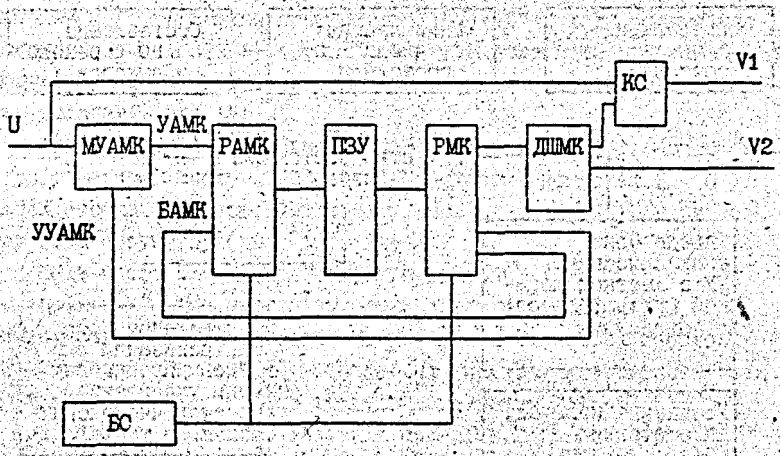


Рис. 8. Типовая схема УУ на гибкой логике

Микропрограммы хранятся в ПЗУ. Выходные данные ПЗУ принимаются в регистр микрокоманды (РМК). Операционная часть содержимого РМК поступает на дешифратор микрокоманды (ДШМК), формирующий выходные сигналы второго типа (V2) и входные сигналы для комбинационной схемы КС, предназначенную для формирования выходных сигналов первого типа V1, на которую поступают также входные сигналы U.

Адресная часть микрокоманды содержит безусловную часть адреса следующей микрокоманды (БАМК), поступающую непосредственно на регистр адреса микрокоманды (РАМК) и код управления условной частью адреса следующей микрокоманды (УУАМК), поступающей на входы управления мультиплексора условной части адреса следующей микрокоманды (МУАМК), на информационные входы которого поступают входные сигналы U.

Разнесенные по фазе синхросигналы на входы РАМК и РМК поступают из блока синхронизации (БС).

3.2. Основные этапы синтеза

Алгоритм синтеза показан на рис. 9.

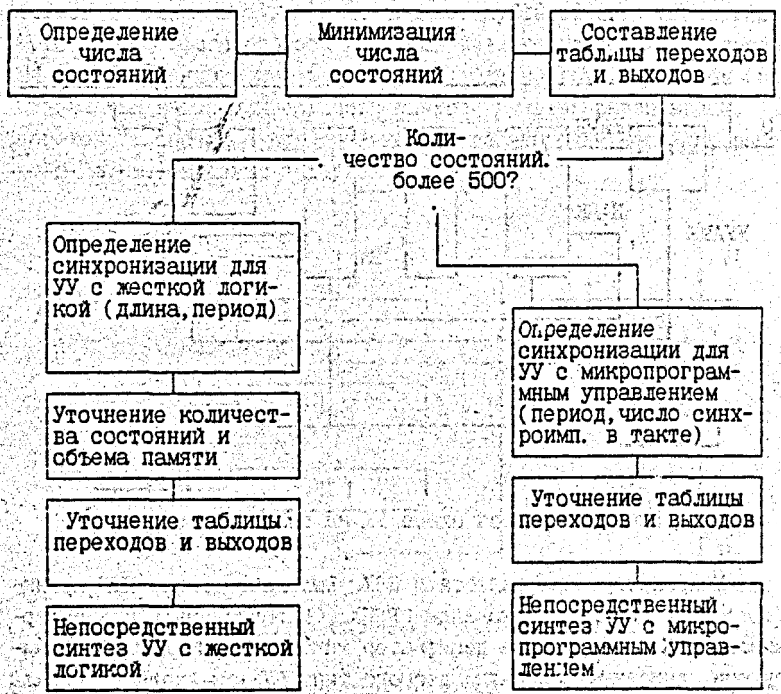


Рис. 9. Укрупненная блок схема алгоритма синтеза УУ

При определении количества состояний устройство управления рассматривается как автомат Мура и, следовательно, все выходные сигналы V включаются в множество V2. Анализируется массив S на предмет совпадения в любой момент времени t множества Q(t) U V(t). Одинаковым множествам поставим во взаимно однозначное соответствие состояния автомата. Таким образом формируется множество состояний Q'(t).

Минимизация количества состояний заключается в следующем. Определяется множество V1 выходных сигналов, зависящих от

входных сигналов. Исключаем из множества V2 элементы множества V1. После этого повторяем процедуру определения количества состояний, но анализируем множества Q'(t) U V2(t). Полученное множество состояний Q"(t) является искомым. В конце этапа делается проверка эффективности минимизации количества состояний, т.е. оценивается соотношение |Q'|/|Q"|. Если количество состояний уменьшилось менее, чем в 1,3 раза, то минимизацию следует признать неэффективной и отказаться от нее, т.е. следует полагать, что |V1|=0.

Таблица переходов и выходов F автомата имеет число L строк, равное числу состояний (мощности множества дан Q"). Каждая строка соответствует одному состоянию Q"(R) и имеет следующую структуру (рис. 10), где

поле q содержит номер текущего состояния;

поле V1/U - множество выходных сигналов 1-го типа и множество соответствующих подмножеств входных сигналов, необходимых для выработки каждого из выходных (каждому из сигналов vi из V1 соответствует одно и то ж подмножество Ui);

поле V2 - множество выходных сигналов 2-го типа ;

поля q1*/U, q2*/U, ..., qL*/U содержат номера следующих состояний и соответствующие множества входных сигналов, в зависимости от которых происходит переход из состояния q в следующее состояние q1*, q2*, ..., qL*.

q	V1/U	V2	q1*/U	q2*/U	...	qL*/U

Рис. 10. Вид таблицы переходов и выходов

Число столбцов таблицы F определяется строкой с наибольшим числом полей (она соответствует состоянию с наибольшим числом переходов в следующее состояние).

Определение типа УУ производится на основании анализа количества состояний: если |Q"|>500, то переходить к синтезу УУ на гибкой логике (т.е. с микропрограммным управлением), в противном случае синтезировать автомат на жесткой логике. Однако, если синтезированная структура на жесткой логике не будет

удовлетворять исходным требованиям, то безусловно выполняется синтез микропрограммируемой структуры.

Далее следуют этапы, содержание которых зависит от выбранного типа устройства:

- определение типа синхронизации;
- уточнение количества состояний;
- уточнение таблицы переходов и выходов.

В результате их выполнения получается окончательная таблица переходов и выходов, которая содержит исходные данные для непосредственного синтеза УУ.

На этапе расчета синхронизации вычисляется длительность рабочего такта T_T. Далее для выбранной структуры микропрограммного УУ формируется последовательность разнесенных по фазе синхросигналов, число К которых выбирается в пределах 4-8. Для этой цели на выходе тактового генератора ставится сдвиговый регистр. Здесь же вычисляется и разрядность счетчика циклов (находится в пределах 0-4). Частота тактового генератора, количество синхроимпульсов в такте и длительность тактовых импульсов в рабочем такте вычисляются на основании данных о характеристиках элементов схемы (задержка срабатывания РАМК и ПЗУ, время разброса синхросигналов, время установки РМК, задержка РМК по данным и по синхронизации, задержка МУАМК по управлению).

Уточнение таблицы F заключается во введении в матрицу столбца, в котором будет обозначено время пребывания автомата в данном состоянии.

Определение типа синхронизации для УУ на жесткой логике заключается в получении следующих параметров:

- длительности рабочего такта T_T;
- длительности синхроимпульса t;
- частоты тактового генератора f=1/T_T.

УУ с жесткой логикой обладает меньшими возможностями асинхронной работы по сравнению с микропрограммным УУ, так как фактически работа происходит по одному синхросигналу. Поэтому для получения какой-либо асинхронности предлагается ввести дополнительные состояния УУ. И если нам необходимо пребывать в некотором состоянии I тактов, вводим дополнительные log I триггеров в памяти П1 и П2, которые образуют как бы счетчик циклов. При поступлении ТИ изменяется только содержимое допол-

нительных триггеров, собственно код состояния не меняется. Это происходит до того момента, когда код, содержащийся в дополнительных триггерах не достигнет определенного значения, допускающего изменения состояния.

Длительность синхроимпульса, частота тактового генератора, количество синхроимпульсов в такте и длительность тактовых импульсов в рабочем такте вычисляются на основании данных о времени срабатывания и установки D- и JK-триггеров и времени срабатывания базового вентиля.

При выборе количества дополнительных триггеров I равно максимальному количеству рабочих тактов между истинным переключением состояний (I выбирается в пределах 1-8).

Уточнение количества состояний заключается в разбиении каждого состояния на несколько состояний в зависимости от того, сколько ТИ требуется для перехода в следующее состояние.

В результате выполнения этапа получаем: уточненное количество состояний, количество триггеров в каждой из памяти П1 и П2, равное сумме количества основных и дополнительных триггеров.

Уточнение таблицы F заключается в введении в нее новых строк, соответствующих введенным в Q" дополнительных состояний. Кроме того, в таблицу F вводится новый столбец, в который помещается параметр МТ, обозначающий время пребывания в каждом состоянии. Этот параметр по сути обозначает код, заносимый в дополнительные триггера при истинной смене состояний. Далее с каждым последующим синхроимпульсом происходит модификация содержимого дополнительных триггеров на (-1), до достижения ими нуля. Нулевое содержимое дополнительных триггеров является разрешением смены кода состояния в основных триггерах.

3.3. Алгоритм синтеза структуры на жесткой логике

Непосредственный синтез структуры жесткой логики содержит три подэтапа:

а) кодирование состояний;

б) формирование функций возбуждения памяти;

в) выбор длительности и периода следования тактовых импульсов (ТИ);

д) определение логических выражений для V1 и V2.

Исходными данными для кодирования состояний являются:

- массив S;
- минимизированный массив состояний Q"(R);
- таблица переходов и выходов F.

Результатом выполнения подэтапа является введение в таблицу F нового столбца, в который помещается код состояния. С целью некоторой минимизации количества переключений кодирование состояний осуществляется с помощью кода Грея.

Исходными данными для формирования функций возбуждения является таблица F и количество триггеров в памяти, а результатом выполнения - пара массивов FJ(I) и FK(I), элементами которых являются представленные в дизъюнктивной форме функции возбуждения I-го триггера для J- и K-входов соответственно.

Исходными данными для подэтапа формирования таблицы истинности выходных сигналов являются:

- таблица F;
- массив выходных сигналов V(M);
- массив V1(L) выходных сигналов первого типа;
- массив конъюнкций входных сигналов U1(V1(L));
- количество триггеров KI и количество дополнительных триггеров KD.

Результатом выполнения являются массивы FV1(L) и FV2(M), содержащие выражения в дизъюнктивной форме для выходов V1(L) и V2(M) соответственно 1-го и 2-го типа.

Уточненная структурная схема УУ приведена на рис. 11.

3.4. Алгоритм синтеза структуры на гибкой логике

Этап непосредственного синтеза УУ с микропрограммным управлением содержит подэтапы:

а) формирование массива состояний, связанных по переходам;

б) формирование таблицы адресов микрокоманд;

в) формирование таблицы истинности МУАМК;

г) формирование структуры микрокоманды и формирование таблицы истинности выходных сигналов.

Под связанными состояниями понимаются состояния УУ, свя-

занные по переходам. Это значит, что адреса микрокоманд, соответствующих этим адресам, должны различаться только в части УАМК. Связанные состояния представляются массивом Y(D), в D-й строке которого содержится E элементов - номеров состояний.

Определение структуры микрокоманды включает определение количества и размеров полей и способ кодирования.

Таблица адресов микрокоманд в процессе выполнения подэтапа формируется не как отдельная таблица, а как дополнение к таблице F. В нее вводятся два новых столбца. В один столбец помещается адрес микрокоманды, соответствующей данному состоянию Q"(R), а в другой - параметр, равный количеству разрядов адреса микрокоманды, которые для данного состояния зависят от входных сигналов U. Он определяется, как]log Z(D)[, где Z(D)=|Y(D)| - длина D-й строки массива связанных состояний.

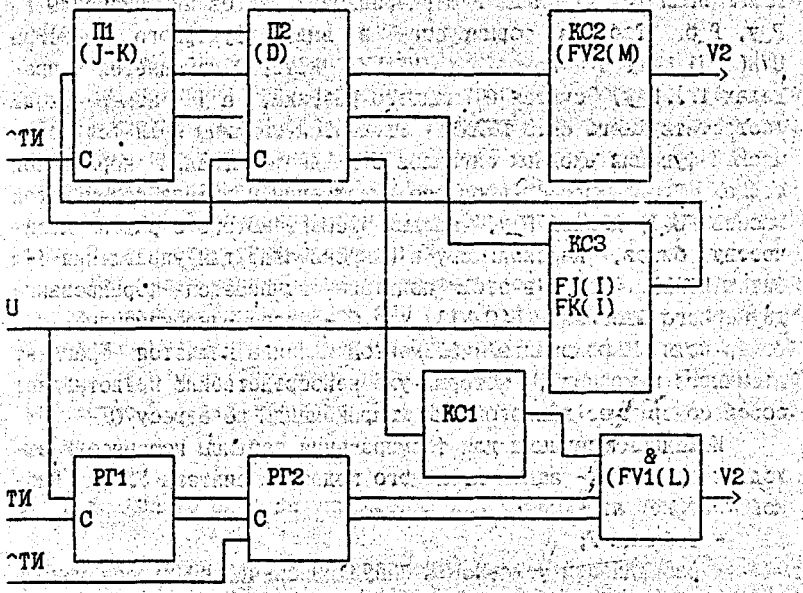


Рис. 11. Уточненная схема УУ на жесткой логике

Предварительно определяется разрядность адреса микрокоманды P, разрядность УАМК P_у и БАМК P_б.

Полный адрес микрокоманды АМК содержит P =]log L[разрядов, где L=|Q"| - число микрокоманд (объем ПЗУ). Условная часть адреса микрокоманды содержит P_у=]logEM [разрядов, где EM - максимальное количество состояний, связанных по переходам (максимальная длина строки массива связанных состояний). Безусловная часть адреса микрокоманды содержит P_б=P-P_у разрядов.

Результатом выполнения подэтапа является дополненная таблица переходов и выходов F.

Заметим, что в процессе выполнения подэтапа может потребоваться увеличение разрядности БАМК P_б и, соответственно, общей разрядности адреса микрокоманды P.

Исходными данными для формирования таблицы истинности МУАМК является таблица F и разрядности адреса микрокоманды P, P_у, P_б. Таблица формируется в виде двумерного массива UVH(I, J), где I - номер бита МУАМК, который изменяется в пределах 1...P_у, считая с младшего разряда, а J - номер входа соответствующего бита МУАМК. Элементом таблицы является логическая функция входных сигналов U в дизъюнктивной нормальной форме. Кроме того, в процессе выполнения подэтапа формируется массив FU(I) длиной P_у, каждый элемент которого равен количеству битов, необходимому в микрокоманде для управления I-м битом МУАМК. Также в этом подэтапе начинается формирование двумерного массива PZU(C,N), где C - адрес микрокоманды, N - номер поля микрокоманды. Элементом массива является фрагмент двоичного микрокода, который уже непосредственно представляет собой содержимое данного поля микрокоманды по адресу C.

Исходными данными для формирования таблицы истинности выходных сигналов - заключительного подэтапа синтеза УУ на гибкой логике являются:

- таблица F;
- разрядности условной и безусловной частей адреса микрокоманды P_у и P_б;
- множества выходных сигналов V и V1;
- массив конъюнкций входных сигналов U1(V1(L)).

Результатом выполнения подэтапа являются:

- окончательно сформированный массив PZU(C,N), где C -

адрес микрокоманды, N - номер поля микрокоманды;

- массив FU(N), где каждый элемент представляет собой количество разрядов в N-м поле микрокоманды;

- параметр NU, равный общему количеству информационных разрядов микрокоманды;

- массив VYH1(L), описывающий выходы первого типа. Каждый его элемент имеет вид V1(L)*N*J*U1(V1(L)), где V1(L) - выходной сигнал, N - номер поля микрокоманды, управляющего выдачей сигнала, J - номер выхода дешифратора, U1(V1(L)) - конъюнкция входных сигналов, срответствующая данному выходному сигналу. Объединение этих входных сигналов и J-го выхода дешифратора дают исходный сигнал V1(L);

- массив VYH2(M), описывающий выходы второго типа. Каждый его элемент имеет вид V2(L)*N*J, где V2(M) - выходной сигнал, N - номер поля микрокоманды, управляющего выдачей сигнала, J - номер выхода дешифратора, на котором возникает данный выходной сигнал;

е) CN - адрес микрокоманды начального состояния.

Для получения возможности асинхронной работы (т.е. для того, чтобы смену состояний УУ производить через различные временные интервалы, кратные некоторой длительности такта) целесообразно ввести схему блокировки рабочих импульсов и счетчик циклов. Уточненная схема микропрограммного УУ дана на рис. 12, при этом учитывается синхронизация.

Для более точного выбора длительности такта, определения разрядности счетчика циклов СчЦ (рис. 12) необходимо проанализировать множество временных интервалов между состояниями Q"(t). Его можно получить из исходного массива S. Полученное множество T будет содержать N-1 элементов t_i = t(i+1) - t(i).

3.5. Пример синтеза управления для устройства умножения

Рассмотрим алгоритм управления умножителя (рис. 13): разрядность 1-го и 2-го операнда - N, разрядность регистра Рг1 - 2N, Рг2 - N, Рг3 - 2N, сумматора - 2N.

Запуск умножителя осуществляется по сигналу U1. По сигналам V1 и V3 осуществляется прием операндов в Рг1 и Рг2 (в Рг1 операнд принимается в младшие N разрядов, в старшие N разрядов

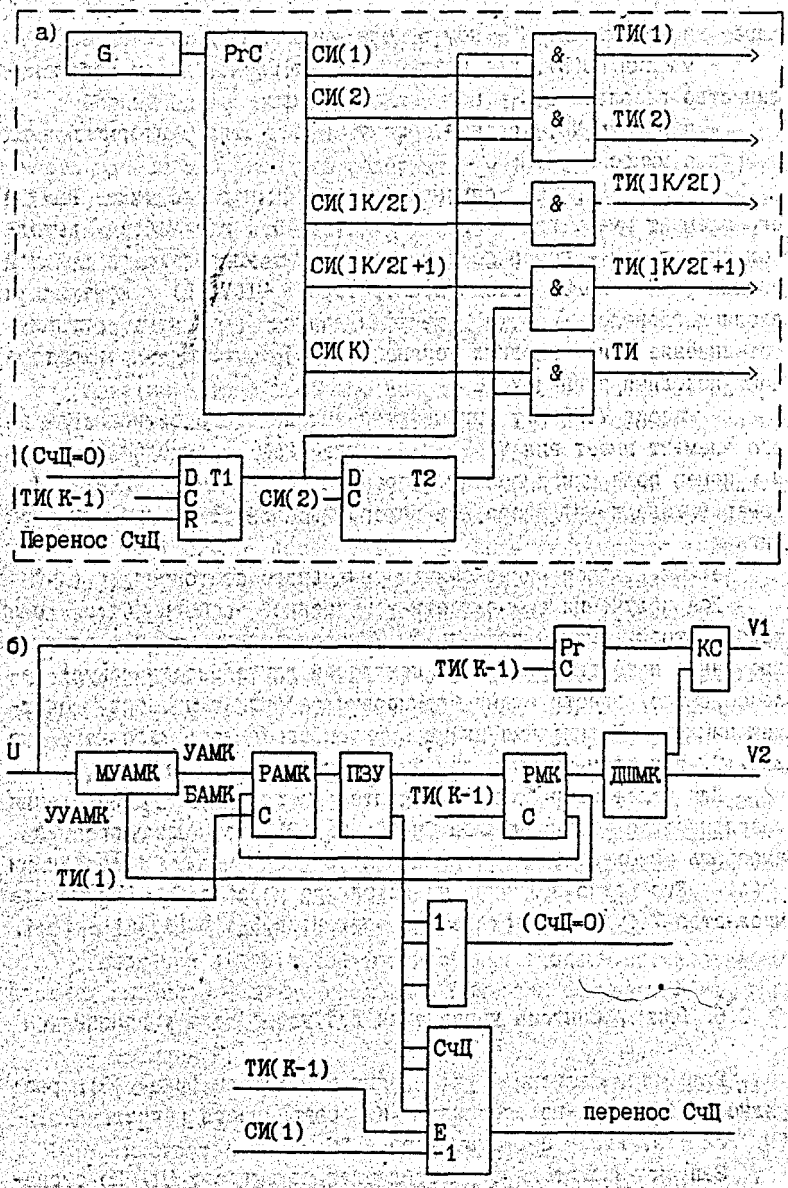


Рис. 12. Уточненная схема УУ на гибкой логике

принимаются нули). Сигналом V6 обнуляется регистр результата Рг3. Далее осуществляется сдвиг вправо второго операнда и, если текущий младший разряд равен 1, происходит прием текущего произведения в Рг3, если - 0, то приема нет. По сигналу U3, когда Рг2=0, операция завершается.

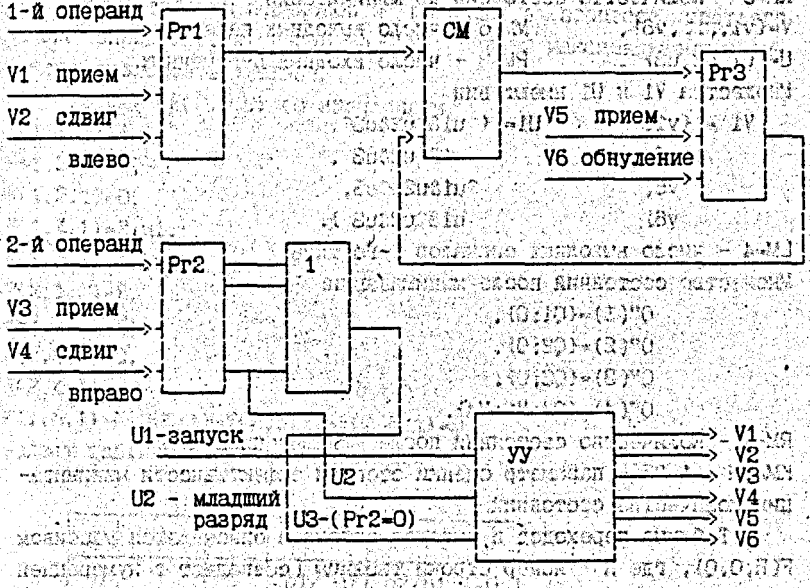


Рис. 13. Фрагмент схемы устройства умножения

Алгоритм УУ представляется следующим массивом S:

```
N=1   S: t=0   u1 u2 u3 Q1  0
N=2   S: t=5   u1 u2 u3 Q2  v1 v3 v6
N=3   S: t=15  u1 u2 u3 Q3  v5
N=4   S: t=20  u1 u2 u3 Q4  v2 v4
N=5   S: t=30  u1 u2 u3 Q3  0
N=6   S: t=35  u1 u2 u3 Q4  v2 v4
N=7   S: t=45  u1 u2 u3 Q3  v5
N=8   S: t=50  u1 u2 u3 Q4  v2 v4
N=9   S: t=60  u1 u2 u3 Q1  0
NM=9  - число строк массива S
```

Множество Q' состоит из пяти состояний:

Q'(1)={Q1;O},
Q'(2)={Q2;V1;V3;V6},
Q'(3)={Q3;V5},
Q'(4)={Q4;V2;V4},
Q'(5)={Q3;O}.

KM=5 - количество состояний до минимизации.

V={v1,...,v6}, MM=6 - число выходных переменных.

U={u1,u2,u3}, PM=3 - число входных переменных.

Множества V1 и U1 имеют вид

V1 = {v1, U1= { u1&^u2&u3^,
v3, u1&^u2&u3 ,
v5, ^u1&u2&^u3,
v6} u1&^u2&u3 }.

LM=4 - число выходных сигналов 1-го типа.

Множество состояний после минимизации:

Q"(1)={Q1;O},
Q"(2)={Q2;O},
Q"(3)={Q3;O},
Q"(4)={Q4;V2;V4}.

RM=4 - количество состояний после минимизации.

KM/RM = 1.25 - параметр оценки степени эффективности минимизации количества состояний.

Таблица переходов и выходов автомата описывается массивом F(R,O,Q), где R - номер строки таблицы (совпадает с нумерацией Q"(R)), O - номер столбца, Q - номер подстолбца.

Первый столбец (O=1, Q=1) содержит только один подстолбец. В нем находится номер состояния R.

Второй столбец (O=2) содержит |V1| подстолбцов, каждый из которых содержит выходной сигнал первого типа V1 и соответствующую ему конъюнкцию входных сигналов из множества U1.

Третий столбец (O=3) содержит |V2| подстолбцов, в каждый из которых помещается выходной сигнал второго типа (т.е. выходной сигнал, не являющийся элементом массива V1), присутствующий в состоянии Q"(R).

Так как состояние Q"(R) определяется совокупностью состояний Q(N) и выходных сигналов V(N), не входящих в V1 (т.е. выходов 2-го типа), то набор выходных сигналов в столбце O=3 является единственным для данного состояния.

Столбцы O = 4, ...; OM содержат номера состояний, в которые УУ переходит из состояния Q"(R) и соответствующие им наборы входных сигналов U. В конце формирования таблицы наборы входных сигналов в этих столбцах минимизируются путем удаления входных сигналов, которые повторяются во всех столбцах и, следовательно, не влияют на переключение состояний.

Кроме таблицы переходов и выходов в процессе третьего этапа определяется параметр OMM, равный максимальному количеству столбцов в таблице.

Массив F(R,O,Q) со значениями

F(1,1,1)=1,
F(1,2,1)=0,
F(1,3,1)=0,
F(1,4,1)=2,u1,
F(1,5,1)=1,^u1,
F(2,1,1)=2,
F(2,2,1)=V1,
F(2,3,1)=0,
F(2,4,1)=3,
F(1,6,1)=F(2,6,1)=F(3,6,1)=F(4,6,1)=0,

задает таблицу переходов и выходов

q	V1/U	V2	q1*/U	q2*/U	MT
q1			q1/^u1	q2/u1	0
q2	v1		q3/u2		0
q3			q4/u1		0
q4			q1/u3	q3/^u3	0

OMM=6 - число столбцов таблицы. Последний нулевой столбец появляется в таблице после вычисления синхронизации (T_T=10).

Непосредственный синтез. Микропрограммная логика

Массив связанных состояний задает два подмножества связанных состояний: Y(1)={ q2,q1,q3 } - состояние q1 связано с тремя состояниями:

- q1 -> q1; q1 -> q2; q3 -> q1

Y(2)={ q4 }, - состояние q2 связано с одним состоянием:

- q2 -> q4

В результате выполнения подэтапа, кроме массива Y(D,E), формируется также одномерный массив Z(D), содержащий длину D-й строки и параметры: EM, равный максимальной длине строки таблицы Y(D,E), и DM, равный количеству строк массива Y(D,E) и длине массива Z(D):

Z(1)=3, Z(2)=1, EM=3, DM=2

Структура микрокоманды: P=2; P_y=2; P_б=0;

D(1,1)=1; D(1,2)=2;

Дополнительные столбцы таблицы F после определения структуры микрокоманды:

F(2,7,1)=0, F(1,7,1)=1, F(3,7,1)=2, F(4,7,1)=3 - адреса микрокоманд (АМК);

F(2,8,1)=2, F(1,8,1)=2, F(3,8,1)=2, F(4,8,1)=0 - число бит, зависящих от U (УАМ).

Окончательный вид таблицы F следующий:

q	V1/U	V2	q1*/U	q2*/U	МТ	АМК	УАМ
q1			q1/^u1	q2/u1	0	1	2
q2	v1		q3/u2		0	0	2
q3			q4/u1		0	2	2
q4			q1/u3	q3/^u3	0	3	0

Таблица истинности МУАМК UVH(I,J):

UVH(1,0)=0 UVH(1,2)=^u1

UVH(2.0)=0 UVH(1,3)=^u2&u3

UVH(1,1)=1 UVH(2,2)=^u3

UVH(2,1)=1

	0	1	2	3
1	0	1	^u1	^u2&u3
2	0	1	^u3	

Матрица настройки ПЗУ (массив ПЗУ):

PZU(1,1)=2

PZU(1,2)=0

PZU(0,1)=0

PZU(0,2)=1

PZU(2,1)=1

PZU(2,2)=1

PZU(3,1)=3

PZU(3,2)=2

	Микрокоманда	
Адрес ПЗУ	Номер поля 1	2
1	10	00
0	00	01
2	01	01
3	11	10

Массив FU имеет значения: FU(1)=FU(2)=2 (для управления как первым, так и вторым битом МУАМК поле микрокоманды состоит из двух разрядов).

Окончательные данные:

```
FU(3)=0        VYH1(1)=v1*4*1*(u1&^u2&u3)
FU(4)=2        VYH1(3)=v5*4*2*(^u1&u2&^u3)
FU(5)=1        VYH1(2)=v3*5*1*(u1&^u2&u3)
FU(6)=1        VYH1(4)=v6*6*1*(u1&^u2&u3)
FU(7)=1        VYH2(2)=v2*7*1
FU(8)=1        VYH2(4)=v4*8*1

PZU(0,4)=1                        Микрокоманда
PZU(0,5)=1   Адрес                Номер поля
PZU(0,6)=1     ПЗУ      1   2    3   4  5  6  7  8  9
PZU(2,4)=2              (2) (2) (0) (2)(1)(1)(1)(1)(0)
PZU(3,7)=1       1       10  00      00 0  0  0  0

PZU(3,8)=1       0       00  01      01 1  1  0  0
FU(9)=0
FUN=10           2       01  01      10 0  0  0  0
CN=1
                 3       11  10      00 0  0  1  1
```

Структурная схема устройства управления на гибкой логике показана на рис. 14. Комбинационная схема КС1 состоит из трех схем ИЛИ-НЕ для формирования инверсий входных сигналов и схемы 2И. МУАМК состоит из двух мультиплексоров на четыре информационных канала с двумя сигналами управления и неинвертирующим выходом (они служат для формирования первого и второго разрядов адреса микрокоманд (условной ее части). Выходная комбинационная схема КС2 состоит из трех элементов ИЛИ-НЕ для формирования инверсий входных сигналов, выдаваемых из выходного регистра и четырех схем ИЛИ. Схема управления не содержит счетчика циклов и, соответственно, блок синхронизации не будет содержать подсхемы блокировки, ТИ суть СИ, k=4.

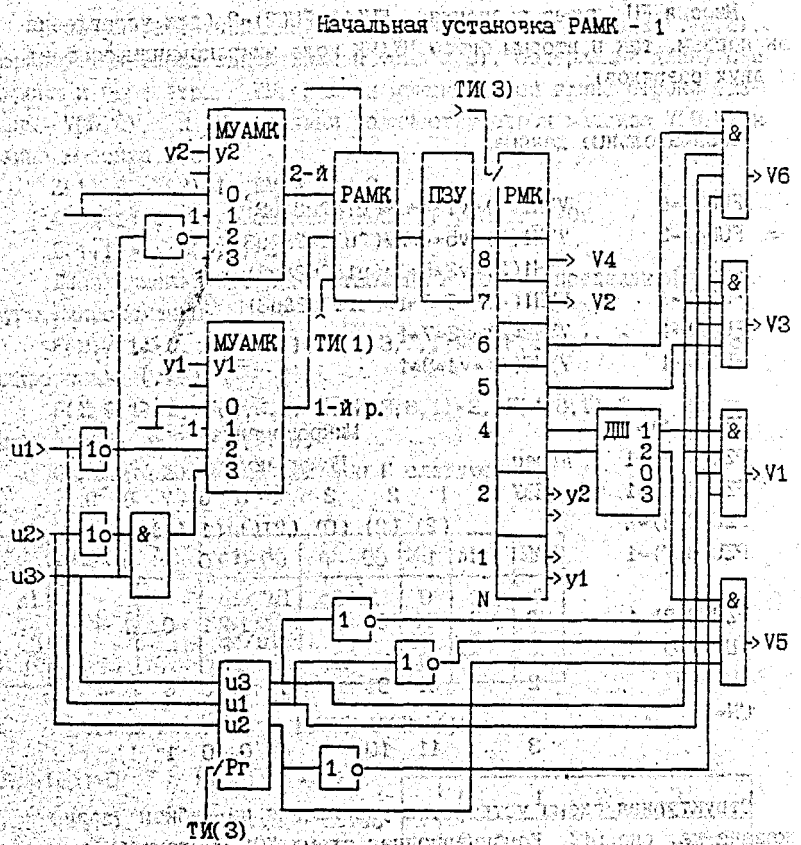


Рис. 14. Схема УУ умножителя, выполненная на гибкой логике

Непосредственный синтез. Жесткая логика

Таблица переходов и выходов дополняется нулевым столбцом F(1,6,1)=F(2,6,1)=F(3,6,1)=F(4,6,1)=0 и столбцом кодов состояний

F(1,7,1)=00

F(2,7,1)=01

```
    F(3,7,1)=11
    F(4,7,1)=10
```

Функции возбуждения элементов памяти:

```
FJ(1)=(^CI(1)&^CI(2)&u1)|(X&CI(1)&^CI(2))|(X&CI(1)&CI(2))|
     |(^u3&^CI(1)&CI(2))=^CI(1)&(^CI(2)&u1|CI(2)&^u3);
FK(1)=(X&^CI(2)&^CI(2)&u1)|(X&^CI(1)&^CI(2)&^u1)|(CI(1)&CI(2))|
     |(X&^CI(1)&CI(2)&^u3)|(X&^u2&u3&CI(2))=CI(1)&CI(2);
FJ(2)=(^CI(2)&CI(1))|(X&CI(2)&CI(1))|(X&CI(2)&^CI(1)&^u3)|
     |(X&CI(2)&^CI(1)&^u2&^u3)=^CI(2)&CI(1);
FK(2)=(X&u1&^CI(2)&^CI1)|(X&^u1^CI(2)&^CI1)|(X&CI(1)&^CI(2))|
     |(CI(2)&^CI(1)&^u2&u3)=CI(2)&^CI(1)&^u2&u3;
```

В этих функциях, некоторые элементарные конъюнкции имеют в своем составе булеву переменную X, которая может в них принимать произвольное значение. Такие элементарные конъюнкции являются несущественными в том смысле, что удаление любой комбинации таких конъюнкций не нарушает требуемой функции выхода. По сути, такое описание функции возбуждения задает множество всевозможных функций, если вместо X поставить 0 или 1.

Функции выходов:

```
    FV1(1)=(^CI(2)&CI(1)&u1&^u2&u3)  - функция выхода v1,
    FV1(2)=(^CI(2)&CI(1)&u1&^u2&u3)  -                 v3,
    FV1(3)=(CI(2)&CI(1)&^u1&u2&^u3)  -                 v5,
    FV1(4)=(CI(2)&CI(1)&u1&^u2&u3)   -                 v6.
    FV2(2)=CI(2)&^CI(1)              -                 v2,
    FV2(4)=CI(2)&^CI(1))             -                 v4.
    CN=00.
```

3.6. Подсистема синтеза устройств управления

3.6.1. Общая характеристика подсистемы

Входными данными является техническое задание на проектирование, включающее: закон функционирования цифрового устройства, представленный в виде конечного автомата или временной диаграммы; требования к проектируемому устройству по быстродействию и сложности. Выходными данными системы является схема, построенная из библиотечных элементов, минимизированная по сложности и удовлетворяющая заданному ограничению на быст-

родействие. В качестве входного и выходного языка используется международный стандарт IEEE 1076-1987 языка VHDL.

Подсистема обеспечивает:

- синтез устройства управления как на жесткой, так и микропрограммной логике (выбор варианта реализации осуществляется в зависимости от количества состояний автомата и критерия оптимизации);

- синтез комбинационной и последовательностных схем из элементов библиотек, включая построение схем в базисе ПЛМ, ПЗУ, и минимизацию ПЛМ путем сокращения промежуточных шин.

Вся информация, необходимая в процессе проектирования хранится в банке данных. Процедуры проектирования взаимодействуют между собой через банк подсистемы, записывая в него и запрашивая из него требуемые данные. На данном этапе развития подсистемы банк данных включает в себя две базы данных: базу данных внутреннего представления и базу данных библиотечных элементов.

Описание проектируемого устройства, выполненное на языке VHDL, транслятор преобразует во внутреннее представление (ВП) - промежуточную форму, удобную для эффективной обработки процедурами проектирования. Для работы процедур и обмена информацией между ними внутреннее представление проекта реализуется в виде сети структур языка программирования Си, размещаемых в оперативной памяти [34].

Сложность структур ВП и значительные объемы данных для описания проекта целиком не позволяют разместить структуры ВП в оперативной памяти и предполагают организацию базы данных внутреннего представления (БД ВП). Эта задача была решена с помощью db_Vista - набора библиотек специальных функций (макросредств), обеспечивающих эффективную разработку БД и дающих высокопроизводительные средства управления базами данных для создания прикладных программ на языке Си.

Проектирование в подсистеме основано на использовании технологических библиотек стандартных элементов БИС и СБИС [35,36]. Структура базы данных библиотечных элементов (БД БЭ) полностью соответствует структуре БД ВП, что обеспечивает единообразие представления и хранения данных, а также дает возможность проектным процедурам в своей работе не выходить за рамки внутреннего представления.

Для доступа к БД реализованы специальные процедуры работы с БД ВП, которые позволяют записать в БД и извлечь из нее все данные, относящиеся к условной единице информации в БД (ENTITY, ARCHITECTURE), процедуры поиска библиотечных элементов по заданным параметрам. Извлекаемые данные будут размещаться в оперативной памяти в виде списков в Си-структурах, описанных в [34].

3.6.2. Процедура синтеза устройств управления

В системе заложено несколько вариантов синтезируемых структур на жесткой (автоматы Мили и Мура, С-автомат) и гибкой логике, а также различные типы кодирования состояний автомата и типы элементов памяти. В зависимости от числа состояний выбирается тип структуры. Далее синтезируются основные блоки выбранной структуры, вычисляются ее характеристики и осуществляется оптимизационный синтез блоков памяти и комбинационной логики. Вычисляется невязка, по которой определяется управляющий коэффициент для оптимизирующего преобразования схемы (путем направленного перебора типов реализации, кодирования и библиотечных элементов).

Процедура синтеза включает в себя следующую последовательность операций:

- считывание из БД Entity проекта;
- определение множества архитектур проекта;
- считывание из БД Architecture проекта, содержащей функциональное описание УУ;
- преобразование внутреннего представления проекта в формат комплекса программ расчета параметров типовых устройств управления;
- выбор типа реализации (жесткая или микропрограммная логика);
- выборка из БД регистров D- и JK-триггеров (если какой-либо тип отсутствует, то выбираются триггеры;
- если тип реализации - гибкая логика, то выборка из БД ПЗУ, мультиплексоров, сдвиговых регистров, тактового генератора, счетчика и конъюнкторов;
- вызов соответствующей программы расчета типовой схемы;

- формирование новой архитектуры текущего проекта, которая задает синтезированную схему (число компонент схемы и их связи задаются типовой структурой и полученными параметрами);

- запись в БД полученной архитектуры;

- формирование Entity и Architecture для не библиотечных компонент схемы, для которых получены лишь их функциональное описание (таковыми являются комбинационные схемы формирования функций возбуждения элементов памяти и выходных полюсов). Architecture задается в форме, принятой в качестве входной для подсистемы логического синтеза;

- запись в БД и регистрация новых проектов, сформированных на предыдущем шаге.

3.6.3. Процедуры синтеза комбинационных схем

Комплекс процедур синтеза комбинационной логики решает следующие задачи:

- реализацию комбинационного блока на одной ПЛМ или ПЗУ и минимизацию площади ПЛМ;

- синтез комбинационной схемы из нескольких ПЛМ или ПЗУ;

- синтез комбинационной схемы из элементов И, ИЛИ, НЕ, И-НЕ, ИЛИ-НЕ с различным числом входов с учетом их стоимости и нагрузочных способностей выходов.

Функционирование синтезируемого комбинационного блока описывается системой полностью определенных булевых функций и задается в виде дизъюнктивных нормальных форм (ДНФ). Последние задаются совокупностью логических формул двух типов. Формулы первого типа описывают все различные элементарные конъюнкции ДНФ, в которых левые части есть внутренние переменные, поставленные в соответствие элементарным конъюнкциям, а правые части - конъюнкции входных переменных или их отрицаний. Формулы второго типа описывают вхождения элементарных конъюнкций в ДНФ, в которых левые части - выходные переменные, а правые - дизъюнкции внутренних переменных.

Так система ДНФ

y1 = x1&^x2 ! x1&^x3, y2 = x1&^x3 ! x1&x2&x3

представляется следующим образом:

```
Entity DNFVO1 is
  port  (X1,X2,X3 : in  bit;
         Y1,Y2 : out bit);
end DNFVO1;
 Architecture PLMO1 of DNFVO1
  is begin
   process (X1,X2,X3)
    variable k1,k2,k3 : bit;
    begin
      k1 := x1 and not x2 ;
      k2 := x1 and not  x3 ;
      k3 := x1 and x2 and x3 ;
      y1 <= k1 or k2;
      y2 <= k2 or k3;
  End process;
End PLMO1;
```

ДНФ на множестве полных элементарных конъюнкций может быть задана с использованием таких конструкций VHDL как IF ... THEN :

```
Entity EN is
 port ( Y  : in bit_vector (0 to 3);
        Y1 : in bit;
        X  : out bit_vector (0 to 1));
end EN;

 Architecture        EX831 of EN  is
   begin
  MAIN : process (Y, Y1)
    begin
     if Y=B"0101" then  X <= B"01"; end if;
     if Y=B"1101" then  X <= B"11"; end if;
   end process MAIN;
  end EX831;
```

Эту же ДНФ можно описать при помощи конструкции CASE языка VHDL :

```
Entity EN is
 port ( Y  : in bit_vector (0 to 3);
        Y1 : in bit;
        X  : out bit_vector (0 to 1));
end EN;
 Architecture        EX831 of EN  is
   begin
  MAIN : process (Y, Y1)
    begin
     case Y is
       when B"0101"  =>   X <= B"01";
       when B"1101"  =>   X <= B"11";
     end case;
  end process MAIN;
  end EX831;
```

В основу процедур синтеза положены алгоритмы декомпозиции, ориентированные на минимизацию числа элементов синтезируемых схем [37-40]. Процедуры реализованы несколькими конкурирующими программами, и в качестве проектного решения выбирается результат работы той программы, которая дает лучшую по заданному критерию схему.

Каждая из перечисленных задач решается выполнением последовательности из трех проектных процедур.

Первая процедура осуществляет подготовку данных для синтеза:

- считывает из БД Entity проекта;
- определяет множество архитектур проекта;
- считывает из БД Architecture проекта, содержащую функциональное описание проекта;
- функциональное описание преобразуется в матричную форму, являющейся входной для второй процедуры;
- считывает из БД требуемые базисные элементы.

Вторая процедура выполняет собственно синтез схемы.

Третья процедура осуществляет формирование результата синтеза:

- формируется новая архитектура проекта, содержащая описание синтезированной схемы и записывается в БД;
- формируются новые проекты (их число определяется числом

ПЛМ или ПЗУ в схеме);

- формируется DEMOS-описание схемы и запись его в рабочий каталог.

При нормальном завершении всех процедур выполняется запись проекта в банк данных.

Функционирование ПЛМ и ПЗУ представляется в виде ДНФ, а их структура - матрицами настройки (программирования).

Синтезированная схема описывается на языке VHDL во внутреннем представлении и в формате системы моделирования ДЕМОС.

4. СИНТЕЗ СИСТОЛИЧЕСКИХ ВЫЧИСЛИТЕЛЕЙ ДЛЯ ДВУМЕРНЫХ ПРЕОБРАЗОВАНИЙ СИГНАЛОВ

4.1. Преобразование сигналов в базисе двумерных функций на линейных систолических вычислителях

В последние годы для решения различных задач цифровой обработки двумерных сигналов широкое распространение получили методы, основанные на преобразованиях Уолша, Фурье, входящих в обобщенный базис функций Виленкина-Крестенсона (ВКФ) [41]. Двумерное преобразование можно выполнить, используя двумерные функции в требуемом базисе. В этом случае формируется одна из N^2 (где N - размерность преобразования по каждой координате) двумерных ВКФ, а соответствующие коэффициенты преобразования вычисляются следующим образом:

$$f_{mn} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} s_{ij} \cdot v_{ij}^{mn} \qquad m,n = \overline{0,N-1} \qquad (1)$$

где s_{ij} и v_{ij}^{mn} - соответствующие элементы матрицы значений сигнала и двумерной функции Виленкина-Крестенсона $V(m,n,x,y)$. Двумерные функции $V(m,n,x,y)$ можно представить в удобном для их формирования виде

$$V(m,n,x,y) = V(m,x) \otimes V(n,y), \qquad (2)$$

где $V(m,x)$ и $V(n,y)$ - вектор-строка и вектор-столбец соответствующих ВКФ,

$\otimes$ - символ кронекеровского произведения.

Соотношение (2) можно использовать для построения двумер-

ных функций ВКФ.

Вычисления в соответствии с (1) хорошо реализуются на систолических процессорах, которые обеспечивают решение задач в реальном времени и могут быть изготовлены по технологии СБИС [42]. При этом наиболее просто выполнять двумерные преобразования на линейных систолических вычислителях. Структура линейного систолического процессора зависит от способа организации потока исходных данных, а также от количества вычислительных каналов. На рис. 15 показана структура одноканального систолического вычислителя для размерности N = 4. Процессор данного типа имеет N вычислительных ячеек (процессорных элементов ПЭ), а для получения одного коэффициента требуется $T_0 = 2N-1$ тактов. Вычислительная ячейка содержит умножитель, сумматор и регистр, предназначенный для хранения в каждом такте получаемого результата.

Ниже представлен рекурсивный алгоритм вычисления (1).

$$C_j^0 = 0;$$

$$C_j^k = C_j^{k-1} + s_{kj}\, v_{kj}^{mn} \qquad k,j = \overline{1,N}, \quad N = 2^n, \tag{3}$$

где k – номер шага вычислений.

$$f_{mn} = \sum_{j=1}^{N} C_j^N. \tag{4}$$

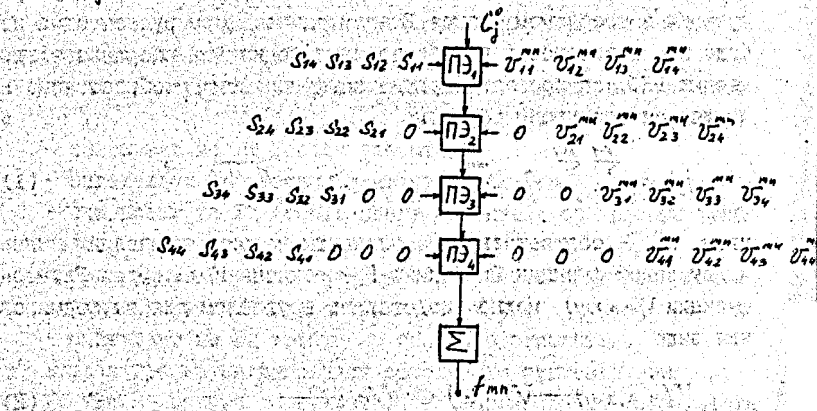


Рис. 15. Структура одноканального систолического вычислителя

Для повышения быстродействия можно использовать метод объединения элементов входных массивов в группы по 2^p элемен-

тов. В этом случае для выполнения вычислений требуется 2^{n-p} процессорных элемента ($N=2^n$, а $p=\overline{0,n-1}$), а также $T_p=\frac{2^{p+1}}{2^p}N-1$ временных такта [43].

На рис. 16 приведена зависимость числа тактов T, необходимых для вычислений от числа элементов входных массивов в группе при различных N.

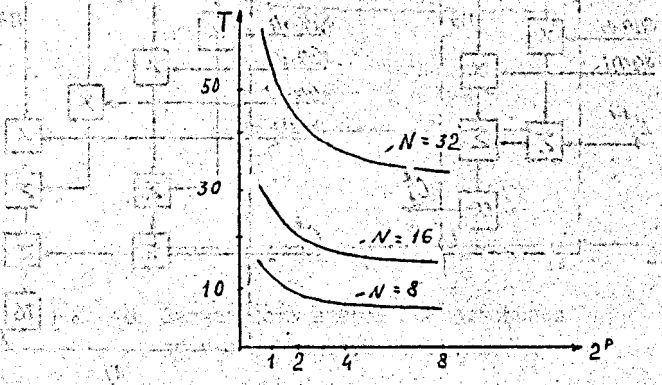


Рис. 16. Зависимость временных затрат от числа элементов в группе

Как видно из рис. 16, значительный выигрыш в быстродействии (уменьшение числа вычислительных тактов) достигается при p, равном один и два. Дальнейшее увеличение приводит к тому, что значение T ассимптотически приближается к минимальной величине, которая равна N.

4.2. Времясложностные характеристики линейных систолических процессоров двумерных преобразований

Объединение элементов входных массивов в группы по 2^p элемента приводит к уменьшению числа процессорных элементов в вычислителе. Одновременно происходит усложнение структуры ПЭ. При этом сложность функциональных узлов ПЭ при обработке s-разрядных данных можно оценить следующим образом: регистр – s, сумматор – $3s$, умножитель – s^2 условных единиц [44]. Тогда аппаратные затраты на процессорный элемент составляют

$$A = 2^p\left[S^2 + \left(3 + \frac{1}{2^p}\right)S\right]$$ условных единиц. (5)

На рис. 17а и 17б показаны структуры процессорных элементов для вычисления двумерного преобразования ВКФ при $p=1$ и $p=2$.

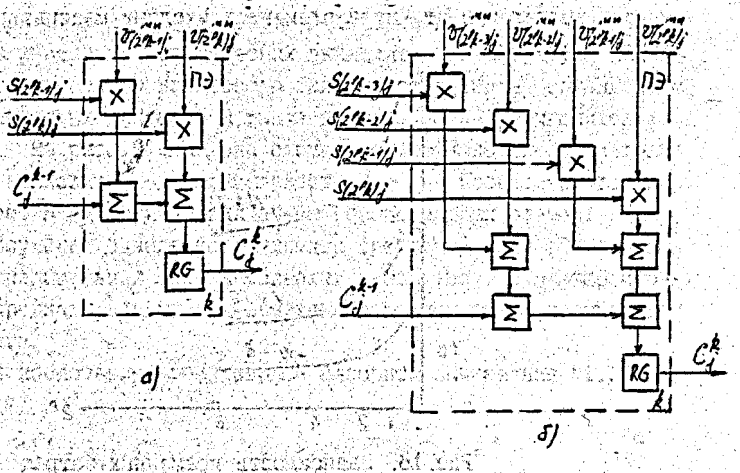


Рис. 17. Структуры процессорных элементов

Аппаратные затраты на систолический вычислитель определяются следующим выражением:

$$A_{z0} = N\left[S^2 + \left(3 + \frac{1}{2^p}\right)S\right]$$ условных единиц. (6)

На рис. 18 представлена зависимость аппаратных затрат A от числа элементов в группе входных данных. Графики получены при $S=4$ для $N=8$ и $N=16$ (кривые 1 и 2) и при $S=8$ для $N=8$ (кривая 3). На графиках видно, что при $p=1$ и $p=2$ происходит заметное уменьшение A_z, а в случае дальнейшего увеличения p аппаратные затраты стремятся к величине $A_z = N(S^2+3S)$. Увеличение N приводит к увеличению кривизны графика на интервале с $p=\overline{0,2}$.

На рис. 19 показан двухканальный линейный систолический вычислитель двумерного преобразования, который получается в результате деления массива входных данных на две равные части, параллельно направлению их ввода в одноканальный процессор, показанный на рис. 15.

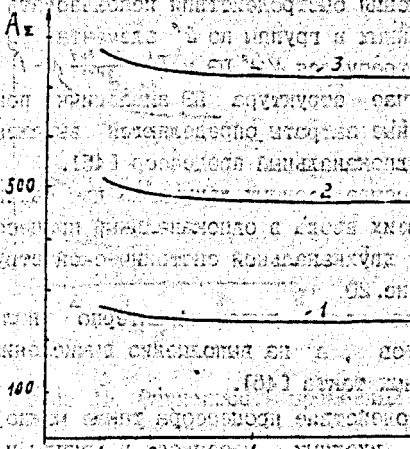


Рис. 18. Зависимость затрат оборудования от числа элементов в группе

Такой вычислитель характеризуется наличием N процессорных элементов и временными затратами в $1,5N-1$ временных такта.

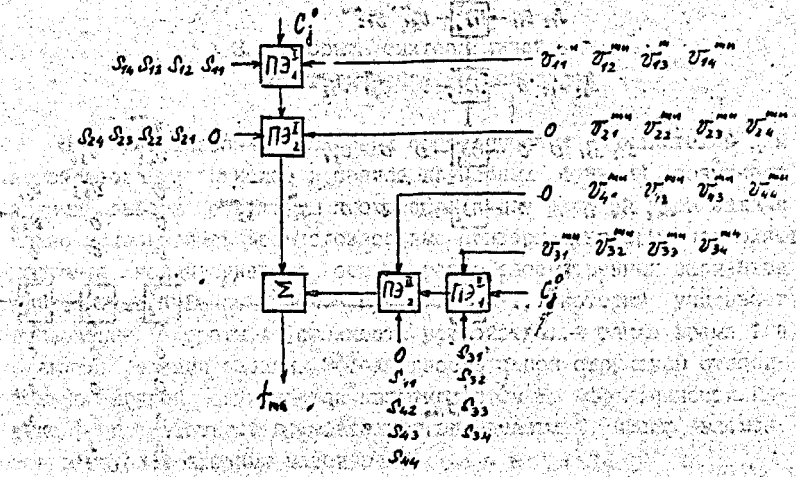


Рис. 19. Структура двухканального вычислителя первого типа

Если для повышения быстродействия использовать метод объединения входных данных в группы по 2^p элемента, то на организацию вычислений потребуется $N/2^p$ ПЭ и $T_p' = \frac{2^{p+1}+1}{2^{p+1}} N - 1$ временных такта. В этом случае структура ПЭ аналогична показанным на рис. 17, а аппаратные затраты определяются выражением (6) и равны затратам на одноканальный процессор [45].

При делении массива входных данных на две части перпендикулярно направлению их ввода в одноканальный процессор получается второй вариант двухканальной систолической структуры, которая показана на рис. 20.

Для вычислителя этого типа характерно наличие $2N$ процессорных элементов, а на выполнение вычислений требуется $3/2N-1$ временных такта [46].

Повысить быстродействие процессора также можно путем объединения элементов входных массивов в группы и усложнения структуры вычислительной ячейки. На организацию вычислений требуется $2N/2^p$ ПЭ и $T_p'' = \frac{2^{p+1}+1}{2^p} N - 1$ временных такта. Процесс повышения эффективности вычислений в двухканальных систолических процессорах показан на рис. 21, где а) -зависимость для процессора первого типа, а б) -второго.

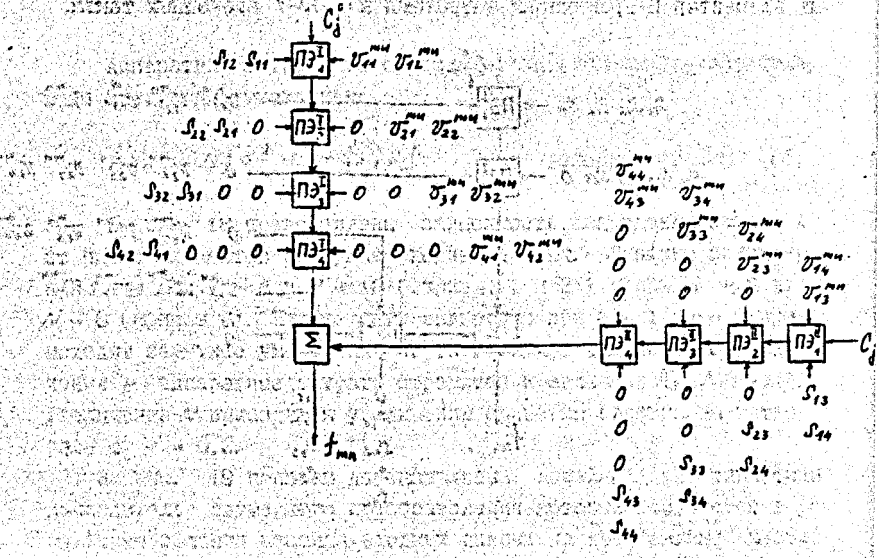


Рис. 20. Структура двухканального вычислителя второго типа

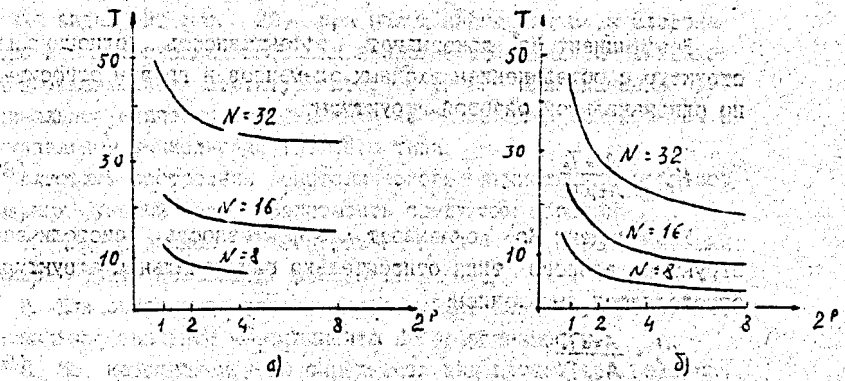


Рис. 21. Зависимость временных затрат от числа элементов в группе

Аппаратные затраты на систолический процессор второго типа определяются следующим образом:

$$A''_{\Sigma} = 2N\left[S^2 + (3 + \frac{1}{2^p})S\right]$$ условных единиц. (7)

4.3. Алгоритм синтеза линейных систолических вычислителей

При проектировании линейных систолических вычислителей для преобразования сигналов в базисе двумерных функций возникает задача выбора структуры процессора. Для решения этой задачи можно использовать времясложностные критерии, которые позволяют получить количественные оценки, ранее рассмотренных вариантов вычислителей. Воспользуемся критерием А·Т, который учитывает аппаратные затраты А (сложность устройства), а также время Т в (тактах) решения задачи. Эффективность одной структуры относительно другой оценивается коэффициентом Кр эффективности. Индекс р соответствует показателю р, определяющему число элементов в группах входных массивов.

Коэффициент Кр° показывает эффективность одноканальных структур с объединением входных элементов в группы относительно одноканальной базовой структуры:

$$K_p^{\circ} = \frac{A_{x0} T_c}{A_{xp} T_p} \qquad (8)$$

Коэффициент Кр' показывает эффективность систолических структур второго типа относительно одноканальной структуры и определяется выражением:

$$K_p' = \frac{A_{x0} T_c}{A_{xp} T_p'} \qquad (9)$$

Эффективность двухканального процессора третьего типа по отношению к одноканальному вычислителю определяет коэффициент Кр", который находится следующим образом:

$$K_p'' = \frac{A_{x0} T_c}{A_{xp}'' T_p''} \qquad (10)$$

Зависимости коэффициентов Кр° (кривая 1), Кр' (кривая 2) и Кр" (кривая 3) эффективности от показателя р представлены на рис. 22. Значения коэффициентов получены для S = 8 и N = 128.

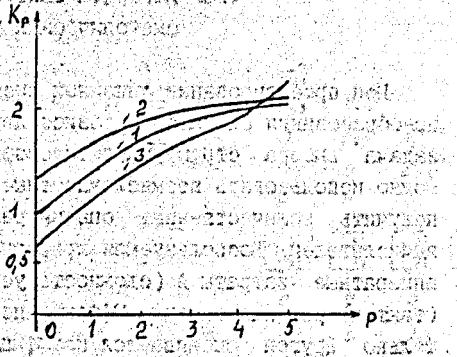


Рис. 22. Зависимость коэффициента эффективности от числа элементов в группе

Как видно из рис. 22, при малых значениях p, а следовательно и малых входных массивах, наиболее эффективной оказывается двухканальная структура второго типа. При значениях p>5 максимальное значение коэффициента эффективности достигается в двухканальном вычислителе третьего типа.

Алгоритм построения систолического вычислителя в базисе двумерных функций можно представить следующим образом:

1. Для заданной размерности $N \times N$ ($N = 2^n$) обрабатываемых массивов задать показатель $p \in [0, n-1]$.

2. Для выбранного показателя p найти, используя рис. 22, максимальное значение коэффициента Kp эффективности.

3. По коэффициенту Kp определить вид процессора (одноканальный или двухканальный первого или второго типа)

4. Вычислить число R элементов в группе $R = 2^p$.

5. Найти общее количество процессорных элементов в вычислителе, равное N/R.

6. Определить пределы изменения номера k элемента в группе $k = \overline{1,R}$.

7. Последовательно задавая номера g ($g = \overline{1,N/R}$) процессорного элемента и номера k элемента в группе ($k = \overline{1,R}$), определять $(k+1)$-й и $(k+1+R)$-й входы g-го ПЭ и их связи с $[k + R(g-1)]$-ми выходами соответственно блока сдвиговых регистров (обеспечивающего формирование массива значений сигнала в требуемой форме) и генератора двумерных базисных функций.

При этом выходы $(N\ 2R)$-го и $(N-R)$-го ПЭ соединены с входами сумматора (для двухканального процессора), а выходы и первые входы процессорных элементов используются для построения вычислительных каналов.

Структура, например, двухканального вычислителя, получаемого в соответствии с предложенным алгоритмом, представлена на рис. 23 и содержит группы сдвиговых регистров 1 (осуществляющих задержку на $(i-1)$ такт, где i - номер группы), процессорные элементы 2, генератор 3 двумерных базисных функций, сумматор 4 и накапливающий сумматор 5.

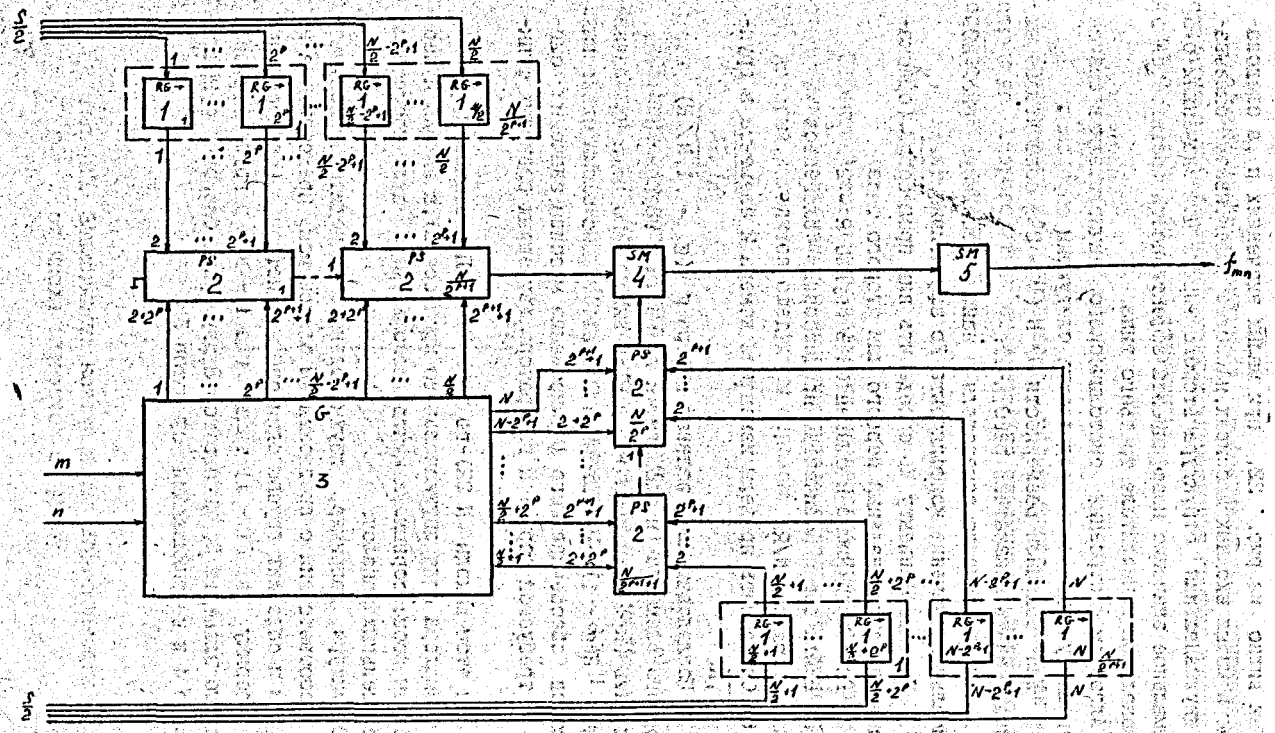


Рис. 23. Обобщенная структура двухканального систолического вычислителя

5. СИНТЕЗ ОТКАЗОУСТОЙЧИВЫХ СХЕМ СБИС

5.1. Постановка задачи

Необходимость обеспечения отказоустойчивости схем реализуемых на СБИС и СБИС-пластинах обусловлена высокими требованиями к их надежности и выходу годных. При этом отказоустойчивость может использоваться для перехода на более высокие степени интеграции микросхем за счет достижения приемлемого выхода годных. Исходными данными для проектирования отказоустойчивых схем в САПР СБИС является функционально-необходимая схема, т.е. исходная без средств отказоустойчивости схема, необходимая и достаточная для выполнения ею заданных функций. В качестве функционально-необходимых схем в данном разделе рассматриваются неоднородные схемы с нерегулярными связями. В [47] показано, что эффективность резервирования в СБИС находится в большой зависимости от выбранных методов, уровня и обьема обеспечения отказоустойчивости, а также типа, резервируемых модулей схемы. Поэтому задача синтеза отказоустойчивых схем состоит в преобразовании исходного варианта схемы на основе выбора при заданных ограничениях оптимальных обьема, уровня и методов обеспечения отказоустойчивости для соответствующих модулей схемы. При этом средства отказоустойчивости могут быть ориентированы как на нейтрализацию производственных отказов, с целью эффективного повышения выхода годных СБИС, так и для нейтрализации эксплуатационных отказов с целью улучшения показателей надежности схем. Под эффективным повышением выхода годных схем при этом понимается такое повышение, при котором увеличивается схем кристаллов с пластины. В общем случае процесс проектирования разбивается на два этапа. На первом этапе производится проектирование отказоустойчивой схемы с целью эффективного повышения выхода годных СБИС, а на втором этапе происходит распределение средств отказоустойчивости на кристалле с целью эффективного повышения как выхода годных, так и вероятности безотказной работы (ВБР) схем.

5.2. Проектирование отказоустойчивых схем с целью эффективного повышения выхода годных схем

Рассмотрим проектирование отказоустойчивых схем для неоднородных структур СБИС на фиксированном иерархическом уровне схемы. Процесс проектирования в общем случае состоит из двух этапов. Первый этап связан с генерацией возможных вариантов обеспечения отказоустойчивости в схеме, а второй этап - с выбором оптимальной структуры отказоустойчивой схемы в соответствии с требованиями технического задания. В силу ограничения по площади кристалла в качестве методов обеспечения отказоустойчивости будем рассматривать дублирование и трехкратное мажоритарное резервирование. На рис.24 и 25 показан упрощенный пример применения таких методов резервирования для одного модуля схемы. При этом в схеме на рис. 24 предполагается, что имеется внешнее тестирующее устройство, которое соответствующим образом устанавливает триггера в процессе тестирования схемы. Тогда для синтеза отказоустойчивой схемы необходимо найти такой вектор $R_o=(R_{jo}/j=1,N)$, где R_{jo} - оптимальное количество резервных элементов для J-го модуля схемы, N - общее количество модулей на кристалле, чтобы обеспечивалось максимальное значение показателя качества резервирования:

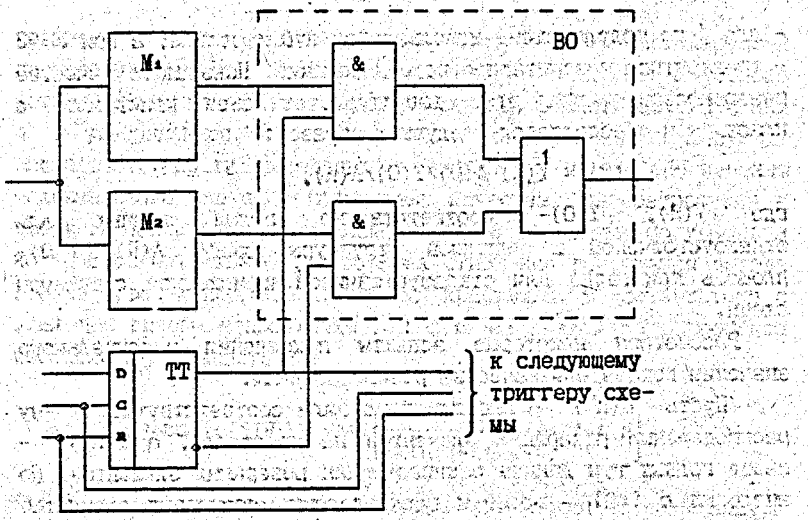
$$\max_{R \in R} \{F(R)\} ,$$

где $R=(R_j/j=1,N)$ - множество всех возможных решений данной задачи, таких, что

$$A(R) \le A_m ,$$

где A_m - максимальная площадь кристалла.

Так как в качестве методов резервирования мы рассматриваем дублирование и трехкратное мажоритарное резервирование, то $R_{jo}=0,2$. Если $R_{jo}=0$, то нецелесообразно использовать для J -го модуля схемы резервирование, при $R_{jo}=1$



ВО – восстанавливающий орган

M_1, M_2 – основной и резервный модуль

Рис.24. Дублирование модуля схемы.

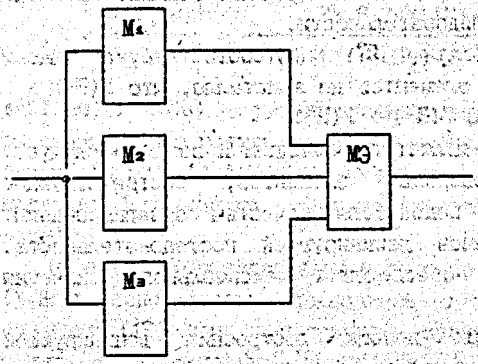


Рис.25. Трехкратное мажоритарное резервирование.

– для j-го модуля схемы используется дублирование, а при $R_{j0}=2$ – трехкратное мажоритарное резервирование. Показатель качества резервирования при этом характеризует сьем кристаллов с пластины и определяется следующим образом:

$$F(R) = Y(R)\ A(0)/Y(0)/A(R),$$

где $Y(R)$, $Y(0)$– соответственно выход годных для отказоустойчивой и исходной структуры схемы; $A(R)$, $A(0)$– площадь кристалла для отказоустойчивой и исходной структуры схемы.

Рассмотрим некоторые аспекты нахождения максимального значения показателя качества резервирования.

Пусть $A(R^i)$ – площадь кристалла соответствующая i -му распределению резервных элементов на кристалле, а $Y(R^i)$ – выход годных при данном распределении резервных элементов. По аналогии с [48] , назовем распределение резервных элементов R_0 , при котором $Y(R^0) > Y(R^i)$ и $A(R^0) \le A(R^i)$ доминирующим над распределением R^i , а соответственно последовательность, состоящую из доминирующих распределений резервных элементов, назовем доминирующей последовательностью, если ни один из ее векторов не доминируется строго никаким другим вектором. Понятно, что распределение резервных элементов, при котором обеспечивается максимальное значение выхода годных, является членом доминирующей последовательности.

Утверждение 1. Если R^0 соответствует такому распределению резервных элементов на кристалле, что $Y(R^0) > Y(R^i)$ и $A(R^0) \le A(R^i)$, то $F(R^0) > F(R^i)$.

Данное утверждение является очевидным и из него следует, что распределение резервных элементов, соответствующее максимальному значению показателя качества резервирования, является одним из членов доминирующей последовательности, построенной исходя из обеспечения максимального значения выхода годных схем.

Таким образом, при помощи построения доминирующей последовательности, исходя из обеспечения максимального значения выхода годных схем, мы можем найти распределение резервных элементов соответствующее максимальному значению показателя качества резервирования. Это позволяет сократить

процесс вычислений. Для построения доминирующей последовательности будем использовать методику, в основе которой лежит метод наискорейшего спуска. В соответствии с ним и утверждением 1 для нахождения доминирующей последовательности резервных элементов с целью обеспечения максимального значения показателя качества резервирования в схему на каждом шаге добавляется такой резервный модуль, при котором обеспечивается наибольшее отношение приращения выхода годных схемы к приращению площади кристалла. В соответствии с этим для выбора направления движения на каждом шаге процесса необходимо определить

$$\delta(Y(R_{j_0}))=\max\{\delta(Y(R_j))\}=\max\left\{\frac{Y(R_j)-Y(R_j-1)}{A(R_j)-A(R_j-1)}/j=1,N\right\} \quad (11)$$

где $Y(R_j)$ - выход годных схемы при добавлении в j-й модуль резервного элемента; $Y(R_j-1)$ - выход годных схемы до добавления в соответствующий модуль схемы резервного элемента; $A(R_j-1), A(R_j)$ - соответственно площадь схемы на кристалле до и после добавления в j-й модуль схемы резервного элемента. Так как выход годных всей схемы можно определить исходя из выхода годных модулей схемы, то

$$\gamma(R_j)=\frac{\delta(Y(R_j))}{Y(R_j-1)}=\frac{Y_j(R_j)-Y_j(R_j-1)}{Y_j(R_j-1)((A_j(R_j)-A_j(R_j-1))}, \quad (12)$$

где $Y_j(R_j)$ и $A_j(R_j)$ - соответственно выход годных и площадь на кристалле j-го модуля схемы при добавлении в него резервного элемента; $Y_j(R_j-1), A_j(R_j-1)$ - выход годных и площадь на кристалле j-го модуля схемы до добавления в него резервного элемента.

Очевидно, что содержание метода не изменится, если вместо $\delta(Y(R_j))$ использовать показатель $\gamma(R_j)$. Как следует из выражения (12) данный показатель зависит только от выхода годных и площади на кристалле j-го модуля схемы при резервировании и до резервирования его.

Утверждение 2. Так как дублирование и трехкратное мажоритарное резервирование являются методами одного класса эквивалентности, то $Y(R)$, где $R=\{R_j/j=1,N\}$ в общем случае не

является монотонной.

Здесь под методами резервирования одного класса эквивалентности понимаются методы нейтрализующие одинаковое количество отказов элементов схемы. Понятно, что при переходе от дублирования к трехкратному мажоритарному резервированию выход годных схемы может уменьшаться.

Основываясь на приведенных выше теоретических посылках можно предложить следующий алгоритм поиска оптимальных методов и типа резервируемых элементов схемы для полностью неоднородных структур СБИС.

1. Определяем для всех модулей схемы показатели выбора направления движения при дублировании и трехкратном мажоритарном резервировании.

$$\gamma_d(R_j) = \frac{Y_j(1) - Y_j(0)}{Y_j(0)(A_j(1)-A_j(0))}, \quad j=1,N \qquad (13)$$

$$\gamma_m(R_j) = \frac{Y_j(2) - Y_j(0)}{Y_j(0)(A_j(2)-A_j(0))}, \quad j=1,N \qquad (14)$$

2. Формируем первый вектор выбора направления движения:

$$\Gamma(1) = \{\gamma_d(R_j), \gamma_m(R_j)/j=1,N\}.$$

3. Определяем максимальную компоненту вектора $\Gamma(1)$:

$$\gamma(R_{k_0}) = \max\{\Gamma(1)/j=1,N\}.$$

В результате этой операции будет получен тип резервируемого элемента и соответствующий для него метод резервирования.

4. Вычисляем показатель качества резервирования

$$F(R_{k_0}) = Y(R_{k_0})A(0)/Y(0)/A(R_{k_0}).$$

5. Производим следующий шаг, для этого исключаем из вектора $\Gamma(1)$ показатель направления для модуля зарезервированного на предыдущем шаге:

$$\Gamma(2) = \Gamma(1) - \gamma(R_{k_0})$$

и продолжаем процесс начиная с п. 3. Данная процедура продолжается до тех пор, пока не будут достигнуты ограничения или не станет максимальным значение показателя качества резервирования. Контролируя показатель качества резервирования

при этом можно сделать вывод о целесообразности введения резерва в кристалл вообще или выбрать на определенном уровне резервирования оптимальный метод резервирования для соответствующих элементов схемы.

5.3. Распределение средств отказоустойчивости на кристалле

В общем случае, как уже отмечалось выше, средства отказоустойчивости должны быть ориентированы как на нейтрализацию призводственных отказов с целью повышения выхода годных, так и эксплуатационных отказов с целью повышения вероятности безотказной работы схем. Пусть Ry – распределение резервных элементов на кристалле для повышения выхода годных схем, а Rp – распределение резервных элементов на кристалле для повышения вероятности безотказной работы схем. Тогда G(Rp)=P(Rp)/Po – коэффициент повышения надежности схем, где P(Rp) – вероятность безотказной работы схемы при соответствующем обьеме резервных элементов, Po – вероятность безотказной работы функционально-необходимой структуры схемы. Назовем распределение резерва на кристалле сбалансированным, если F(Ry) > 1, G(Rp) > 1 и P > Pd, а при F(Ry) > 1 и P > Pd – приоритетным по выходу годных, соответственно при G(Rp) > 1, Y > Yd, P > Pd– приоритетным по надежности, где Pd, Yd – заданные в техническом задании требования по вероятности безотказной работы и выходу годных схем.

Стратегия распределения. Распределение необходимо производить из такого критерия, при применении которого вероятность безотказной работы схемы максимально увеличивается, а показатель качества резервирования минимально уменьшается, т.е. обеспечивается $\{\max(\Delta P)\}$ и $\{\min(\Delta F)\}$.

Исходя из этого для распределения резерва на кристалле можно воспользоваться методом наискорейшего спуска, суть которого в данном случае будет состоять в том, что на каждом шаге процесса для повышения вероятности безотказной работы в схему добавляется такой очередной метод обеспечения отказоустойчивости к соответствующему типу модуля схемы, при котором обеспечивается наибольшее отношение приращения

коэффициента повышения надежности схемы к приращению показателя качества резервирования. В соответствии с этим для выбора направления движения на каждом шаге процесса необходимо определить

$$\delta(P(R_{j_0}))=\max\{\delta(P(R_j))\}=\max\left\{\frac{P(R_{pj})-P(R_{pj-1})}{F(R_{yj})-F(R_{yj-1})}/J=1,N\right\}, \quad (15)$$

где $P(R_{pj})$, $F(R_{yj-1})$ – соответственно коэффициент повышения надежности схемы и показатель качества резервирования при добавлении в схему J-го резервного модуля для повышения ВБР, $P(R_{pj-1})$, $F(R_{yj})$ – соответственно коэффициент повышения надежности схемы и показатель качества резервирования до резервирования J-го модуля схемы.

Утверждение 3. Если какой-то метод обеспечения отказоустойчивости для J-го модуля схемы дает максимальный выигрыш в эффективном повышении выхода годных схем, то аналогичный метод обеспечения отказоустойчивости для соответствующего модуля схемы даст максимальный выигрыш в повышении ВБР схемы, в предположении, что данный метод резервирования может использоваться как для повышения ВГ, так и ВБР.

Из данного утверждения следует, что распределение резерва необходимо производить исходя из группы элементов зарезервированных для повышения ВГ и общая площадь кристалла в этом случае остается постоянной.

Утверждение 4. При распределении резерва на кристалле с целью повышения ВБР, если $A(R)=\text{const}$ и $Y(R_y^0-1)<Y(R_y^1-1)$, то $F(R_y^0-1)<F(R_y^1-1)$ и наоборот.

Данное утверждение является очевидным и из него следует, что направление движения можно определять на основе следующего показателя:

$$S(R_{pj})=\frac{(P_j(R_{pj})-P_j(R_{pj-1}))\ Y_j(R_{yj})}{(Y_j(R_{yj})-Y_j(R_{yj-1}))\qquad Y_j(R_{yj-1})}, \quad (16)$$

где $J=1,N_y$. N_y – количество основных элементов схемы, зарезервированных для повышения выхода годных схемы.

Исходя из проведенных выше рассуждений можно предложить

следующий алгоритм проектирования отказоустойчивых схем для полностью неоднородных структур с целью эффективного повышения как ВГ, так и ВБР.

1. Производим проектирование отказоустойчивой схемы исходя из эффективного повышения выхода годных.

2. Определяем для зарезервированных модулей схемы на этапе проектирования по ВГ показатели выбора направления движения согласно выражению (16).

3. Формируем первый вектор выбора напрэвления движения:

$$G(1) = \{ S(R_{Pj}) / j=1.N_y \} .$$

4. Определяем максимальную компоненту вектора G(1)

$$S_m(R^i_{P0}) = \max\{ G(1)/j=1,N_y \}.$$

В результате этой операции получим метод обеспечения отказоустойчивости для соответствующего модуля схемы с целью повышения ВБР.

5. Вычисляем показатель качества резервирования, вероятность безотказной работы и коэффициент повышения надежности схемы.

6. Производим следующий шаг путем формирования вектора

$$G(2) = G(1) - S_m(R^i_{P0})$$

и продолжаем процесс начиная с п. 4.

Данная процедура продолжается до тех пор пока не будут выполнены требования одного из заданных критериев распределения (сбалансированного или приоритетного). В результате выполнения данной процедуры можно получить оптимальные методы обеспечения отказоустойчивости и тип резервируемых модулей схемы как для повышения выхода годных, так и для повышения ВБР схем.

5.4. Подсистема САПР отказоустойчивых схем

На основе созданной методологии высокоуровневого синтеза отказоустойчивых схем, рассмотренной частично в данной работе, разработана подсистема САПР отказоустойчивых схем. Структурная схема данной подсистемы изображена на рис. 26. Она может работать как в автономном режиме, так и в составе САПР СБИС верхнего уровня. Исходными данными для подсистемы САПР

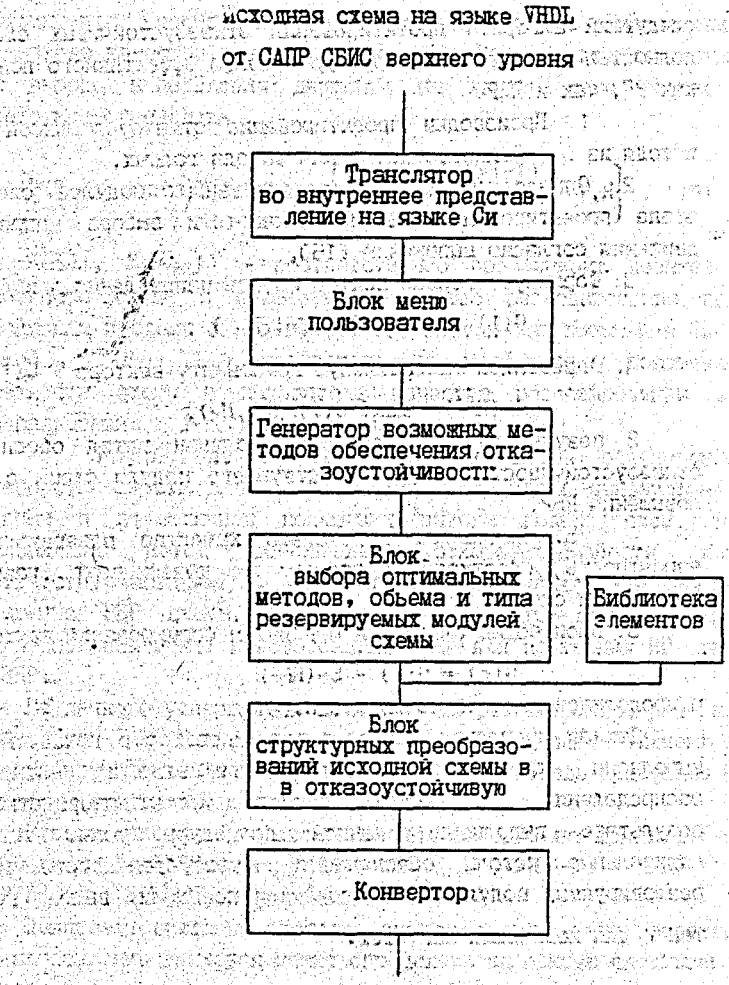


Рис. 26. Структурная схема подсистемы САПР отказоустойчивых схем

является описание функционально-необходимой схемы на языке VHDL с соответствующими параметрами и требования технического задания на проектирование. Транслятор преобразует исходное описание схемы на языке VHDL во внутреннее представление на языке Си и идентифицирует одинаковые типы модулей схемы. Блок меню пользователя запрашивает в диалоговом режиме необходимые данные для проектирования в соответствии с блок-схемой, частично изображенной на рис.27.(здесь не показан третий режим проектирования отказоустойчивых схем по выходу годных и ВБР). Генератор возможных методов обеспечения отказоустойчивости в зависимости от типа схемы задает набор возможных методов построения отказоустойчивых схем. Для полностью неоднородных схем такими методами являются трехкратное мажоритарное резервирование и дублирование. Библиотека элементов содержит элементы необходимые для построения отказоустойчивых схем, которые описаны на языке VHDL. Для неоднородной схемы - это описание восстанавливающего органа, мажоритарного элемента и триггера:

```
component DUB_1
 port(A:in bit_vector(0 to 1);G,G^ : in bit;B:out bit);
component MAJ_1
 port (A:in bit_vector(0 to 2);C: out bit);
component DIR_T
 port(G,C,R: in bit;C,C^ : out bit);
```

Модуль структурных преобразований в соответствии с результирующими данными работы блока выбора оптимальных методов, обьема и типа резервируемых модулей схемы преобразует исходную схему в отказоустойчивую. Укрупненный алгоритм работы этого блока можно представить в следующем виде:

1. Введение в схему $R_д$ триггеров для управления восстанавливающими органами, где $R_д$ - количество модулей схемы, к которым целесообразно применять дублирование.

2. В схему добавляется три входные шины для управления триггерами схемы.

3. Введение в схему $S_в$ восстанавливающих органов. При этом

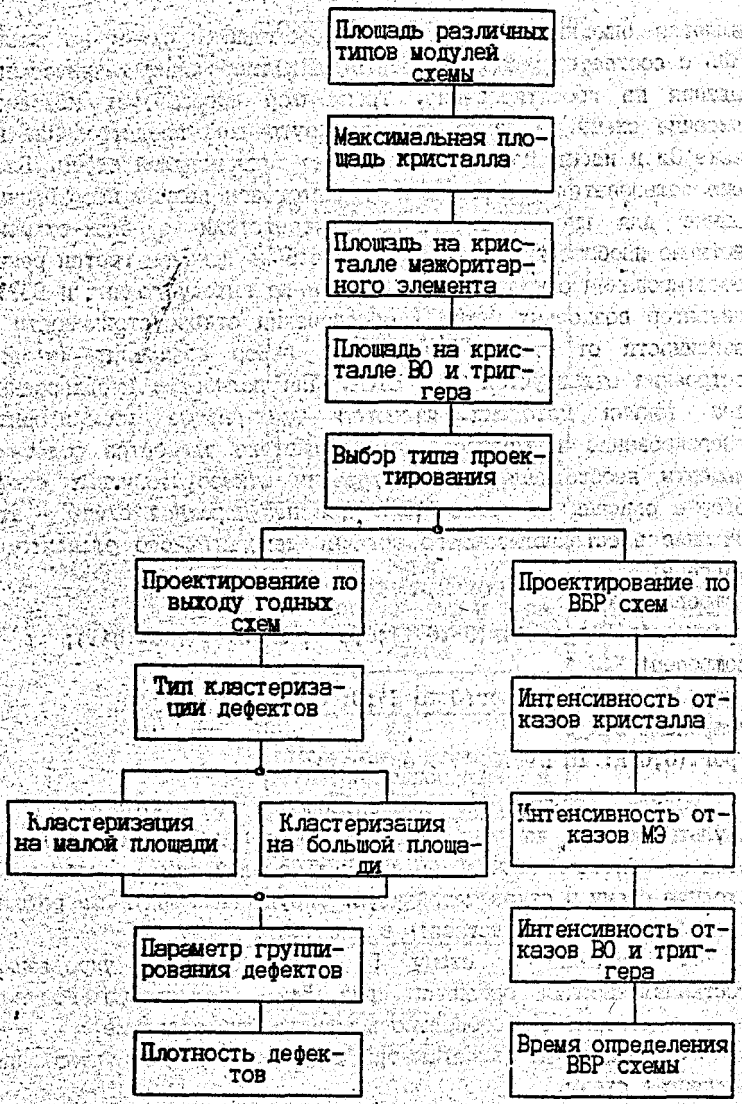


Рис. 27. Блок-схема ввода данных.

$$S_d = \sum_{i=1}^{R_d} N_i ,$$

где N_i - количество выходов i-го модуля схемы.

4. Введение в схему S_m мажоритарных элементов,

где $$S_m = \sum_{j=1}^{R_m} N_j ,$$

а R_m - количество модулей схемы, к которым применяется мажоритарное резервирование.

5. Трассировка и нумерация связей.

На основании данных работы блока структурных преобразований, конвертор преобразует внутреннее представление схемы на языке Си в отказоустойчивую схему на языке VHDL. В основе моделей для расчета выхода годных схем лежат выражения разработанные в [49] и [50].

5.5. Проектирование систолического отказоустойчивого коррелятора

Вычисление корреляционной функции сигналов является широко распространенной операцией в цифровой обработке сигналов. Она может использоваться в системах ориентации летательных аппаратов различного класса (задача сопоставления изображения шаблону), для сжатия данных, адаптивной фильтрации и т.д. Большой объем вычислений, необходимость обработки сигналов в реальном масштабе времени и векторизуемость алгоритма вычисления корреляционной функции создают предпосылки для реализации соответствующего вычислителя на систолическом массиве.

В данном разделе рассматриваются вопросы проектирования отказоустойчивого систолического процессора для вычисления одномерной корреляционной функции сигнала. При этом в отличие от известных подходов, когда средства отказоустойчивости вводятся на последнем этапе проектирования, здесь структура систолического массива, организация входного потока и ввода данных выбирались исходя из обеспечения тестопригодности и

отказоустойчивости схемы при малых аппаратных затратах.

Организация систолического массива

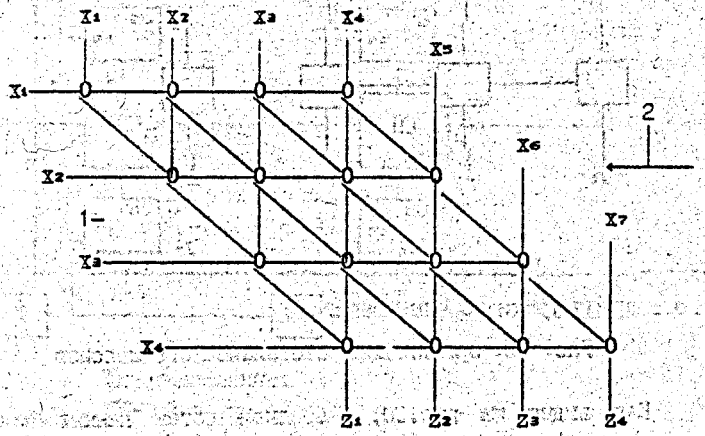
При проектировании систолического массива необходимо обеспечить решение следующих задач: высокая скорость вычислений, слайсовость структуры в соответствии с размерностью задачи, возможность простого обхода отказавших элементов, организация простого ввода входных данных, наблюдаемости и управляемости элементов систолического массива. Рассмотрим решение указанных выше задач.

Корреляционная функция характеризует тесноту стохастической связи между значениями случайного процесса в различные моменты времени и в общем случае определяется исходя из следующего выражения [31]:

$$Z(k) = \sum_{i=1}^{N} X_i \, X_{i+k} \quad ,$$

где N – длина измеряемой последовательности, определяющая точность вычислений; n – число ординат корреляционной функции; $k = \overline{0,n-1}$ – интервал сдвига случайного процесса; X_i, X_{i+k} – значения случайного процесса в моменты времени i и $i+k$.

Для проектирования систолического массива вычисления корреляционной функции воспользуемся формальным методом предложенным в [32]. В соответствии с ним первый этап проектирования состоит в построении графа зависимостей вычислений алгоритма. В результате работы над этим этапом были проанализированы различные варианты графа зависимостей и исходя из решения перечисленных выше задач был выбран граф зависимостей вычислений изображенный на рис.28. Второй этап проектирования состоит в отображении графа зависимостей в граф потока сигналов, который изображен на рис.29. Плоскость и направление проецирования показаны на рис.28. Приведенный граф зависимостей обладает свойствами систолической структуры и путем соответствующей организации входного потока данных легко получить систолический массив процессорных элементов (рис.30).



1 – плоскость проецирования

2 – направление вектора проецирования

Рис. 28. Граф зависимостей

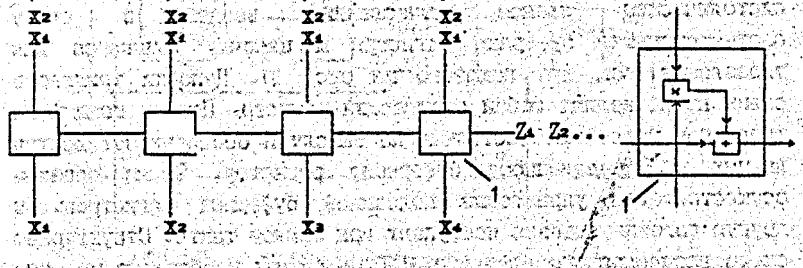
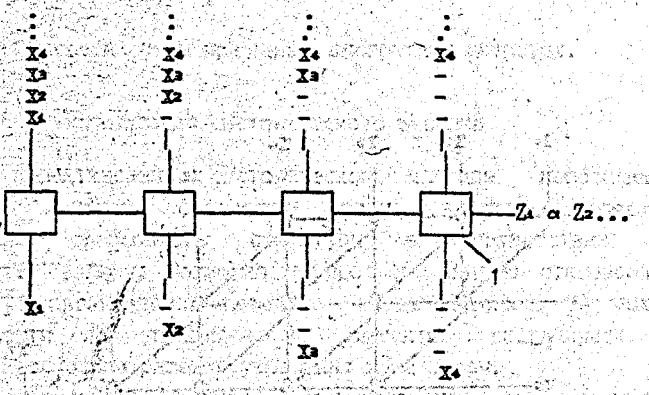


Рис. 29. Граф потока сигналов

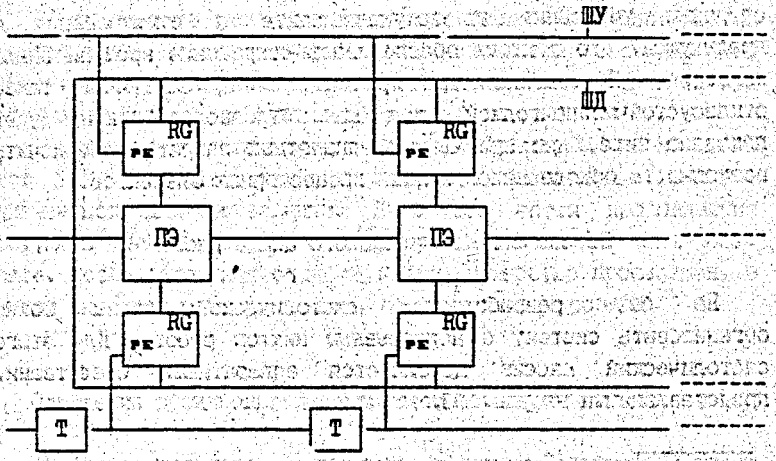


α – время такта работы конвейера

Рис. 30. Организация систолического массива

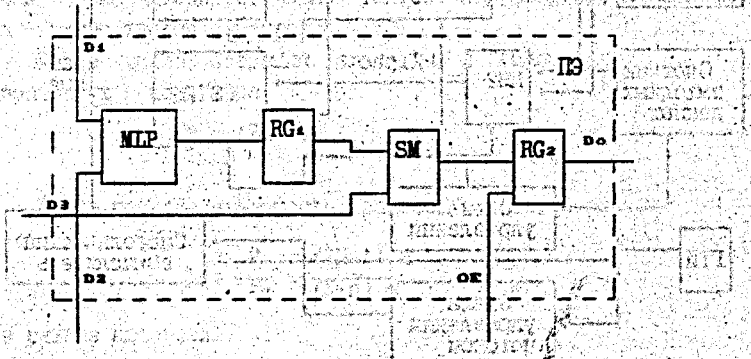
Как видно из рис.30, входной поток данных является однородным, что обеспечивает простую организацию их ввода в процессорный массив, структура является наращиваемой по N и обеспечивает любую точность по n ; данные на выходе появляются через такт конвейера; поток информации между процессорными элементами является однонаправленным. При этом такая схема обладает временным и пространственным параллелизмом.

Для организации ввода данных в соответствии с систолическим планом вычислений, введем в схему соответствующие буферные регистры и цепочку триггеров для управления ими, как показано на рис. 31. Цепочки триггеров схемы представляют собой сдвигающий регистр. Причем каждый из триггеров имеет три состояния по выходу и обеспечивает доступ данных к определенному буферному регистру. Таким образом осуществляется управление половиной буферных регистров. В другую половину данные поступают при каждом такте. Структурная схема процессорного элемента (ПЭ) при этом изображена на рис. 32, где выходной регистр RG_2 имеет три состояния. Внесение в схему регистра RG_4 обусловлено тем, что время, затрачиваемое на суммирование, превосходит время запоминания информации. А это, в свою очередь, позволяет частично совместить во времени



ШД- шина данных
ШУ- шина управления

Рис. 31. Фрагмент схемы систолического процессора



D_1, D_2 - входные данные от буферных регистров
D_3 - даные от предыдущего ПЭ
OE - сигнал управления Z-состоянием ПЭ
D_o - выходные данные к следующему ПЭ

Рис. 32. Схема процессорного элемента

операции умножения в текущем такте и суммирования в предыдущем, что в итоге обеспечивает сокращение времени цикла конвеера. Приведенная на рис.31 схема является также отказоустойчивопригодной, так как позволяет, как будет показано ниже, при приемлемых аппаратных затратах обеспечить возможность обхода неисправных процессорных элементов.

Обеспечение непрерывного цикла работы

На базе разработанной систолической схемы можно организовать систему с непрерывным циклом работы. Для этого систолический массив дополняется аппаратными средствами, представленными на рис. 33.

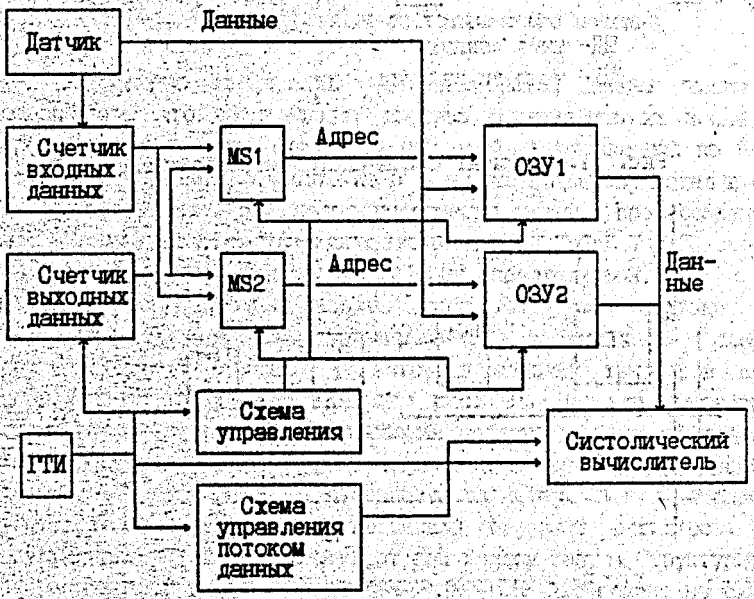


Рис. 33. Система с непрерывным циклом работы

В результате использования двух блоков ОЗУ достигается одновременный прием данных и их обработка систолической структурой. Организацию непрерывного цикла работы системы при этом можно описать следующим образом.

Внешние данные записываются в первый блок ОЗУ. Как только этот блок полностью заполнится, записанные данные начинают считываться для вычислений. А в это время продолжающие поступать внешние данные записываются во второй блок ОЗУ. Далее, после того, как один из блоков полностью просмотрен, а другой заполнен новыми данными, блоки меняются ролями.

Опредение производительности систолического процессора

Производительность систолического процессора будем определять следующим образом.

Пусть α - время такта работы конвейера. Тогда общее время вычислений

$$t_o = (N+n)\,\alpha ,$$

где N - характеризует число тактов на загрузку конвейера, а n - число тактов вычислений.

Общее число операций сложения и умножения, выполняемых в конвейере

$$V = (2N-1)n .$$

Тогда производительность процессора в общем режиме

$$\gamma = \frac{V}{t_o} = \frac{(2N-1)\,n}{(N+n)\,\alpha} ,$$

а в режиме насыщения

$$\gamma_n = \frac{2N-1}{\alpha} .$$

Приведенные выше выражения позволяют оценивать производительность систолического процессора в различных режимах.

Обеспечение отказоустойчивости и тестопригодности

Для обеспечения отказоустойчивости необходимо ввести в систолический массив дополнительные элементы, позволяющие обходить неисправные процессоры. При этом средства отказоустойчивости должны быть ориентированы как на нейтрализацию производственных отказов с целью увеличения съема кристаллов с пластины, так и эксплуатационных отказов с целью улучшения показателей надежности схемы. Для обеспечения отказоустойчивости в такой схеме целесообразно использовать скользящее резервирование и алгоритм реконфигурации "сдвиг в линейке", так как они позволяют обеспечить большую эффективность резервирования при малых аппаратных издержках. Рассмотрим обеспечение отказоустойчивости в такой схеме на этапе производства. В этом случае в схему вводятся дополнительные логические ключи с тремя состояниями (рис.34) и ПЗУ дефектов, которое представляет в общем случае набор плавких перемычек (рис.35). На этапе производства СБИС в него заносится информация (путем пережигания соответствующих плавких перемычек) о дефектных процессорных элементах. В результате этого неисправные процессорные элементы будут обходиться в цепочке. Размерность ПЗУ дефектов соответствует общему количеству процессорных элементов в систолической линейке.

Для обеспечения тестопригодности и отказоустойчивости систолического массива на этапе эксплуатации в схему дополнительно вводится цепочка триггеров управления ключами (рис.36), которая представляет собой сдвигающий регистр. Число триггеров при этом равно количеству ПЭ. Здесь предполагается наличие внешнего тестирующего устройства, которое на этапе тестирования путем подачи последовательного кода на вход сдвигающего регистра, устанавливает триггеры в соответствии с работоспособностью принадлежащих им процессорных элементов. В результате этого каждый триггер схемы может подключить или отключить к общей цепочке схемы соответствующий процессорный элемент. При отказе 1 -го процессорного элемента ПЭ(1) происходит такое его исключение из линейки, при котором функции выполняемые ПЭ(1)...ПЭ(N)

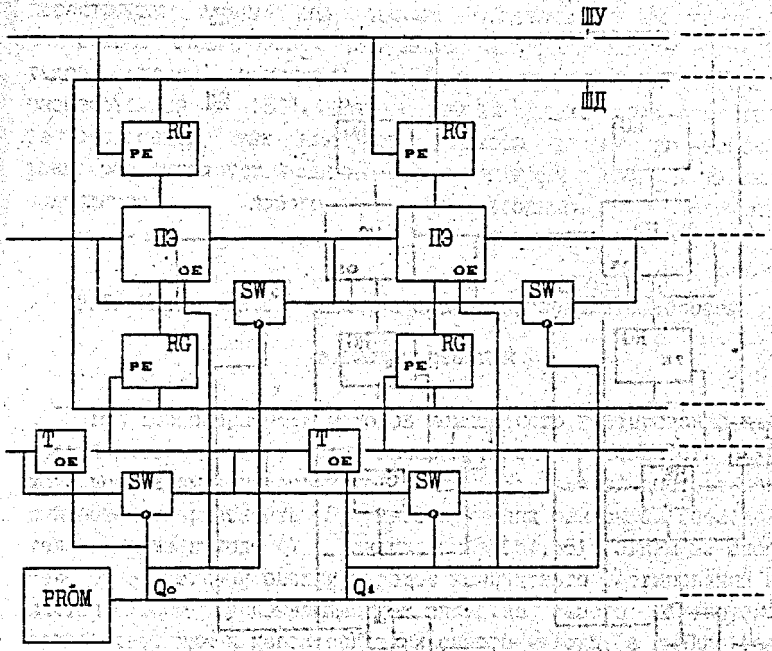
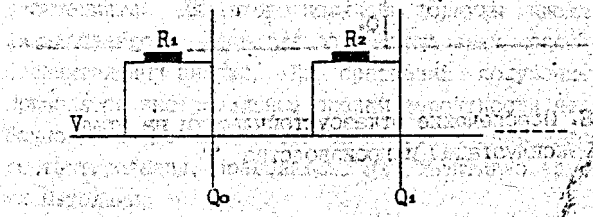


Рис. 34. Обеспечение отказоустойчивости на этапе производства



R_1, R_2 - программируемые связи
V - единичный уровень сигнала
Q_0, Q_1 - выходные данные

Рис. 35. Организация фрагмента ПЗУ дефектов

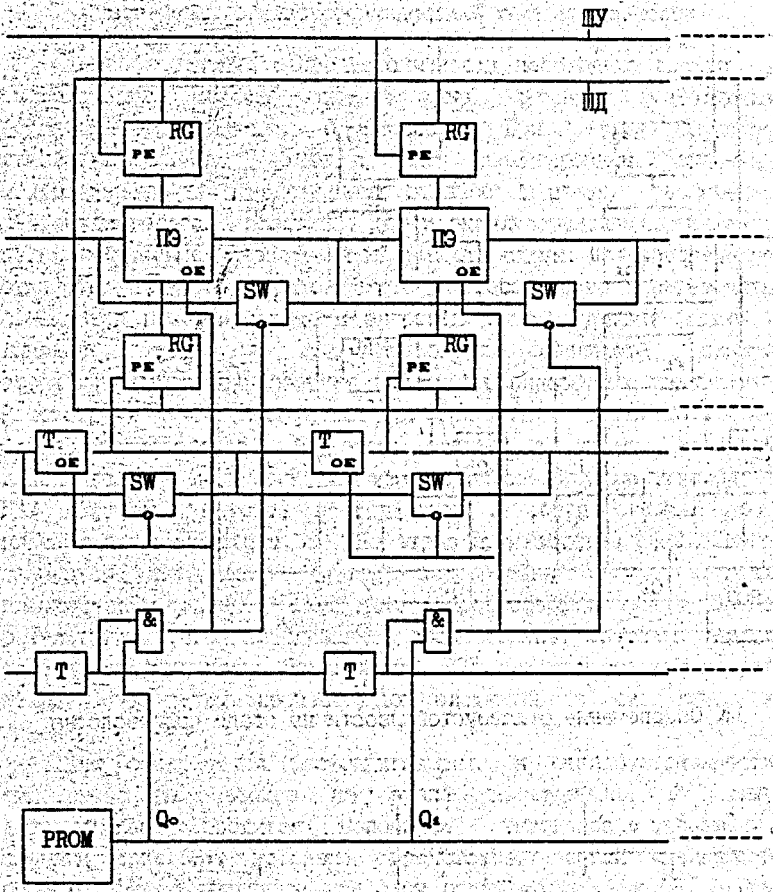


Рис. 36. Обеспечение отказоустойчивости на этапе эксплуатации и производства

передаются ПЭ(1+1) ... ПЭ(N+1) соответственно (т.е. происходит сдвиг вправо на один элемент). Следует также отметить, что в такой схеме легко реализуются два подхода к структурной перестройке: 1) обеспечение заранее меньшего уровня производительности, когда в исходном состоянии все процессорные элементы рабочие; 2) обеспечение структурной

перестройки схемы для замены отказавших элементов на резервные. Общий объем резервных процессорных элементов в такой схеме выбирается в соответствии с методикой, разработанной в [53]. Разработанная схема является также тестопригодной, так как при помощи управления цепочкой триггеров отказоустойчивости можно транспортировать на выход содержимое любого процессорного элемента.

5.6. Достоверность методов сжатия последовательностей тестовых реакций

Под достоверностью методов компактного тестирования понимается достоверность сжатия бинарной последовательности на выходе исследуемых цифровых устройств (ЦУ). Мерой достоверности считается вероятность $P_н$ необнаружения ошибочной последовательности на выходе ЦУ. Как изгестно [54,55], одним из способов сравнительной оценки методов компактного тестирования является способ, учитывающий как свойства самого тестируемого устройства, так и характеристики самого метода, а именно распределение вероятностей необнаружения ошибочной последовательности задачного веса. В настоящей работе рассматривается распределение вероятностей необнаружения ошибочной последовательности фиксированного веса кодом Хэмминга и некоторыми его модификациями. Для этого получены формулы числа кодовых слов фиксированного веса для рассматриваемых кодов и произведен их сравнительный анализ. На основании полученных результатов, приводятся экстремальные оценки вероятности необнаружения ошибочной последовательности рассматриваемыми методами компактного тестирования, позволяющие их объективно сравнивать с другими методами.

Количество кодовых слов фиксированного веса бинарного кода Хэмминга и его модификаций

Будем рассматривать бинарный циклический код C длины $n(C)=2^m-1$, порождающий многочлен $g(x)$ которого, является примитивным многочленом степени m над полем GF(2), а также следующие его модификации:

A - суженный код длины $n(A)=2^m-1$, т.е. код с порождающим многочленом $(x+1)*g(x)$;

E - расширенный код длины $n(E)=2^m$, полученный из исходного кода C путем добавления общей проверки на четность. Как известно [56], код C эквивалентен бинарному коду Хэмминга длины 2^m-1, распределение весов которого дает следующая теорема.

Теорема 1. Число C^k кодовых слов веса k кода Хэмминга длины $n=2^m-1$ определяется выражением

$$C^k=\frac{C^k_n+(-1)^{]k/2[}*C^{[k/2]}_{(n-1)/2}*n}{n+1},$$

где $0<k<n$, $[t]$ - наибольшее целое число, не превосходящее действительное число t, $]t[$ - наименьшее целое число, большее действительного числа t.

Доказательство. Известно [57], что весовая функция C(x) бинарного кода Хэмминга имеет вид

$$C(x)=[(1+x)^n+n*(1-x)^{a+1}*(1+x)^a]/(n+1),$$

где $a=(n-1)/2$. Преобразуя последнее выражение, получим

$$C(x)=[(1+x)^n+n*(1-x^2)^a*(1-x)]/(n+1).$$

Разлагая $(1+x)^n$ и $(1-x^2)^a$ в биномиальные ряды, имеем

$$C(x)=(\sum_{k=0}^{n}C^k_n*x^k+n*\{\sum_{k=0}^{a}(-1)^k*C^k_a*x^{2k}+\sum_{k=0}^{a}(-1)^{k+1}*C^k_a*x^{2k+1}\})/(n+1).$$

Применяя приведенные выше обозначения [ ] и] [, произведем следующие преобразования весовой функции

$$C(x)=(\sum_{k=0}^{n}C^k_n*x^k+n*\{\sum_{k=0}^{a}(-1)^{]2k/2[}*C^{[2k/2]}_a*x^{2k}+$$

$$+\sum_{k=0}^{a}(-1)^{](2k+1)/2[}*C^{[(2k+1)/2]}_a*x^{2k+1}\})/(n+1).$$

Объединяя ряды, находящиеся в фигурных скобках, получим

$$C(x)=\sum_{k=0}^{n}\{(C^k_n+(-1)^{]k/2[}*C^{[k/2]}_a*n)/(n+1)\}*x^k$$

Учитывая, что весовая функция определяется как $\sum_{k=0}^{n}C^k*x^k$, и приравнивая коэффициенты при одинаковых степенях x, получаем требуемое утверждение. Теорема доказана.

Используя теорему, получим формулы числа кодовых слов E^k, A^k фиксированного веса k кодов E и A, т.е. расширенного

и суженного кодов Хэмминга.

Следствие 1. Число E^k кодовых слов веса k расширенного кода Хэмминга длины $N=2^m$ определяется выражением

$$E^k = \frac{C_N^k + (-1)^{k/2} * C_{N/2}^{k/2} * (N-1)}{N}, \text{ если k четно и}$$

$E^k = 0$, если k нечетно, где $0<k<N$.

Доказательство. Как известно [58], расширенный код Хэмминга инвариантен относительно транзитивной группы подстановок. Тогда по теореме Прейндж [5], величины C^k и E^k связаны соотношением:

$$E^{2j} = \frac{C^{2j-1} * N}{2j}, \text{ где } 0<j<2^{m-1}.$$

Используя теорему 1 и некоторые свойства биномиальных коэффициентов, получим

$$E^{2j} = \frac{C_{N-1}^{2j-1} + (-1)^j * C_{N/2-1}^{j-1} * (N-1)}{N} * \frac{N}{2j} =$$

$$= \frac{C_N^{2j} + (-1)^j * C_{N/2}^j * (N-1)}{N}, \text{ что и доказывает первую}$$

часть следствия. Вторая часть - очевидна, ввиду отсутствия у кода E кодовых слов нечетного веса.

Следствие 2. Число кодовых слов A^k фиксированного веса k суженного кода Хэмминга длины $n=2^m-1$ определяется выражением

$$A^k = \frac{C_{n-1}^k + (-1)^{k/2} * C_{n/2-1}^{k/2} * (n-1)}{n}, \text{ если k четно и}$$

$A^k = 0$, если k нечетно, где $0<k<n-1$.

Доказательство. Как известно [59], классическое понятие термина "сужение" предполагает удаление из исходного кода

некоторого его подмножества. В нашем случае, сужение кода C - есть удаление из него подмножества последовательностей длины n, имеющих нечетный вес. Тогда между величинами A^k и C^k существует следующее соотношение для четного k: $A^k=C^k$. Используя теперь теорему 1, получаем требуемое утверждение для кодовых слов четного веса. Как и в следствии 1, доказательство второй части утверждения очевидно, ввиду приводимых там рассуждений.

Пример1. Пусть m=3,4,5. Тогда длины рассматриваемых кодов равны соответственно L(C)=7,15,31; L(A)=7,15,31; L(E)=8,16,32. В табл. 1 приведены данные о количестве кодовых слов фиксированного веса k кодов C, A, E.

Вероятность необнаружения ошибочной последовательности фиксированного веса методами компактного тестирования с характеристиками кода Хэмминга и его модификаций

Вероятность P^k необнаружения ошибочной последовательности веса k определяется как отношение количества S^k необнаруживаемых определенным методом компактного тестирования ошибочных последовательностей веса k, к общему числу возможных последовательностей длины L того же веса:

$$P^k = \frac{S^k}{C_L^k}. \qquad (17)$$

Рассматриваемые нами методы компактного тестирования C', A', E' с характеристиками кодов C, A, E построены на основе регистров сдвига с обратными связями, а именно :

C' - на основе порождающего примитивного полинома g(x) степени m над полем GF(2) (так называемый сигнатурный анализатор [6]);

A' - на основе полинома (x+1)*g(x);

E' - на основе полинома g(x) с добавлением общей проверки на чётность (т.е. с операцией суммирования по модулю 2 выходной последовательности). Соответственно, разрядность сигнатур равна m для C' и (m+1) - для A', E', а длина равна длинам

Таблица 1

Количество кодовых слов фиксированного веса k кода Хэмминга и его модификаций

m=3

k	C^k	A^k	E^k
0	1	1	1
1	0	0	0
2	0	0	0
3	7	0	0
4	7	7	14
5	0	0	0
6	0	0	0
7	1	0	0
8	-	-	1

m=4

0	1	1	1
1	0	0	0
2	0	0	0
3	35	0	0
4	105	105	140
5	168	0	0
6	280	280	448
7	435	0	0
8	435	435	870
9	280	0	0
10	168	168	448
11	105	0	0
12	35	35	140
13	0	0	0
14	0	0	0
15	1	0	0
16	-	-	1

Продолжение табл. 1

m=5

0	1	1	1
1	0	0	0
2	0	0	0
3	155	0	0
4	1085	1085	1240
5	5208	0	0
6	22568	22568	27776
7	82615	0	0
8	247845	247845	330460
9	628680	0	0
10	1383096	1383096	2011776
11	2648919	0	0
12	4414865	4414865	7063784
13	6440560	0	0
14	8280720	8280720	14721280
15	9398115	0	0
16	9398115	9398115	18796230
17	8280720	0	0
18	6440560	6440560	14721280
19	4414865	0	0
20	2648919	2648919	7063784
21	1383096	0	0
22	628680	628680	2011776
23	247845	0	0
24	82615	82615	330460
25	22568	0	0
26	5208	5208	27776
27	1085	0	0
28	155	155	1240
29	0	0	0
30	0	0	0
31	1	0	0
32	-	-	1

L(C), L(A), L(E) соответствующих кодов.

Теорема 2. Вероятности $P^k(C)$, $P^k(A)$, $P^k(E)$ необнаружения ошибочной последовательности фиксированного веса k методами компактного тестирования C', A', E' определяются следующим образом:

$$P^k(C)=(1+(-1)^{]k/2[}*\frac{C^{[k/2]}_{(n-1)/2}}{C^k_n}*n)/(n+1),$$

для любого k от 0 до 2^m-1;

$$P^k(A)=(1+(-1)^{k/2}*\frac{C^{k/2}_{(n-1)/2}}{C^k_n}*n)/(n+1),$$

если k четно, и $P^k(A)=0$, если k нечетно;

$$P^k(E)=(1+(-1)^{k/2}*\frac{C^{k/2}_{N/2}}{C^k_N}*(N-1))/N,$$

если k чётно, и $P^k(E)=0$, если k нечётно, где $n=2^m-1$, $N=2^m$.

Доказательство. Заметим, что количество необнаруживаемых ошибочных последовательностей фиксированного веса рассматриваемыми методами есть число кодовых слов того же веса соответствующих им кодов. Поэтому, для доказательства данной теоремы, воспользуемся полученными ранее формулами C^k, A^k, E^k числа кодовых слов веса k кодов C, A, E. Подставляя их в формулу (17), и произведя несложные преобразования, получим требуемое утверждение. Теорема доказана.

Замечание. Нетрудно проверить, что вероятности необнаружения ошибочной последовательности четного веса k одинаковы для всех рассматриваемых кодов, где $0<k<2^m-2$. Кроме того, для сигнатурного анализатора, т. е. метода с характеристиками кода Хэмминга выполняется соотношение

$P^{2j}(C)=P^{2j-1}(C)$, где $0<j<2^{m-1}-1$.

Следующий пример наглядно демонстрирует указанные выше свойства.

Пример 2. Пусть m=3,4,5. Тогда длины рассматриваемых кодов равны соответственно n(C)=7,15,31; n(A)=7,15,31; n(E)=8,16,32. В табл. 2 приведены данные о вероятностях $P^k(C)$, $P^k(A)$, $P^k(E)$ необнаружения ошибочной последовательности фиксированного веса k методами с характеристиками кода Хэмминга и его модификаций.

Экстремальные оценки вероятности необнаружения ошибочной последовательности методами компактного тестирования с характеристиками кода Хэмминга и его модификаций

Рассмотрим распределение вероятностей $P_н^k$ необнаружения ошибочной последовательности, содержащей ошибки кратности k, определяемое следующим соотношением:

$$P_н^k=P_c^k*P^k,$$

где P_c^k - вероятность возникновения ошибки кратности k, а P^k - вероятность необнаружения возникшей ошибки этой же кратности. Тогда вероятность $P_н$ необнаружения ошибочной последовательности

$$P_н=\sum_{k=1}^{L} P_н^k=\sum_{k=1}^{L} P_c^k*P^k. \qquad (18)$$

Она используется в качестве меры эффективности метода компактного тестирования [59]. Так как получение точного распределения вероятностей P_c^k, определяемого особенностями проверяемой схемы, множеством возможных ее неисправностей и тестовой последовательностью, как правило невозможно, определим предельные значения необнаружения ошибочной последовательности $P_н$ методами с характеристиками кодов C, A, E. Для рассматриваемых кодов существуют такие веса, для которых вероятность необнаружения ошибок соответствующими методами равна нулю. Так

$P^k(C)=0$, при $k=1,2,2^m-3,2^m-2$;

$P^k(A)=0$, при $k=2,2^m-2$, а также всех нечетных k;

Таблица 2

Вероятность необнаружения ошибочной последовательности фиксированного веса методами компактного тестирования с характеристиками кода Хэмминга и его модификаций

m=3

k	$P^k(C)$	$P^k(A)$	$P^k(E)$
0	1.00000000	1.00000000	1.00000000
1	0.00000000	0.00000000	0.00000000
2	0.00000000	0.00000000	0.00000000
3	0.20000000	0.00000000	0.00000000
4	0.20000000	0.20000000	0.20000000
5	0.00000000	0.00000000	0.00000000
6	0.00000000	0.00000000	0.00000000
7	1.00000000	0.00000000	0.00000000
8	-	-	1.00000000

m=4

0	1.00000000	1.00000000	1.00000000
1	0.00000000	0.00000000	0.00000000
2	0.00000000	0.00000000	0.00000000
3	0.07692308	0.00000000	0.00000000
4	0.07692308	0.07692308	0.07692308
5	0.05594406	0.00000000	0.00000000
6	0.05594406	0.05594406	0.05594406
7	0.06759907	0.00000000	0.00000000
8	0.06759907	0.06759907	0.06759907
9	0.05594406	0.00000000	0.00000000
10	0.05594406	0.05594406	0.05594406
11	0.07692308	0.00000000	0.00000000
12	0.07692308	0.07692308	0.07692308
13	0.00000000	0.00000000	0.00000000
14	0.00000000	0.00000000	0.00000000
15	1.00000000	0.00000000	0.00000000
16	-	-	1.00000000

m=5

0	1.00000000	1.00000000	1.00000000
1	0.00000000	0.00000000	0.00000000
2	0.00000000	0.00000000	0.00000000
3	0.03448276	0.00000000	0.00000000
4	0.03448276	0.03448276	0.03448276
5	0.03065134	0.00000000	0.00000000
6	0.03065134	0.03065134	0.03065134
7	0.03141762	0.00000000	0.00000000
8	0.03141762	0.03141762	0.03141762
9	0.03118441	0.00000000	0.00000000
10	0.03118441	0.03118441	0.03118441
11	0.03128436	0.00000000	0.00000000
12	0.03128436	0.03128436	0.03128436
13	0.03122649	0.00000000	0.00000000
14	0.03122649	0.03122649	0.03122649
15	0.03127074	0.00000000	0.00000000
16	0.03127074	0.03127074	0.03127074
17	0.03122649	0.00000000	0.00000000
18	0.03122649	0.03122649	0.03122649
19	0.03128436	0.00000000	0.00000000
20	0.03128436	0.03128436	0.03128436
21	0.03118441	0.00000000	0.00000000
22	0.03118441	0.03118441	0.03118441
23	0.03141762	0.00000000	0.00000000
24	0.03141762	0.03141762	0.03141762
25	0.03065134	0.00000000	0.00000000
26	0.03065134	0.03065134	0.03065134
27	0.03448276	0.00000000	0.00000000
28	0.03448276	0.03448276	0.03448276
29	0.00000000	0.00000000	0.00000000
30	0.00000000	0.00000000	0.00000000
31	1.00000000	0.00000000	0.00000000
32	-	-	1.00000000

$P^k(E)=0$, при $k=2,2^m-2$, а также всех нечетных k;
Поэтому минимальное значение вероятности P^k, определяемого согласно (17), равно нулю для всех рассматриваемых методов.

Перейдем к определению максимальной $maxP^k$, а также минимальной ненулевой $minP^k$, оценки вероятности P^k методов с характеристиками кодов С, А, Е.

Теорема 3. Максимальное $maxP^k$ и минимальное $minP^k$ значение вероятности необнаружения ошибочной последовательности фиксированного веса методами компактного тестирования с характеристиками кода Хэмминга и его модификаций определяется следующим образом:

$$maxP^k=1/(2^m-3),\quad minP^k=(2^m-8)/((2^m-3)*(2^m-5)),$$

где максимальное значение определено для тех весов k, для которых $P^k<1$, а минимальное - для тех весов k, для которых $P^k>0$.

Доказательство. Учитывая замечание предыдущего раздела, доказательство достаточно провести лишь для одного из рассматриваемых методов. Докажем утверждение теоремы для метода с характеристиками кода Е.

Вначале отметим, что для кода Е, как впрочем и для С, А, верно следующее соотношение $P^k = P^{L-k}$, где L-длина кода. Поэтому достаточно рассмотреть последовательность (P^{2i}) лишь для тех i, для которых $2<i<2^{m-2}$. Тогда из того, что

$$\frac{C^i_{N/2}}{C^{2i}_N}=\frac{C^{i-1}_{N/2}}{C^{2i-2}_N}*\frac{2i-1}{N-2i+1},\quad \text{где } N=2^m,$$

следует, что при увеличении i от 2 до 2^{m-2} отношение $C^i_{N/2}/C^{2i}_N$ строго убывает. Кроме того, из выражения

$$P^{2i}=(1+(-1)^i*\frac{C^i_{N/2}}{C^{2i}_N}*(N-1))/N,$$

следует, что $P^{2i}>1/N$ при четном i и $P^{2i}<1/N$ при нечетном i. Таким образом, последовательность (P^{2i}) монотонно убывает и ограничена снизу 1/N для четных i, и монотонно возрастает и

ограничена сверху 1/N для нечетных i. Поэтому, максимальное $\max P^k$ и минимальное $\min P^k$ значение достигается, соответственно, при наименьшем четном и наименьшем нечетном значении i, т.е. при i=2 и i=3, соответственно. Тогда

$$\max P^k = P^4 = (1+(N-1)*C^2_{N/2}/C^4_N)/N = 1/(N-3) = 1/(2^m-3)$$

$$\min P^k = P^6 = (1-(N-1)*C^3_{N/2}/C^6_N)/N = (N-8)/\{(N-3)*(N-5)\} =$$

$$= (2^m-8)/\{(2^m-3)*(2^m-5)\}$$

Теорема доказана.

Теорема дает предельные значения вероятности необнаружения ошибочной последовательности для весов, для которых указанная вероятность отлична от нуля и единицы. Более того, несложно определить, при каких весах достигаются эти предельные значения. В табл. 3 приведены веса, для которых достижимы предельные значения при указанных выше условиях.

Таблица 3

Предельные значения вероятности необнаружения ошибочной последовательности фиксированного веса

		C	A	E
$\min P^k$	значение	$\frac{2^m-8}{(2^m-3)*(2^m-5)}$	$\frac{2^m-8}{(2^m-3)*(2^m-5)}$	$\frac{2^m-8}{(2^m-3)*(2^m-5)}$
	вес	$5,6,2^m-6,2^m-5$	$6,2^m-6$	$6,2^m-6$
$\max P^k$	значение	$\frac{1}{2^m-3}$	$\frac{1}{2^m-3}$	$\frac{1}{2^m-3}$
	вес	$3,4,2^m-4,2^m-5$	$4,2^m-4$	$4,2^m-4$

Известно [54], что максимальное $\max P_н$ значение вероятности $P_н$ необнаружения ошибочной последовательности конкретным ме-

тодом компактного тестирования равно максимальной величине распределения вероятностей P^k. Аналогичное утверждение верно и для минимальной $minP_н$ ненулевой вероятности $P_н$. Используя результаты теоремы 3, имеем, что

$$maxP_н(C)=maxP_н(A)=maxP_н(E)=\frac{1}{2^m-3} \text{ и}$$

$$minP_н(C)=minP_н(A)=minP_н(E)=\frac{2^m-8}{(2^m-3)*(2^m-5)}.$$

ЗАКЛЮЧЕНИЕ

В материале по математическому обеспечению ЭВМ описаны алгоритмы и процедуры, предназначенные для логического проектирования цифровых схем в базисе элементов библиотек интегральных технологий.

На основе анализа систем поведенческого моделирования сформулированы требования к языковым средствам и системам их моделирования; описан типовой состав изобразительных средств и рассмотрена организация поведенческого моделирования, способ преобразования исходного описания проекта в функционально-адекватный текст на языке высокого уровня, приведен алгоритм событийного моделирования функционально-адекватного описания блока.

Предлагаемые алгоритмы синтеза дают возможность выполнить синтез схем логического управления как на жесткой, так и на микропрограммной логике, а также синтез произвольных неоднородных отказоустойчивых структур для нейтрализации как эксплуатационных, так и производственных отказов. Рассмотрены примеры синтеза устройства управления умножителя, систолического процессора для вычисления корреляционной функции сигнала и систолического вычислителя для двумерных преобразований сигналов.

Выполнена экспериментальная проверка указанных подходов и алгоритмов, реализованных в виде программного обеспечения на языках Паскаль и Си. Предложены структуры подсистем САПР СБИС, которые являются частью системы высокоуровнего синтеза. Эти подсистемы рассчитаны на использование ПЭВМ класса PC AT в среде MS DOS и обеспечивают восходящий и нисходящий подходы к проектированию.

ЛИТЕРАТУРА

I. "IEEE, VHDL Language Refernce Manual" IEEE std 1076-1987, Mar. 1988.

2. IEEE Seandard 1076. VHDL Tutorial" CAD Language Systems Inc., March 1989.

3. Степанец В.Я., Гурский Л.И. Автоматизированный синтез СБИС. - Минск: Наука и техника, 1991.

4. Розов М.М. Транслятор с языка VHDL в САПР цифровых СБИС // Промышленные САПР в области электроники и вычислительной техники : Тез. докл. II научного совещания-семинара (10-16 июня 1991, г. Брест). - Минск: Ин-т техн. кибернетики АН БССР, 1991. - С. 25-26.

5. A VHDL based design environment for VLSI Circuits / Vyas M.C., Reddy G.N.//IEEE SEAUTHEASTCON'89 Conf and Exib. -Energy and Inf. Technol. SouthEast, New York.- 1989.-P. 301-405.

6. A compiled-code VHDL simulator/ Reddy G.N., Vyas M.C. //Proc. 31st Midwest Symp. Cuircuirts and Syst.,1988.- P. 152-155.

7. Typed circal: A high level framework for hardware verification/Milne George J.//Fusion Hardware Des.and Verificat :Proc IFIP WG 10.2 Work Conf., Glasgow 1988.-P.127-138.

8. Высокоуровневый синтез цифровых систем // М.С.Макфарланд, Э.С.Паркер, Р.Камposano // ТИИЭР.-1990.-Т.78.-N 2. - С.84-103.

9. A space logistics simulation implementation in ADA/ Borrego Jesus//Winter. Simul.Conf.Proc., San Diego, 1988.- P. 761-764.

10. Проектирование интегральных схем: Направления и проблемы /Р.К.Кейвин, Дж.Л.Хилберт //ТИИЭР.-1990.-Т.78.- N 2.- С.213-235.

II. VHDL design and analysis system targets concurrent engineering/Meyer Ernest//Comput Des.-1990.-29, N.11.-P.110.

12. Verification of VHDL designs using VAL/ Augustin Larry M.//25th ACM/IEEE Des. Autom.Conf., Anaheim,88,Proc.- P. 48-53.

13. Параметрический синтез СБИС-систем /Дж.Аллен //ТИИЭР.-1990.-Т.78.- N 2. - С. 124-145.

I4. La simulation totale, credo de Viewlogic/P.R.// Mesures.-1989.-N609.-P.102.

15. UTILE system:a unified environment from simulation to test/ Becu Jean Luc//Proc. 1st Eur. Test Conf., Paris,1989.-P.369-376.

16. Synthesis from register-transfer level VHDL/Straus Jim. //Compcon Spring'89 :34th IEEE Comput. Soc. Int.Conf., 1989.- P. 473-477.

17. The IBM VHDL design system/Saunders. Larry F//24th ACM/IEEE Des. Autom.Conf.,Miami Beach,87,Proc, P.- 484-490.

18. Verification of VHDL designs using VAL/ Augustin Larry M.//25th ACM/IEEE Des. Autom.Conf.,Anaheim ,88,Proc, P. 48-53.

19. VHDL: a call for standarts/Coelho David R. //25th ACM/IEEE Des. Autom.Conf.,Anaheim,88,Proc, P. 40-47.

20. Experience with the VHTL environment/Loughzail M. // 25th ACM/IEEE Des. Autom.Conf.,Anaheim 88,Proc, P. 28-33.

21. VLSI system models/De Vivek K.//Proc IEEE Custom Integreted Circuits Conf.,Rochester,1988.-P.22.7.1-22.7.4.

22. A hardware design laguage for logic simulation in synthesis in VLSI/Roesner Volfgang//VLSI and Compu.:1st Int. Conf.Comput. Techn1., Syst. and Appl.,Hamburg,1987.-P. 311-314.

23. Армстронг Дж. Р. Моделирование цифровых систем на языке VHDL: Пер. с англ. - М. : Мир, 1992. - 175 с.

24. Скляров В.А., Новиков В.С., Ярмолик В.Н. Автоматизация проектирования ЭВМ.-Минск: Высшая школа, 1990.

25. Иоффе М.И. Диагностирование логических схем. Алгоритмы моделирования и автоматического синтеза теста.-М.: Наука,1989.- 158 с.

26. Майоров С.А., Новиков Г.И. Принципы организации цифровых машин.-М.: Машиностроение, 1974.

27. Ньютон А.Р., Санджованни-Винчентелли А.Л. Системы автоматизированного проектирования специализированных ИС // ТИИЭР.- Т. 75.-N 6, июль 1987. - С. 30-43.

28. Киносита К., Асада К., Карацу О. Логическое проекти-

рование СБИС : Пер. с япон. - М. : Мир, 1988. - 309 с.

29. Закревский А.Д. Логический синтез каскадных схем.- М.: Наука, 1981. - 416 с.

30. Бибило П.Н. Автоматизация логического проектирования цифровых устройств на программируемых матричных БИС. - Минск : БелНИИНТИ, 1989. - 56 с.

31. Каган Б.М. Электронные вычислительные машины и системы.-М.: Энергия, 1979.

32. Баранов С.И.. Синтез микропрограммных автоматов.-Л.: Энергия, 1979.

33. Баранов С.И.. Автоматизация проектирования цифровых устройств.-Л.: Судостроение, 1979.

34. Нозик В.М. Банк данных для САПР СБИС с гибкой архитектурой // Промышленные САПР в области электроники и вычислительной техники : Тез. докл. II научного совещания-семинара (10-16 июня 1991, г. Брест). - Минск: Ин-т техн. кибернетики АН БССР, 1991. - С. 14-15.

35. Интеллектуальные системы автоматизированного проектирования БИС и СБИС // Под ред. В.А. Мищенко.- М.: Радио и связь, 1988.-272 с.

36. Система автоматизированного логического проектирования цифровых СБИС / Мищенко В.А., Дудкин А.А., Дорошка Т.М., и др. / Тез. докл. международ. конф. "САПР-91:Новые информационные технологии в проектировании".- Минск: БГУ им В.И. Ленина, 1991.-С.18.

37. Мищенко В.А., Дудкин А.А., Бобрович С.В. Параметрически управляемый итерационный алгоритм проектирования в САПР СБИС / Тез. докл. международ. конф. "САПР-93: Новые информационные технологии в науке, образовании и бизнесе".- М., 1993.-С.43-44.

38. Дудкин А.А. Логический синтез схем в САПР СБИС верхнего уровня / Тез. докл. международ. конф. "САПР-93: Новые информационные технологии в науке, образовании и бизнесе".- М.,1993.-С.44-45.

39. Дудкин А.А. Синтез комбинационных схем из ПЛМ в САПРСБИС // Проблемы построения САПР СБИС. - Минск: Ин-т техн. кибернетики АН БССР, 1990.- С. 117-123.

40. Погарцев А.Г. Новые алгоритмы совместной минимизации булевых функций. - Автоматика и вычислительная техника.- 1980.- N 1. - С. 34 - 41.

41. Садыхов Р.Х., Чеголин П.М., Шмерко В.Г. Методы и средства обработки сигналов в дискретных базисах.-Минск: Наука и техника,1987.

42. СБИС для распознавания образов и обработки изображений / Под ред. К.Фу.-М.: Мир, 1988.

43. Мачнев А.Г., Садыхов Р.Х., Золотой С.А. Алгоритм построения специализированных систолических вычислителей двумерных преобразований // Технология построения САПР СБИС / Под ред.В.А.Мишенко.-Минск: Ин-т техн. кибернетики АН БССР, 1989.

44. Систолические вычислители / Под ред. Я.О. Дуброва.- Львов: ИППММ АН УССР,1989.

45. Многофункциональные систолические структуры / Под ред. Я.О.Дуброва.- Львов: ИППММ АН УССР,1989.

46. Мачнев А.Г., Садыхов Р.Х. Линейные систолические структуры вычислителей двумерных преобразований // Методы и средства цифрового преобразования и обработки сигналов.- Рига,1989.-С.334-336.

47. В.А. Головко. Обеспечение отказоустойчивости в СБИС// Технология авиационного приборо- и агрегатостроения.-Саратов: НИТИ.-Вып. 3, 1989.- С. 69-74.

48. Оптимальные задачи надежности / Под ред. И.А. Ушакова. -М.: Изд-во стандартов, 1968.- 292 с.

49. Головко В.А.. Статистические модели выхода годных для отказоустойчивых схем на кристалле// Микроэлектроника.-Т. 21.- N 1.- 1992.- С. 20-26.

50. Головко В.А.. Некоторые аспекты определения выхода годных для отказоустойчивых схем на кристалле // Микроэлектроника.-Т. 21.- N 5.-1992.-С. 42-49.

51. Мирский Г.Я. Аппаратурное определение характеристик случайных процессов.- М.:Энергия, 1972.-456 с.

52. Кун С. Матричные процессоры на СБИС.- М.: Мир, 1991.-672 с.

53. Головко В.А. Методы синтеза отказоустойчивых структур СБИС. -Дисс. ... канд. техн. наук.- Минск, 1990.- 196 с.

54. Ярмолик В. Н. Применение сигнатурного анализа для

контроля и диагностики сетевых дискретных структур // АиВТ-1985.- N 4.- С. 73-79.

55. Калоша Е. П., Кашнельсон Е. И., Ярмолик В. Н. О достоверности методов компактного тестирования // Автоматика и телемеханика.- 1989.- Т 9.- С. 160-166.

56. Лидл Р., Нидеррайтер Г. Конечные поля / В 2-х т.-Т.2: Пер. с англ.- М.: Мир, 1989.- 822 с.

57. Michelson A. M., Levesque A. H. Error-controlTechniques for Digital Communication.- New York: Wiley, 1985.

58. Берлекэмп Э. Алгебраическая теория кодирования : Пер.с англ.- М.: Мир, 1971.- 477 с.

59. Ярмолик В. Н. Контроль и диагностика цифровых узлов ЭВМ.-Минск: Наука и техника, 1988.- 240 с.

ОГЛАВЛЕНИЕ

Дудкин Александр Арсентьевич, Головко Владимир Адамович, Муравьев Геннадий Леонидович, Мачнев Александр Григорьевич, Качков Илья Владимирович, Махнист Леонид Петрович, Гладышук Виталий Борисович

АЛГОРИТМЫ И ПОДСИСТЕМЫ АВТОМАТИЗИРОВАННОГО ЛОГИЧЕСКОГО ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ СБИС

Отв. за выпуск Н. А. Рудая
Редактор Л. Е. Чеховская

Подписано к печати 20.11.94.
Формат бумаги 60x84 1/16. Офсетная печать.
Уч.-изд. л. 7,5. Усл. печ. л. 6,9. Тираж 100 экз. Зак. 88.

Институт технической кибернетики АН Беларуси.
220012, Минск, ул. Сурганова, 6.

Отпечатано на ротапринте Института технической кибернетики АН Беларуси. 220012, Минск, ул. Сурганова, 6.