1 Описание языка высокого уровня COLAMO

1.1 Синтаксис языка COLAMO

1.1.1 Описание переменных

Переменная в языке COLAMO является идентификатором, обозначающим некоторую область, в которой хранится значение переменной. В языке существует три типа переменных: скалярная переменная, массивы и константы. Скалярные переменные и массивы в отличие от констант могут изменять свои значения в процессе выполнения программы. Все переменные делятся на локальные и глобальные переменные за исключением констант, которые всегда являются глобальными. Каждая переменная в языке характеризуется способом ее хранения и способом доступа к ней.

1.1.2 Скалярные переменные

Синтаксис объявления скалярной переменной в языке COLAMO в расширенной форме Бэкуса-Наура (РБНФ) выглядит следующим образом:

VAR <*СписокПеременных*> : <*ТипДанных*> <*СпособХранения*> ;

Ключевое слово VAR указывает на объявление переменной.

В качестве списка переменных может выступать как одна переменная, так и набор переменных, разделенных между собой запятой.

Директива *ТипДанных* определяет формат (тип) представления значения, соответствующего конкретной переменной.

Для скалярных переменных определены такие типы данных, как целые числа, числа с плавающей запятой, битовые и логические (булевские) типы данных.

Целочисленные типы данных включают в себя шесть целочисленных типов данных обеспечивающих работу как с 32-разрядными типами данных, так и с 64-разрядными.

Список целочисленных типов данных и типов с плавающей запятой языка COLAMO представлен в таблице 1.1.

Все типы данных языка COLAMO соответствуют стандартным типам данных традиционных языков программирования и удовлетворяют требованиям стандарта IEEE754.

В языке COLAMO в отличие от традиционных языков представлены такие типы данных, как Int32 и UInt32. Переменная, объявленная как Int32 или UInt32, занимает в памяти для хранения своего значения две ячейки: в четной ячейке

хранится младшая часть 64-разрядного слова, в нечетной – старшая часть слова. Соответственно и обработка таких переменных также ведется в два этапа.

Таблица 1.1 - Список целочисленных типов данных и типов с плавающей запятой языка COLAMO

Тип данных	Описание
Integer	32-битное целое со знаком
UInteger	32-битное целое без знака
Int32	64-битное целое со знаком, представленное как два слова по 32 разряда
UInt32	64-битное целое без знака, представленное как два слова по 32 разряда
Int64	64-битное целое со знаком
UInt64	64-битное целое без знака
Real	32-битное с плавающей запятой одинарной точности
Real32	64-битное с плавающей запятой, представленное как два слова по 32 разряда
Double	64-битное с плавающей запятой
Int64	64-битное целое со знаком
UInt64	64-битное целое без знака

Наличие таких типов данных в языке обусловлено возможностью организации различных способов хранения данных в памяти реконфигурируемой вычислительной системы.

Логический тип данных в языке COLAMO используется для хранения булевских значений – true и false.

Не допускается неявное преобразование значения логической переменой к целому значению или обратно, т.е. для работы с логическими переменными допускается использование true или false. Использование вместо false значение ноль или ненулевое значение вместо true приведет к синтаксической ошибке.

Для работы с данными на уровне битов в язык COLAMO вводится битовый тип переменных – BIT, применяемый в таких языках описания аппаратуры как VHDL и Verilog.

Возможность использования битовых переменных позволит программисту работать с данными на уровне битов, что расширяет класс решаемых задач на языке COLAMO. В данном случае появляется возможность программирования задач символьной обработки данных.

Использование битовых переменных прямо противоположно использованию логических переменных. Битовая переменная может принимать только одно из двух значений: ноль или единица. Использование для битовой переменной в качестве значения true или false приведет к синтаксической ошибке.

К скалярным переменным в языке относится еще один тип данных – Number.

Данный тип переменной используется для обозначения индексных переменных, используемых в операторе цикла FOR.

Директива *СпособХранения* характеризует способ хранения переменой в памяти. В языке COLAMO существует три типа хранения переменных:

- Mem – мемориальная переменная, соответствующая внутренней ячейке внешней памяти;
- Com – коммутационная переменная, представляющая собой связь между элементами в информационном графе задачи;
- Reg – регистровая переменная, представляет собой регистровую ячейку.

Архитектура современных ПЛИС включает в себя внутреннюю память, которая может быть реализована как с помощью распределенной логики, так и с помощью аппаратно-реализованных блоков памяти BRAMs.

Преимуществами внутренней памяти являются её достаточно высокая производительность, емкость и возможность организации одновременного доступа к памяти нескольких независимых процессов чтения и записи.

Для доступа к внутренней памяти в язык COLAMO вводится новый способ хранения переменных – InterMem.

Более подробное описание способов хранения переменных будет рассмотрено в разделах 1.2.

Рассмотрим пример объявления переменных:

```
Var A, B : Integer Mem;
Var C : Double Reg;
```

В данном случае мы объявили переменные A и B как целочисленные 32-разрядные данные, расположенные во внешней памяти, а переменную C как 64-разрядное слово с плавающей запятой, которой соответствует регистровая ячейка.

Рассмотрим пример объявления переменной типа Number:

```
Var I : Number;
```

Переменная I является индексной переменной и не соответствует ни одному из способов хранения переменных рассмотренных выше.

1.1.3 Массивы

Кроме скалярных переменных в языке COLAMO допускается использование массивов, которые могут быть как одномерные, так и многомерные.

Массивы в языке COLAMO являются только статическими и переопределение их размера не допустимо в отличие от традиционных языков, позволяющих организовывать работу с динамическими массивами.

Синтаксис объявления переменой типа массив в РБНФ выглядит следующим образом:

VAR <*СписокПеременных*> : **Array** <*ТипДанных*> [<*РазмерностьМассива*>] <*СпособХранения*> ;

Ключевым словом при объявлении переменой типа массив является слово Array.

Директива *СпособХранения*, как и для статических переменных, определяет способ хранения элементов массива. По способу хранения элементов массива массивы можно разделить на мемориальные массивы, коммутационные массивы, регистровые массивы и массивы внешней памяти.

Директива *РазмерностьМассива* представляет собой описание размерности массива с указанием способа обработки данных и количество данных, соответствующих данной размерности, или набор данных, разделенных между собой запятой.

Такой способ описания параметров массива связан с тем, что в отличие от однопроцессорных ЭВМ, для которых массивы представляются одномерными, лежащими в памяти машины, массивы в реконфигурируемых вычислительных системах представляются двумерными, где первая координата соответствует номерам каналов элементов распределенной памяти, вторая - ячейкам памяти в канале.

Для обращения к элементам массива используются два основных способа доступа: параллельный доступ (задаваемый типом Vector) и последовательный доступ (задаваемый типом Stream). Аналогичные предложения были реализованы в языке программирования IVTRAN.

Рассмотрим пример объявления переменной типа массив:

Var A : Array Integer [10: Vector, 100 Stream] Mem;

В рассмотренном примере переменная A является массивом, содержащим целочисленные данные, расположенные во внешней памяти. Тип доступа Vector указывает на необходимость размещения элементов массива в десяти каналах памяти, а тип Stream указывает на то, что в каждом канале памяти будет располагаться по 100 элементов массива.

Последовательность описание векторных составляющих массива и потоковых не регламентирована.

1.1.4 Константы

Предваряя переменную ключевым словом Const и инициализируя её, мы объявляем константу. Константа также является переменной, но в отличие от других переменных, как говорилось ранее, она не может менять своего значения.

Константы могут использоваться для определения размерности параметров массивов, а также участвовать в вычислительных выражениях.

Синтаксис объявления константы в РБНФ выглядит следующим образом:

Const <*Идентификатор*> = <*Значение*> ;

Значение константы может быть представлено как целочисленным данным, данным с плавающей запятой, так и в виде шестнадцатеричного значения с указанием типа данного. В этом случае синтаксис объявления константы в РБНФ выглядит следующим образом:

Const <*Идентификатор*> = (<*ТипЗначения*>) <*Значение*> ;

Указание директивы *ТипЗначения* позволяет явно указать тип значения, получаемого константой.

Для определения типа получаемого значения константой используются типы данных, соответствующие скалярным переменным, за исключением 32-разрядных данных с плавающей точкой. Для представления шестнадцатиричного значения к 32-разрядным данным с плавающей запятой необходимо указать тип Float.

Рассмотрим пример объявления констант:

```
Const A = 12;
Const B = (Integer) $10E;
```

В рассмотренном примере константа A получает значение 12, а константа B является целочисленной переменной и принимает значение, равное 1065353216.

В случае объявления константы B в виде Const B = (Float) $3F8; константа B будет являться переменной с плавающей точкой и иметь значение, равное 1.0;

В языке COLAMO допускается вычисление значения константы, выполняемое на этапе трансляции, но при этом все переменные, участвующие в вычислении значения константы, должны быть заранее проинициализированы.

1.1.5 Инициализация переменных

Инициализация переменных в языке COLAMO определяется оператором DEFINE. Инициализация переменной выполняется один раз после описания переменных; в случае, если явной инициализации переменной нет, то такой

переменной присваивается нулевое значение. Вид инициализации переменных зависит от типа инициализируемой переменной.

Синтаксис инициализации переменной типа Number в РБНФ выглядит следующим образом:

Define <*ИмяПеременной*> = <*Значение*> ;.

Если переменная типа Number используется в качестве индексной переменной в операторе цикла, то значение этой переменой будет переопределено в соответствии с параметрами, указанными в операторе For.

Как и для констант, здесь допускается вычисление значения для переменой типа Number, но при этом накладываются те же условия, что и для констант.

Инициализация переменных, организующих работу с внутренней памятью, выглядит иначе, чем инициализация переменных типа Number. Инициализация внутренней памяти осуществляется путем указания файла данных, в котором расположены данные инициализации.

Синтаксис инициализации переменной, для которой определен способ хранения InterMem, выглядит в РБНФ следующим образом:

Define <*ИмяПеременной*> = “ <*ПутьКФайлу*> ”;.

Файл, указанный в директиве, должен иметь расширение TXT (*.txt), у которого на каждой строке в файле должно располагаться одно данное, или DAT (*.dat), в котором все данные расположены подряд и представлены в шестнадцатеричной форме.

В языке COLAMO помимо инициализации переменных существует инициализация конструкции Implicit. Для данной конструкции необходимо с помощью оператора Define указать принадлежность к одному из способов реализации вычислений, описанных в её теле.

Синтаксис инициализации конструкции Implicit может выглядеть в РБНФ следующим образом:

Define <*ИмяКонструкции*> as SubCadr;

или

Define <*ИмяКонструкции*> as LocalProc >> <*ИмяМикропроцессора*> ;.

Более подробная информация о конструкции Implicit будет рассмотрена в подпункте 1.1.3.5.

1.1.6 Описание вычислительных структур

В языке высокого уровня COLAMO присутствуют пять вычислительных структур: кадр (Cadr), подкадр (SubCadr), структура Let, структура LocalProc и структура Implicit.

Каждая структура обладает своими семантическими особенностями и позволяет различными способами реализовывать соответствующие им вычисления.

Переменные, объявленные в таких структурах, являются локальными и доступ к ним извне запрещен.

Все операторы в языке COLAMO делятся на локальные и глобальные операторы. Локальными операторами являются операторы, расположенные в структурах: Cadr, SubCadr и Let. Глобальными операторами являются операторы, расположенные между кадрами или используемые в теле структуры LocaProc.

1.1.6.1. Вычислительная структура CADR. Фундаментальным типом вычислительной структуры в языке COLAMO является конструкция "кадр". Кадром является программно-неделимая компонента, представляющая собой совокупность арифметико-логических команд, выполняемых на различных элементарных процессорах, обладающих распределенной памятью и соединенных между собой в соответствии с информационной структурой алгоритма таким образом, что вычисления производятся с максимально возможными параллелизмом и асинхронностью.

Кадр фактически определяет вычислительную структуру и потоки данных в реконфигурируемой вычислительной системе в конкретный момент времени. При этом все операции в теле кадра выполняются асинхронно с максимальным параллелизмом, а последовательность смены кадров однозначно определяется программистом.

Синтаксис объявления структуры кадр в РБНФ выглядит следующим образом:

Cadr <*ИмяКадра*>;
    <*СписокОператоров*>
EndCadr;,

где Cadr, Endcadr - ключевые слова языка указывающие на организацию структуры кадр;

Идентификатор *ИмяКадра* является идентификационным именем конструкции кадр;

Блок *СписокОператоров* содержит список операторов, выполняемых структурой кадр, рассмотренных в разделе 1.1.2.

Программа на языке COLAMO может содержать несколько кадров, выполняемых последовательно друг за другом в случае отсутствия операторов условного перехода.

Так как все программы на языке COLAMO являются одноструктурными, то существует ограничение по использованию множества кадров в одной задачи. Данное ограничение связано с невозможностью перестройки вычислительной структуры на реконфигурируемой вычислительной системе в процессе выполнения задачи. В данном случае при программировании многокадровых программ пользователь должен использовать в блоке *СписокОператоров* только одну структуру Let. Наличие в блоке *СписокОператоров* нескольких операторов вызова структуры Let в независимости от их имени является синтаксической ошибкой, так как нарушается принцип одноструктурного программирования.

Наличие одной конструкции Cadr в теле параллельной программы позволяет использовать любые операторы и структуры языка COLAMO, но с одним ограничением: конструкция Let с одним и тем же именем в любой момент времени должна выполняться только одна, т.е. параллельная обработка одной и той же конструкции LET запрещена.

1.1.6.2. Вычислительная структура SubCadr. При реализации параллельных вычислений с помощью кадров возможна ситуация, когда функционально законченные подграфы кадров изоморфны. Здесь целесообразно использовать подкадры, являющиеся структурными аналогами подпрограмм. Оператор объявления подкадра в РБНФ выглядит следующим образом:

SubCadr <*ИмяПодкадра*> (In : <*СписокВходов*>; Out : <*СписокВыходов*>) : <*ТипРезультата*> ;
<*ОписаниеПеременных*>
<*СписокОператоров*>
EndSubCadr;

Ключевые слова языка SubCadr и EndSubCadr определяют структуру подкадра. Идентификатор *ИмяПодкадра* определяет идентификационное имя подкадра. Списки *СписокВходов* и *СписокВыходов* определяют список входных и выходных формальных параметров, ограниченных круглыми скобками.

В качестве *ТипРезультата* может использоваться любой тип данных языка COLAMO за исключением типа Array.

Блок *ОписаниеПеременных* содержит объявление локальных переменных, принадлежащих структуре описываемого подкадра, и описание переменных, объявленных в списках *СписокВходов* и *СписокВыходов*. Синтаксис объявления

переменных в структуре SubCadr эквивалентен объявлению переменных, рассмотренных в разделе 1.1.1.

Блок *СписокОператоров* содержит список операторов, выполняемых структурой SubCadr.

При объявлении структуры Subcadr и указании типа возвращаемого значения в блоке *СписокОператоров* необходимо наличие специальной предопределенной переменной Result.

Директива *ТипРезультата* при объявлении структуры SubCadr может отсутствовать, в таком случае структура SubCadr эквивалентна конструкции Procedure (процедура), используемой в традиционных языках программирования.

Блок *СписокОператоров* позволяет работать со всеми операторами, рассмотренными в разделе 1.1.2, за исключением конструкции LET и Cadr.

Обращение к подкадру производится по идентификационному имени, после которого следует список фактических параметров, ограниченных круглыми скобками.

Рассмотрим пример объявления и вызова структуры SubCadr на следующем примере:

```
Var a, b, c : Integer Mem;
SubCadr ExampleSubCadr (In : a1,b1; Out : c1);
Var a1,b1,c1 : Integer Com;
   c1 := a1 + b1;
EndSubCadr;

Cadr Example;
  ExampleSubCadr(a,b,c);
EndCadr;                                                    (1.1)
```

В рассмотренном примере подкадр ExampleSubCadr выполняет сложение значений, получаемых со входов a1 и b1, и возвращает вычисленное значение на выход c1. В кадре Example осуществляется вызов структуры ExampleSubCadr, в которую в качестве параметров передаются мемориальные переменные A, B и C. После выполнения операторов структуры Example мемориальная переменная C будет содержать результат вычислений, выполненных в структуре ExampleSubCadr.

1.1.6.3. Вычислительная структура Let. При реализации параллельных вычислений с помощью кадров или подкадров возможна ситуация, когда необходимо использовать неизменяемую вычислительную структуру, которая в процессе выполнения программы при смене кадров не перегружается. Такая вычислительная структура должна представлять собой функционально законченный

изоморфный граф. Для реализации неизменяемой вычислительной структуры в COLAMO используется конструкция Let, являющаяся структурным аналогом подкадра. Оператор объявления конструкции Let в РБНФ выглядит следующим образом:

Let <*ИмяLet*> (In : <*СписокВходов*>; Out : <*СписокВыходов*>) : <*ТипРезультата*> ;
<*ОписаниеПеременных*>
<*СписокОператоров*>
EndLet;

Ключевые слова языка Let и EndLet определяют область действия структуры Let. Идентификатор *ИмяLet* определяет идентификационное имя структуры Let. Списки *СписокВходов* и *СписокВыходов* определяют список входных и выходных формальных параметров, ограниченных круглыми скобками.

В качестве *ТипРезультата* может использоваться любой тип данных языка COLAMO за исключением типа Array.

Блок *ОписаниеПеременных* содержит объявление локальных переменных, принадлежащих структуре Let, и описание переменных, объявленных в списках *СписокВходов* и *СписокВыходов*. Все локальные переменные являются внутренними и не связаны с именами переменных в кадрах или других структурах языка COLAMO.

Синтаксис объявления переменных в структуре Let соответствует объявлению переменных, рассмотренных в пункте 1.1.1.

Блок *СписокОператоров* содержит список операторов, выполняемых структурой Let.

При объявлении структуры Let и указании типа возвращаемого значения в блоке *СписокОператоров* необходимо наличие специальной предопределенной переменной Result.

Директива *ТипРезультата* при объявлении структуры Let может отсутствовать, в таком случае структура Let эквивалентна конструкции Procedure (процедура), используемой в традиционных языках программирования.

Блок *СписокОператоров* позволяет работать со всеми операторами, рассмотренными в пункте 1.1.2, за исключением конструкции Let и Cadr.

Обращение к подкадру производится по идентификационному имени, после которого следует список фактических параметров, ограниченных круглыми скобками.

Рассмотрим пример объявления и вызова структуры подкадр на следующем примере:

Var a, b, c : Integer Mem;

```
Let ExampleLet (In : a1,b1; Out : c1);
Var a1,b1,c1 : Integer Com;
    c1 := a1 + b1;
EndLet;
Cadr Example;
  ExampleLet (a,b,c);
EndCadr;                                  (1.2)
```

Конструкция Let не может вызываться из структуры SubCadr и является независимой локальной вычислительной структурой. Более подробное описание работы с конструкцией Let рассмотрено в подпункте 1.3.4.3.

1.1.6.4. Вычислительная структура LocaProc. Рассмотренные ранее вычислительные структуры были ориентированы на аппаратную реализацию выполняемых ими вычислений на реконфигурируемых вычислительных системах. Аппаратная реализация вычислений не всегда эффективна, поэтому целесообразно выполнять вычисления процедурно.

Для процедурной реализации вычислений в языке COLAMO вводится конструкция LOCALPROC, которая позволяет выполнять вычисления на указанном специализированном процессоре. Специализированный процессор последовательно выполняет все вычисления, описанные в тексте программы. В общем виде структура LOCALPROC выглядит в РБНФ следующим образом:

LocalProc <*ИмяLocalProc*> (In : <*СписокВходов*>; Out : <*СписокВыходов*>) >> <*ТипПроцессора*>;
   <*ОписаниеПеременных*>
   <*СписокОператоров*>
EndLocalProc;

Ключевые слова языка LocalProc и EndLocalProc определяют область действия структуры LocalProc. Идентификатор *ИмяLocalProc* определяет идентификационное имя структуры LocalProc. Списки *СписокВходов* и *СписокВыходов* определяют список входных и выходных формальных параметров, ограниченных круглыми скобками.

Идентификатор *ТипПроцессора* определяет тип процессора, в команды которого будут оттранслированы операторы структуры LocaProc.

Блок *ОписаниеПеременных* содержит объявление локальных переменных, принадлежащих структуре LocalProc, и описание переменных, объявленных в списках *СписокВходов* и *СписокВыходов*. Все локальные переменные являются

внутренними и не связаны с именами переменных в кадрах или других структурах языка COLAMO.

Синтаксис объявления переменных в структуре LocalProc соответствует объявлению переменных, рассмотренных в пункте 1.1.1.

Блок *СписокОператоров* поддерживает работу только с арифметическими операциями языка COLAMO.

Рассмотрим программу, демонстрирующую использование конструкции LocalProc.

```
Var a, b, c, d, e: array Integer [1000: stream] Mem;
Var i : Number;
LocalProc F (in: a1,b1,c1,d1; out: e1) >> Micro;
Var a1, b1, c1, d1, e1: Integer Com;
    e1:=(a1*b1)-(d1*c1);
EndLocalProc;
Cadr Example32;
  For I := 0 to 999 do
    F(a[i],b[i],c[i],d[i],e[i]);
Endcadr;                                                    (1.3)
```

Представленная программа содержит описание структуры LocalProc, которая имеет четыре входных параметра и один выходной. Для реализации вычислений, соответствующих операторам структуры LocalProc, указан процессор Micro. В данном случае все операторы структуры LocalProc будут оттранслированы в специализированные команды микропроцессора Micro.

1.1.6.5. Вычислительная структура Implicit. В случае изменения аппаратного ресурса реконфигурируемой вычислительной системы или параметров задачи в языке COLAMO необходимы средства перехода от структурной реализации вычислений к процедурной и обратно. Для этого в язык вводится структура с неявным указанием принадлежности к структурной (аппаратная реализация вычислений) или процедурной реализации вычислений Implicit.

Данная конструкция заимствована из языка Фортран IV, в котором оператор Implicit является оператором неявного описания типа и длины константы, переменной, массива и функции.

Конструкция Implicit на языке COLAMO неявно указывает на способ реализации подпрограммы, которая может быть в зависимости от контекста реализована либо процедурно, либо структурно. Программист должен указать, как необходимо интерпретировать вычисления, указанные в конструкции различных Implicit-процедур.

В общем виде структура LOCALPROC выглядит в РБНФ следующим образом:

Implicit <*ИмяImplicit*> (In : <*СписокВходов*>; Out : <*СписокВыходов*>);
<*ОписаниеПеременных*>
<*СписокОператоров*>
EndImplicit;

Ключевые слова языка Implicit и EndImplicit определяют область действия структуры Implicit. Идентификатор *ИмяImplicit* определяет идентификационное имя структуры Implicit. Списки *СписокВходов* и *СписокВыходов* определяют список входных и выходных формальных параметров, ограниченных круглыми скобками.

Блок *ОписаниеПеременных* содержит объявление локальных переменных, принадлежащих структуре Implicit, и описание переменных, объявленных в списках *СписокВходов* и *СписокВыходов*. Все локальные переменные являются внутренними и не связаны с именами переменных в кадрах или других структурах языка COLAMO.

Синтаксис объявления переменных в структуре Implicit соответствует объявлению переменных, рассмотренных в пункте 1.1.1.

Блок *СписокОператоров* поддерживает работу только с арифметическими операциями языка COLAMO.

Рассмотрим программу, демонстрирующую использование конструкции Implicit:

```
Var a, b, c, d, e: array Integer [1000: stream] Mem;
Var i : Number;
Define F is LocalProc >>Micro;
Implicit F (in: a1,b1,c1,d1; out: e1);
Var a1, b1, c1, d1, e1: Integer Com;
    e1:=(a1*b1)-(d1*c1);
EndImplicit;
Cadr ExampleImplicit;
  For i:=0 to 999 do
    F(a[i],b[i],c[i],d[i],e[i]);
Endcadr;                                        (1.4)
```

В рассмотренном примере вычисления, выполняемые в конструкции Implicit, будут выполняться процедурно на специализированном процессоре Micro, который определяется в операторе Define. Если оператор Define будет иметь вид Define F is SubCadr;, то все вычисления в структуре Implicit будут выполняться структурно.

Конструкция Implicit позволяет сделать процесс программирования более гибким и обеспечивает быстрый переход на различные уровни реализации вычислений в зависимости от имеющегося ресурса вычислительной системы.

1.1.7 Операторы языка

Как упоминалось ранее, все операторы в зависимости от места их использования делятся на локальные операторы и глобальные. Как правило, все локальные операторы реализуются структурно, а глобальные в зависимости от контекста программы могут быть реализованы как структурно, так и процедурно.

Все операторы языка COLAMO подразделяются на:

1) операторы присваивания;

2) операторы цикла FOR;

3) условные операторы выбора.

1.1.7.1. Оператор присваивания. Оператор присваивания используются для определения значения переменной, полученного в результате решения некоторого арифметического выражения.

Операторы присваивания в языке COLAMO делятся на арифметические и логические. Арифметические и логические операторы присваивания имеют следующую синтаксическую форму записи:

<ИмяПеременной> := *<Выражение>* ;

где *ИмяПеременной* – имя переменной или элемент массива, а *Выражение* – любое арифметическое или логическое выражение.

Выполнение оператора заключается в вычислении выражения *Выражение* и присваивания его объекту, соответствующего идентификатору *ИмяПеременной*.

В случае если тип полученного значения или его формат не совпадает с типом или форматом представления объекта, над которым выполняется операция присваивания, необходимо выполнить преобразование типа или его формата представления. Функции преобразования типов и формата представления данных рассмотрены в таблице 1.2.

Таблица 1.2 - Функции преобразования типов

Функция	Описание
Int2Flt	Выполняет преобразование числа в формате с фиксированной запятой в формат с плавающей запятой
Flt2Int	Выполняет преобразование числа в формате с плавающей запятой в формат с фиксированной запятой

Int32ToInt64	Выполняет преобразование типа Int32 к типу Int64
Int64ToInt32	Выполняет преобразование типа Int64 к типу Int32
Int2Int32	Выполняет преобразование типа Integer к типу Int32
Int32ToInt	Выполняет преобразование типа Int32 к типу Integer

Рассмотрим пример использования оператора присваивания с преобразованием типов данных:

```
Var a, b, c : Array Real [10 : Stream] Mem;
Var i: Number;
Cadr ExampleMem;
  For i := 1 to 5 do
    Begin
      c[i] := Flt2Int(a[i] + b[i]);
    End;
Endcadr;                                              (1.5)
```

Оператор присваивания выполняет присвоение результата работы функции Flt2Int элементу переменной C. Так как переменные A и B имеют тип Real, а переменная результата имеет тип Integer, то необходимо выполнение функции преобразования типа Real к типу Integer, для чего используется функция Flt2Int.

Выполнение логического оператора присваивания заключается в выселении логического выражения, которое может принимать только два значения - True или False.

Как правило, все операторы присваивания в языке COLAMO являются локальными, но в случае использования оператора присваивания над переменной типа Number он может быть интерпретирован как глобальный оператор.

1.1.7.2. Оператор цикла. Для организации циклов в языке COLAMO используется оператор FOR. Цикл FOR является одним из основных видов циклов, которые имеются во всех универсальных языках программирования. Основная идея, заложенная в его функционирование, заключается в том, что операторы, находящиеся внутри цикла, выполняются фиксированное число раз, в то время как переменная цикла (известная еще как индексная переменная) пробегает определенный ряд значений.

Синтаксис оператор цикла FOR выглядит следующим образом:

FOR *<ИндекснаяПеременная>* := *<НачальноеЗначение>* To *<КонечноеЗначение>* [Step *<ЗначениеШага>*] [While *<Условие>*] Do *<СписокОператоров>*,

где FOR, To, Step, While - ключевые слова, определяющие соответствующий им параметр в конструкции оператора FOR;

ИндекснаяПеременная - имя переменной цикла;

НачальноеЗначение - начальное значение переменной цикла;

КонечноеЗначение - выражение, определяющее предел приращений переменной цикла;

ЗначениеШага – значение приращения переменной цикла;

Условие - логическое выражение, при невыполнении которого производится выход из цикла.

В общем случае количество итераций, выполняемых оператором цикла, вычисляется формулой:

$$КоличествоИтераций = \left[\frac{КонечноеЗначение - НачальноеЗначение}{ЗначениеШага} \right] + 1 \qquad (1.6)$$

Исходя из этой формулы, максимальное значение, которое может принять *ИндекснаяПеременная*, можно рассчитать по формуле:

$$Значение = НачальноеЗначение + (КоличествИтераций * ЗначениеШага) \qquad (1.7)$$

В отличие от традиционных языков программирования оператор цикла в языке имеет дополнительный параметр While. Использование параметра While позволяет определять условие досрочного завершения выполнения оператора FOR. До тех пор пока условие в параметре While является истиной, выполняется следующая итерация цикла, в противном случае выполняется досрочный выход из цикла. Использование оператора FOR с параметром While похоже на работу оператора While…Do языка Delphi, но в отличие от данного аналога всегда обеспечивает завершение работы цикла и не приводит к его “зависанию”.

Если в списке операторов находится один оператор, то могут отсутствовать синтаксические скобки группового оператора Begin и End, в противном случае необходимо обрамлять соответствующие вычисления синтаксическими скобками.

Как и для традиционных языков программирования, операторы цикла могут быть вложенными, однако последовательность их выполнения в языке COLAMO может быть изменена в отличие от традиционных языков программирования.

Для оператора FOR в языке COLAMO существует и другая форма записи:

FOR <*ИндекснаяПеременная*> := <*КонечноеЗначение*> DownTo <*НачальноеЗначение*> [Step <*ЗначениеШага*>] [While <*Условие*>] Do <*СписокОператоров*>.

При использовании в операторе ключевого слова DownTo организуется обратный счет, т.е. начальное значение индексной переменной будет соответствовать значению параметра *КонечноеЗначение*, а конечное значение будет рассчитываться по формуле:

$$Значение = КонечноеЗначение - (КоличествИтераций * ЗначениеШага) \qquad (1.8)$$

Более подробная информация об использовании операторов цикла рассмотрена в разделе 3.

1.1.7.3. Условный оператор выбора. Оператор условного выбора предназначен для выполнения тех или иных действий в зависимости от истинности или ложности некоторого условия. В языке COLAMO Оператор IF может быть внутренним или внешним по отношению к кадру.

Условие задается выражением, имеющим результат логического типа. Синтаксис оператора условного перехода в РБНФ выглядит следующим образом:

IF <*Условие*> Then <*СписокОператоров1*> [Else <*СписокОператоров2*>] ;

Если при выполнении условного оператора условие возвращает истину (значение равное True), то выполняется оператор, указанный в конструкции IF, иначе передается управление на выполнение следующему оператору, стоящему послу оператора IF.

Если список операторов *СписокОператоров1* или *СписокОператоров2* состоит более чем из одного оператора, то необходимо использовать синтаксические скобки Begin и End, в противном случае наличие синтаксических скобок не обязательно.

Рассмотрим пример использования оператора условного выбора:

```
Var a, b, c, d : Real Mem;
Cadr ExampleIf;
   If a = b then
      c := F(a + b);
    d := a - b;
Endcadr;                                                    (1.9)
```

В рассмотренной программе оператор условного перехода влияет только на оператор присваивания C := F(A + B); так как отсутствуют синтаксические скобки обрамляющие операторы присваивания для переменной D. В данном случае если

условие A=B выполняется т.е. равно истине, то выполняется вычисление оператора C := F(A + B);, в противном случае выполняется переход к оператору D := A - B;.

В языке COLAMO, как и в традиционных языках программирования, оператор условного перехода может иметь альтернативную ветку выполнения ELSE.

Если при выполнении условного оператора параметр *Условие* возвращает значение равное True, то выполняется обработка операторов, расположенных в *СписокОператоров1*, иначе выполняются операторы, расположенные в *СписокОператоров2*.

Допускаются вложенные конструкции условных операторов:

IF *Условие1* Then IF *Условие2* Then *СписокОператоров1* Else *СписокОператоров2*;

При такой записи операторы, расположенные в блоке *СписокОператоров2*, будут принадлежать оператору условного перехода, выполняющего условие *Условие2*. В данном случае операторы блока *СписокОператоров2* будет выполнены в случае, если условие *Условие1* будет выполнено, а условие *Условие2* будет иметь значение ложь (т.е. будет иметь значение false).

Для определения принадлежности оператора Else и соответствующего ему блока *СписокОператоров2* к операторам условного перехода, выполняющим условия *Условие1* и *Условие2*, необходимо явно указать принадлежность к одному из операторов условного перехода, используя синтаксические скобки Begin и End. Рассмотрим следующий пример:

```
IF Условие1 then
  Begin
    IF Условие2 then
        СписокОператоров1 ;
  End
Else
  СписокОператоров2 ;                                    (1.10)
```

В представленной программе блок *СписокОператоров2* будет выполняться в случае невыполнения условия *Условие1* и не будет зависеть от оператора условного перехода, выполняющего обработку условия *Условие2*, так как область действия оператора, реализующего выбор необходимой ветви, согласно условию *Условие2*, ограничен синтаксическими скобками Begin и End.

Особенности использования оператора условного выбора рассмотрено в разделе 3.

1.1.7.4. Условный оператор множественного выбора Switch. Для реализации ветвлений в языке введен условный оператор множественного выбора Switch. Условный оператор Switch позволяет сразу выполнять выбор одной ветви из множества взаимоисключающих ветвей исполнения в отличие от условного оператора IF.

Оператор Switch выполняет анализ значения некоторого выражения и в зависимости от его значения выполнит те или иные действия. В общем случае формат записи условного оператора множественного выбора в РБНФ выглядит следующим образом:

```
Switch <Выражение> Of
 Begin
   Case <Значение1> : <СписокОператоров1>;
   Case <Значение2> : <СписокОператоров2>;
   .........
   Case <ЗначениеN> : <СписокОператоровN>;
   Default : <СписокОператоров>;
 End;                                                    (1.11)
```

Результатом вычисления параметра *Выражение* в операторе Switch … Of должно быть целое число, которое будет определять, какую ветку Case необходимо выполнить.

В процессе выбора необходимого блока Case выполняется сравнение вычисленного значения параметра *Выражение* с *ЗначениеN*, соответствующего каждому блоку Case.

Наличие блока Default в операторе Switch необязательно.

В отличие от языка C++, для которого наличие синтаксических скобок необязательно, при объединении операторов в блоки в языке COLAMO наличие синтаксических скобок обязательно в случае обработки нескольких операторов одним блоком Case.

Рассмотрим пример использования оператора Switch

```
For i := 0 to 3 do
  Switch i Of
     Begin
        Case 0 : Begin
                      c := F(a + b);
                      d := F(a);
                  End;
        Case 1 : c := F(a - b);
```

```
        Default : d := a - b;
    End;                                              (1.12)
```

В рассмотренном примере в случае значения индексной переменной, равного 0, выполняется блок Case 0, в котором будут вычислены значения переменных C и D. На каждой итерации цикла FOR будет выполняться только один блок Case. В случае отсутствия соответствующего блока для индексной переменной I выполняются операторы, соответствующие блоку Default, если он присутствует в операторе Switch.

Особенности обработки оператора Switch рассмотрены в подразделе 3.2.

1.2 Семантика языка

1.2.1 Семантика использования переменных

В языке COAMO, как уже упоминалось ранее, существует четыре способа хранения переменных, определяемые следующими ключевыми словами языка COLAMO: Mem, Com, Reg, InterMem и InterMem2. В зависимости от способа хранения переменной на её использование в программе накладываются следующие правила:

- правило однократного присваивания;
- правило единственной подстановки.

Правило однократного присваивания указывает на необходимость использования переменной на запись только один раз в один момент времени. Правило единственной подстановки накладывает ограничение на использование переменной как на запись, так и на чтение и требует от программиста точно определять, какие действия будут выполняться над переменной - запись или чтение в процессе выполнения всех вычислений параллельной программы.

1.2.2 Мемориальная переменная

Мемориальной переменной называется величина, хранящаяся в ячейке памяти и, следовательно, сохраняющая свое значение до очередного присваивания. Рассмотрим пример использования мемориальной переменной на примере, показанном на рисунке 1.1.

```
Var a,b,c,d : Array Integer [10 : Stream] Mem;
Var i: Number;
Cadr ExampleMem;
  For i := 1 To 5 Do
    Begin
      a[i] := b[i] · c[i] + d[i];
    End;
Endcadr;
```

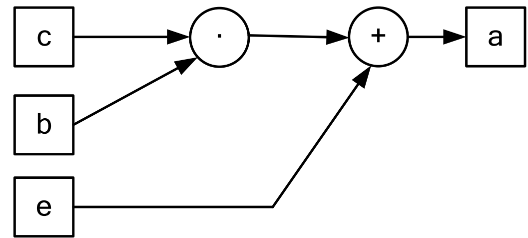


Рисунок 1.1 - Использование мемориальной переменной

Пример, представленный на рисунке 1.1, является синтаксически правильным и удовлетворяет всем принципам языка COLAMO. Ограничением, накладываемым на использование мемориальных переменных, является правило единственной подстановки.

В данном случае запрещается использование в левых и правых частях вычислительных выражений переменных, расположенных в одинаковых каналах памяти. Пример данного случая показан ниже:

```
Var a,b,c,d : Array Integer [10 : Stream] Mem;
Var i : Number;
Cadr ExamplePEP;
  For i := 1 to 5 do
    Begin
      a[i] := b[i] + c[i];
      d[i] := a[i] + c[i];
    End;
EndCadr;                                                    (1.13)
```

В рассмотренном примере программист пытается одновременно выполнить два разных процесса: чтение и запись, так как в языке COLAMO все вычисления выполняются параллельно, что является синтаксической ошибкой.

Еще одним ограничением, накладываемым на мемориальные переменные, является правило однократного использования. В данном случае запрещается одновременное выполнение операции присваивания для одной и той же мемориальной переменной. Пример данного случая показан ниже:

```
Var a,b,c,d : Array Integer [10 : Stream] Mem;
Var i : Number;
Cadr ExamplePOP;
  For i := 1 to 5 do
    Begin
```

```
        a[i] := b[i] + c[i];
        a[i] := d[i] + c[i];
      End;
EndCadr;                                                    (1.14)
```

В рассмотренном примере программист пытается одновременно выполнить два процесса записи в один и тот же канал памяти, что является синтаксической ошибкой.

Более сложной для понимания является реализация в теле циклов двух процессов чтения, рассмотренных на следующем примере:

```
Var a, c, d : Array Integer [100 : Stream] Mem;
Var i, j : Nbumber;
Cadr B;
   For i:= 0 to 99 do
      c[i] := f1(a[i]);
   For j:= 0 to 99 do
      d[j] := f2(a[j]);
Endcadr;
```

Контроллер распределенной памяти не может реализовать два независимых процесса для мемориальных переменных, в результате будет выдано предупреждение. В данном случае результат работы программы - труднопрогнозируемый.

Данный текст целесообразно свести к одному процессу, как это показано на следующем примере:

```
Var a, c, d : Array Integer [100 : Stream] Mem;
Var i, j : Nbumber;
Cadr B;
        For i := 0 to 99 do
           Begin;
              c[i] := f1(a[i]);
              d[j] := f2(a[j]);
           End;
Endcadr;
```

Также можно записать текст программы в виде двух кадров:

```
Var a,c,d : Array Integer [100 : Stream] Mem;
```

```
Var i, j : Nbumber;
Cadr B1;
    For i := 0 to 99 do
        c[i] := f1(a[i]);
Endcadr;
Cadr B2;
    For j := 0 to 99 do
         d[j] := f2(a[j]);
Endcadr;
```

В разных кадрах, т.е. независимых вычислительных структурах могут выполняться различные процессы.

1.2.3 Регистровая переменная

При реализации программы возникает необходимость запоминания текущих значений, полученных в процессе вычислений, и их дальнейшего использования в определенные моменты времени, например, при выполнении следующего кадра.

Для реализации этой возможности в языке COLAMO используются регистровые переменные.

В отличие от мемориальных переменных на регистровую переменную действует только правило однократного присваивания, что делает регистровые переменные более гибкими в использовании.

Рассмотрим использование регистровых переменных на примере программы, показанной на рисунке 1.2.

```
Var a, b, c, f : Integer Mem;
Var z : Integer Reg;
Var i : Number;
Cadr ExampleReg;
   z := b * c;
   a := z + f;
EndCadr;
```

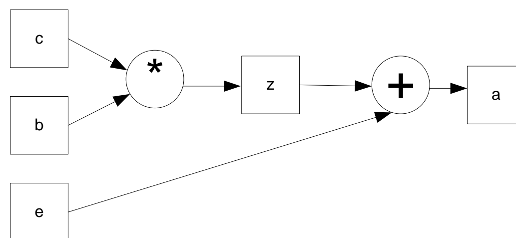


Рисунок 1.2 - Использование регистровой переменной

Однако использование регистровых переменных может привести к неоднозначности выполнения программы. Это связано с тем, что все компоненты кадра выполняются асинхронно и независимо, и невозможно однозначно сказать, что выполнится первым – чтение или запись в регистровую переменную,

следовательно, неизвестно, корректны ли будут вычислительные операции в данном случае или нет.

Решением в данной ситуации является разделение некорректного кадра на несколько кадров и использование структуры Let для реализации операторов, расположенных в теле кадра. В этом случае порядок выполнения кадров жестко регламентирует значения получаемых переменных:

```
Var a, b, c, d : Integer Mem;
Var i : Number;
Let Calc(In : b1,c1,d1, Index; Out : a1);
Var a,b,c,d, Index : Integer Com;
Var z : Integer Reg;
   If  Index = 0 then
      z := b1 + c1;
   a1 := z + d1;
EndLet;
Cadr S1;
  Calc(b, c, d, 0, a);
Endcadr;
Cadr S2;
  Calc(b, c, d, 1, a]);
Endcadr;                                                    (1.15)
```

Подобная конструкция обеспечивает детерминизм получения результатов, но сокращает число арифметико-логических операций в кадре и, как следствие, количество элементарных процессоров (ЭП), реализующих кадр. Однако это приводит к уменьшению реальной производительности при решении задачи и снижает эффективность РВС.

1.2.4 Коммутационная переменная

В целях оптимизации использования вычислительных ресурсов РВС для реализации вычислительной структуры можно использовать коммутационные переменные, как показано на рисунке 1.3.

Имя коммутационной переменной "сцепляется" с вершиной графа, соответствующей элементарному процессору, по аналогии с "сцеплением" (hoked), реализованным в языке Пролог. Использование коммутационной переменной не требует использования дополнительной ячейки памяти, как в случае с регистровой переменной.

```
Var a,b,c,d,e : Array Real [10 : Stream] Mem;
Var z : Real Com;
Cadr ExampleCom;
  For I := 0 to 9 do
    Begin
      z := b[I] * c[I];
      a[I] := z + e[I];
    End;
EndCadr;
```

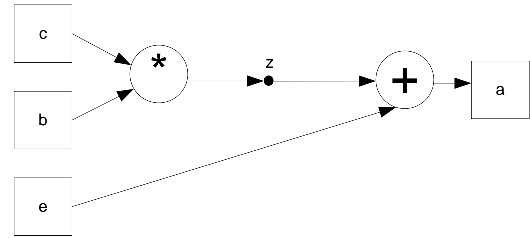


Рисунок 1.3 - Использование коммутационной переменной

На коммутационную переменную, как и на регистровую переменную, действует правило однократного присваивания.

Недостатком использования коммутационных переменных является то, что коммутационная переменная представляет собой точку на графе, значение которой получить невозможно.

Решением в данном случае является сохранение значения коммутационной переменной в регистровой переменной или мемориальной переменной в зависимости от имеющегося ресурса вычислительной системы.

1.2.5 Внутренняя память

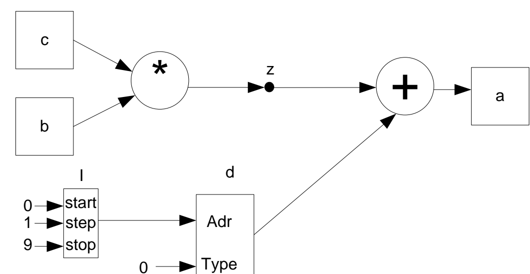
Как говорилось ранее, в язык COLAMO добавлены новые способы хранения переменных InterMem и InterMemN, обеспечивающие хранение данных во внутренней памяти ПЛИС, где N – количество доступных портов при работе с внутренней памятью ПЛИС.

Способ хранения переменных InterMem используется для работы с однопортовой внутренней памятью ПЛИС.

Для переменной однопортовой внутренней памяти, как и для мемориальной переменной, действуют правила однократного присваивания и единственной подстановки. Данные ограничения накладываются вследствие особенностей работы однопортовой внутренней памяти ПЛИС, которая позволяет осуществлять над ней только один процесс записи или чтения.

Рассмотрим пример использования однопортовой внутренней памяти, представленный на рисунке 1.4.

```
Var a,b,c : Array Real [10 : Stream] Mem;
Var d : Array Real [10 : Stream] InterMem;
Var z : Real Com;
Var i : Number;
Cadr ExampleInterMem;
  For I := 0 to 9 do
    Begin
      z := b[i] * c[i];
```



```
    a[i] := z + d[i];
  End;
EndCadr;
```

Рисунок 1.4 - Использование однопортовой внутренней памяти

В рассмотренной программе, на рисунке 1.4 программист использует однопортовую внутреннюю память, в которой хранится переменная D, так как для нее указан способ хранения InterMem. Блок внутренней памяти имеет два входа данных: adr – определяет адрес чтения или записи и Type – определяет, как будет использоваться внутренняя память на чтение или запись.

Хранение переменной в двухпортовой памяти определяется директивой InterMem2.

При работе с переменной, соответствующей двухпортовой внутренней памяти, отсутствует ограничение, связанное с правилом единственной подстановки. Отсутствие данного ограничения связано с тем, что двухпортовая внутренняя память позволяет выполнять одновременно процесс чтения и записи.

Рассмотрим пример использования двухпортовой внутренней памяти, представленный на рисунке 1.5.

```
Var a,b,c,e : Array Real [10 : Stream] Mem;
Var d : Array Real [10 : Stream] InterMem2;
Var i : Number;
Cadr ExampleInterMem2;
  For i := 2 to 9 do
    Begin
      d[i] := b[i] * c[i];
      a[i] := e[i] + d[i - 2];
    End;
EndCadr;
```

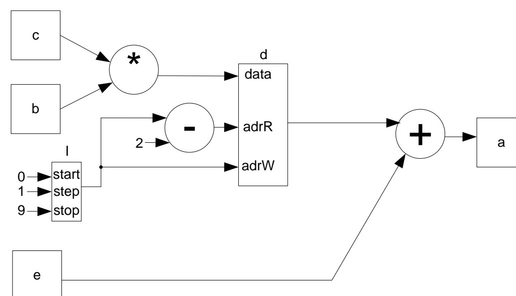


Рисунок 1.5 - Использование двухпортовой внутренней памяти

В рассмотренной программе программист использует двухпортовую внутреннею память, в которой хранится переменная D так как для нее указан способ хранения InterMem2. Блок внутренней памяти имеет три входа данных: adrR – адрес чтения данных, adrW – адрес записи данных и Data – вход данных, которые будут записаны по адресу adrW.

Использование процессов записи и чтения больше чем количество портов используемой внутренней памяти является синтаксической ошибкой.

Более подробное описание работы с внутреннею памятью ПЛИС рассмотрено в подпункте 1.3.4.4.

1.2.6 Битовые переменные

Для работы с данными на уровне битов в языке COLAMO введена битовая переменная.

Для работы с битовыми переменными используются логические операции (And, Or, Xor) и добавлены в язык COLAMO новые битовые операции (Shl, Shr, ShlC, ShrC) представленные в таблице 1.3.

Таблица 1.3 - Битовые операции

Мнемоника	Формат	Действие
SHR	A := A Shr N;	Выполняет сдвиг битов в переменной A вправо на N разрядов
SHL	A := A Shl N;	Выполняет сдвиг битов в переменной A влево на N разрядов
CSR	A := A Csr N;	Выполняет циклический сдвиг битов в переменной A вправо на N разрядов
CSL	A := A Csl N;	Выполняет циклический сдвиг битов в переменной A вправо на N разрядов

В зависимости от способа хранения битовой переменой различается их обработка.

Рассмотрим пример использования мемориальной битовой переменной:

```
Var a,b,c : Bit Mem;
Cadr ExampleBitMem;
   a := b and c;
EndCadr;
```

В представленной программе каждая битовая переменная a, b и c расположена в отдельном канале памяти и занимает одну ячейку памяти. В таком случае значимым в каждой ячейке памяти, зарезервированной под хранения битовой переменной, будет только младший бит, следовательно, такое объявление переменой A будет ресурсозатратным.

Такая же ситуация возникает и при использовании битовых массивов. Пусть объявлена переменная A вида: Var A : Array Bit [10 : Vector, 100 : Stream] **Mem**;. В таком случае программист выделяет по сто ячеек в десяти каналах памяти, в каждой из которых значимым будет также младший бит.

Такое использование мемориальных битовых переменных является нецелесообразным и, как правило, для обработки данных на уровне бит используются массивы битовых переменных, имеющие параллельный доступ к

элементам массива, и способ хранения COM. В данном случае объявление переменой выглядит следующим образом:

Var A : Array Bit [10 : Vector] Com;

Рассмотрим пример программы, в котором программист хочет использовать битовые коммутационные массивы и выполнить над ними логическую операцию AND.

```
Var a, b, c : Array Bit [100: Stream] Mem;
Var d, e, f : Array Bit [32 : Vector, 100: Stream] Com;
Var i : Number;
Cadr BitUse;
  For i := 0 to 99 do
    Begin
      d[*,i] := Separate (a[i]);
      e[*,i] := Separate (b[i]);
      f[*,i] := d[*,i] And e[*,i];
      a[i] := Combine(f[*,i]);
    End;
EndCadr;                                                  (1.16)
```

В данной программе указаны функции SEPARATE и COMBINE. С помощью функции Separate программист выполняет разложение 32-разрядого значения переменой A[i] на независимую друг от друга последовательность битов. Полученная последовательность битов будет принадлежать параметру «*», указанному при обращении к переменой D. В данном случае ограничением для переменной D является количество векторных элементов, соответствующих параметру «*», указанных при объявлении переменной D. На рисунке 1.6 показана граф-схема использования функции Separate.

```
Var a, b : Array Bit [100: Stream] Mem;
Var d, e : Array Bit [32 : Vector, 100: Stream]
Com;
Var i : Number;
Cadr BitUse;
  For i := 0 to 99 do
    Begin
      d[*,i] := Separate (a[i]);
      e[*,i] := d[*,i] shl 3;
      b[i] := Combine(e[*,i]);
    End;
EndCadr;
```

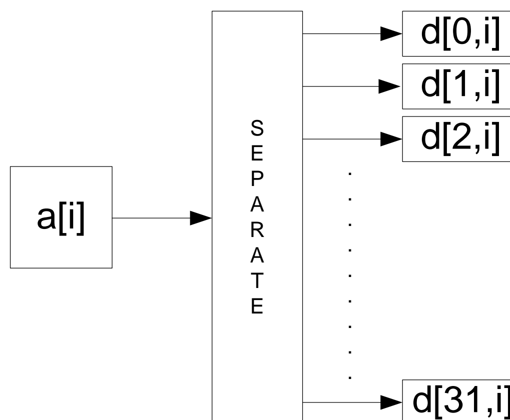


Рисунок 1.6 - Граф-схема использования функции Separate

Функция Combine выполняет сборку всех битов, принадлежащих параметру «*», указанному при обращении к переменой E. В данном случае результатом работы функции Combine будет 32-разрядное слово. На рисунке 1.7 показана граф-схема использования функции Combine.

```
Var a, b : Array Bit [100: Stream] Mem;
Var d, e : Array Bit [32 : Vector, 100: Stream]
 Com;
Var i : Number;
Cadr BitUse;
  For i := 0 to 99 do
    Begin
      d[*,i] := Separate (a[i]);
      e[*,i] := d[*,i] shl 3;
      b[i] := Combine(e[*,i]);
    End;
EndCadr;
```

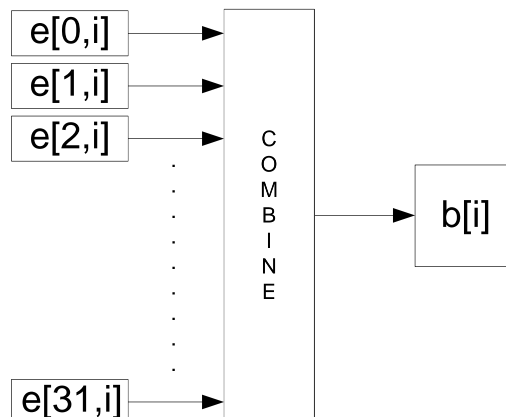


Рисунок 1.7 - Граф-схема использования функции Combine

Функции SEPARATE и COMBINE применяются только при использовании коммутационных и регистровых переменных.

1.2.7 Семантика обращения к элементам массива

Организация доступа к элементам массива в языке COLAMO соответствует традиционным языкам программирования. Для удобства программирования на языке COLAMO для обращения к элементам массива используются срезы и выборки.

В языке COLAMO в зависимости от типа индексации элементов массивов организация потоков данных будет отличаться. В языке COLAMO различаются следующие типы адресации элементов массива: индексная, индексная с базовым смещением и косвенная адресация.

1.2.8 Индексная и косвенная адресация

Индексная организация доступа в языке COLAMO подразумевает использование для адресации к элементам массива переменных типа Number и констант.

Рассмотрим пример индексной адресации:

```
Var a,b,c : Array Real [10 : Stream] Mem;
Var i : Number;
Cadr ExampleIndexAdr;
  For i := 0 to 9 do
```

```
    Begin
      a[i] := b[i] + c[i];
    End;
EndCadr;                                                    (1.17)
```

Данный пример демонстрирует организацию индексного доступа к переменным a, b и c. В данном случае для при организации потоков данных в контроллерах распределенной памяти будут использоваться внутренние адреса.

Использование констант или постоянных величин с индексной переменной, над которыми выполняются аддитивные операции, является индексной адресацией с базовым смещением. Рассмотрим пример использования индексной адресации с базовым смещением:

```
Var a,b,c : Array Real [10 : Stream] Mem;
Var i : Number;
Cadr ExampleIndexConst;
  For i := 0 to 9 do
    Begin
      a[i + 3] := b[i] + c[i];
    End;
EndCadr;                                                    (1.18)
```

Организация адресации для переменной A является индексной адресацией с базовым смещением, а для переменных b и c является индексной адресацией. Организация потоков данных при использовании индексной адресации с базовым смещением ничем не отличается от использования просто индексной адресации.

Использование индексной адресации в том или ином виде позволяет организовывать плотные потоки данных, что обеспечивает наибольшую скорость обработки информации в отличие от использования косвенной адресации.

Косвенная адресация подразумевает собой получение данного по любому адресу в канале памяти. При использовании косвенной адресации вычисление адреса реализуется на структурном уровне, что приводит к использованию операторов организации потоков данных с внешней адресацией.

Рассмотрим пример использования косвенной адресации (см. рис. 1.8).

```
Var a,b,c,d : Array Real [10 : Stream] Mem;
Var i : Number;
Cadr ExampleIndexConst;
  For i := 0 to 9 do
    Begin
      a[d[i]] := b[i] + c[i];
    End;
EndCadr;
```

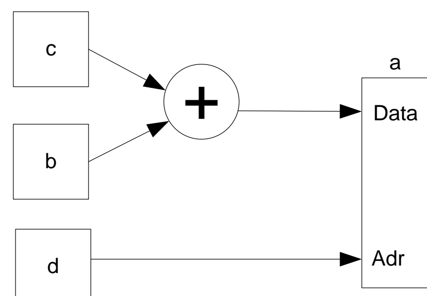


Рисунок 1.8 - Программа и граф-схема использования косвенной адресации

Обращение к переменной A является примером ярко выраженной косвенной адресации, так как адрес записи данных переменной A определяется значением элемента массива d[I]. Таким образом, в отличие от всех рассмотренных ранее примеров в контроллере распределенной памяти будет задействован вход адреса Adr.

Следует отметить, что скорость доступа к элементам массива A падает, так как необходимо выполнить определение значения элемента d[I], на основании полученного значения определить необходимую ячейку памяти в канале памяти переменной A и выполнить в нее запись.

Неявной косвенной адресацией в языке COLAMO является адресация, полученная в ходе использования неаддитивных операций при вычислении адреса доступа к элементу массива, или использование нескольких индексных переменных операторов цикла.

Рассмотрим пример неявной косвенной адресации при использовании неаддитивных операций (см. рис. 1.9)

```
Var a,b,c : Array Real [10 : Stream] Mem;
Var i : Number;
Cadr ExampleIndexConst;
  For i := 1 to 9 do
    Begin
      a[I * 3] := b[i] + c[i - 1];
    End;
EndCadr;
```

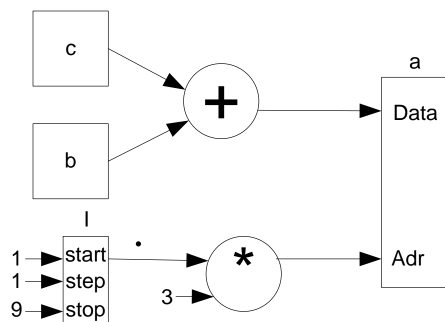


Рисунок 1.9 - Программа и граф-схема использования косвенной адресации

В программе, рассмотренной на рисунке 1.9, для адресации к переменной A используется косвенная адресация, так как для адресации к элементам массива используется неаддитивная операция умножения. В этом случае все вычисления адреса элемента массива реализованы структурно. Вообще, такой тип адресации можно расценивать по-разному и свести тип адресации не к косвенной, а к индексной, но при этом оператор цикла с индексной переменной I должен иметь постоянный шаг, равный 1.

Использование нескольких индексных переменных при обращении к элементам массива также расценивается как вид косвенной адресации.

Рассмотрим пример использования нескольких индексных переменных при обращении к элементам массива на рисунке 1.10.

В программе, рассмотренной на рисунке 1.10, для адресации к переменной A используется косвенная адресация, так как использование нескольких индексных переменных не позволяет однозначно определить шаг изменения адреса памяти на

каждой итерации циклов, поскольку он, в общем случае, не является величиной постоянной, хотя и обладает некоторой закономерностью изменений.

```
Var a,b,c,d : Array Real [10 : Stream, 100 :
Stream] Mem;
Var i, j : Number;
Cadr ExampleTwoIndex;
  For i := 2 to 9 do
    For j := 0 to 99 do
      Begin
        a[I + j] := b[j] + c[j - 1];
      End;
EndCadr;
```

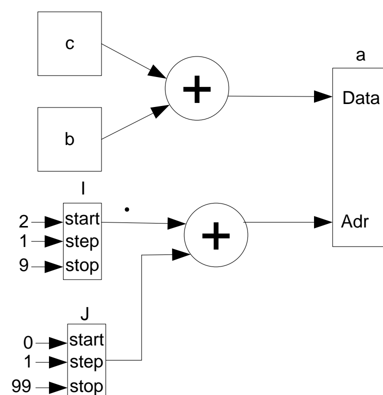


Рисунок 1.10 - Использование нескольких индексных переменных

Использование косвенной адресации в одном из каналов памяти ведет к снижению темпа подачи данных для этого канала памяти, что приводит к замедлению темпа подачи данных для всех каналов памяти, независимого от того, какой тип адресации для них используется.

1.2.9 Срезы и выборки

Для удобства программирования в языке COLAMO используются срезы и выборки при организации доступа к элементам массива.

Срезом в языке COLAMO является перебор всех возможных элементов массива, соответствующих параметру, к которому применяется срез от начального элемента до конечного. Для применения среза при доступе к элементам массива в языке COLAMO используется символ «*». Символ «*» в зависимости от контекста может иметь различное предназначение.

Рассмотрим пример использования символа «*» в качестве среза:

```
Var a,b,c : Array Real [10 : Stream] Mem;
Cadr ExampleSrez;
    a[*] := b[*] + c[*];
EndCadr;                                                        (1.19)
```

В рассмотренной программе для индексации к элементам массивов используются срезы, которые означают, что необходимо сложить попарно элементы массивов B и C и записать их в массив A. Если развернуть данные вычисления можно получить следующую программу:

```
Var a,b,c : Array Real [10 : Stream] Mem;
Var i : Number;
```

```
Cadr ExampleSrez;
  For i := 0 to 9 do
    Begin
      a[i] := b[i] + c[i];
    End;
EndCadr;                                                    (1.20)
```

В данном случае оператор цикла выполняет перебор всех возможных элементов массива b и c. Как видно из представленных программ, использование срезов позволяет упростить программирование задач.

Использование символа «*» при вызове вычислительных структур должно использоваться для массивов имеющих параллельный тип доступа. Рассмотрим пример использования символа «*» со структурой подкадр:

```
Var a : Array Integer [3 : Vector] Com;
Var b : Integer Com;
SubCadr Change(in a1, b1);
Var a1 : Array Integer [3 : Vector] Com;
Var b1 : Integer Com;
    b1 := a1[0] + a1[1] + a1[2];
EndSubCadr;
Cadr ExampleVector;
    Change(a[*], b);
EndCadr;                                                    (1.21)
```

В данной программе символ «*» указывает на необходимость развертки массива А, при которой каждому элементу массива А будет соответствовать свой вход в структуре Change. Данная программа будет эквивалентна следующей программе:

```
Var a : Array Integer [3 : Vector] Com;
Var b : Integer Com;
SubCadr Change(in a1, b1);
Var a1 : Array Integer [3 : Vector] Com;
Var b1 : Integer Com;
    b1 := a1[0] + a1[1] + a1[2];
EndSubCadr;
Cadr ExampleVector;
    Change(a[0], a[1], a[2], b);
EndCadr;                                                    (1.22)
```

Еще одним случаем использования символа «*» является работа с битовыми переменными, рассмотренная в подразделе 2.1.6.

Основным недостатком срезов является полный перебор всех элементов массива, что не всегда удобно. Для устранения этого недостатка в языке COLAMO используются выборки.

Выборка в языке COLAMO позволяет получить доступ к элементам массива в определенном диапазоне, указанном программистом.

Синтаксис выборки выглядит следующим образом N : M, где N – начальный элемент выборки, а M – конечный элемент выборки.

Выборка, как и срез элементов массива, осуществляется с постоянным шагом, равным 1.

Рассмотрим пример использования выборки:

```
Var a,b,c : Array Real [10 : Stream] Mem;
Cadr Vibor;
    a[0 : 3] := b[2 : 5] + c[6 : 9];
EndCadr;                                                    (1.23)
```

В данном примере программист указал конкретные диапазоны элементов, которые будут участвовать в обработке. Данный пример можно переписать следующим образом:

```
Var a,b,c : Array Real [10 : Stream] Mem;
Var i : Number;
Cadr Vibor;
   For  i := 6 to 9 do
      a[i - 6] := b[i - 4] + c[i];
EndCadr;                                                    (1.24)
```

В представленном примере оператор цикла FOR организует поток данных (элементов массивов), а разное начальное значение выборки в массивах a, b и c реализуется за счет смещения индексов при доступе к элементам массивов.

1.2.10 Каскадирование вычислительных структур

На практике при создании параллельно-конвейерных программ для РВС необходимо создавать каскадированные структуры, которым соответствует последовательное соединение изоморфных базовых подграфов. На рисунке 1.11 представлен фрагмент информационного подграфа, содержащего каскадированный базовый подграф.

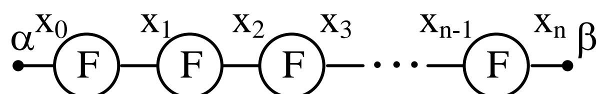


Рисунок 1.11 - Фрагмент информационного подграфа, содержащего каскадированный базовый подграф

Данной структуре соответствует формула

$$\beta = F(F(F...F(\alpha)), \tag{1.25}$$

которую можно записать методом композиции следующим образом:

$$\beta = \overset{\wedge}{\underset{i=1}{o}} F(\alpha). \tag{1.26}$$

Несложно заметить, что подобная конструкция соответствует возведению в *n*-ю степень функции, реализованной в базовом подграфе N. В настоящее время данная структура может быть описана на языке программирования с неявным описанием параллелизма в соответствии со следующей программой:

```
Var X: array Real [n: vector] Com;
Var α: Real Mem;
Var I: Number;
Const n=99;
Cadr Example24;
X[0]:=α;
      For i:=1 to n do
  X[i]:=F(X[i-1]);
            β:= X[n];
Endcadr;                                                    (1.27)
```

Здесь цикл по переменной i задает каскадирование, а начальные значения структуры и перезапись результатов производятся с помощью отдельных операторов. Подобную вычислительную структуру можно просто записать в следующем виде

β:=F<n>(α);

Введение подобной конструкции позволяет значительно сократить текст описания и сделать его более наглядным.

Разумеется, подобная структура является частным случаем построения каскадных вычислений, которые зависят только от одной переменной. На рисунке 1.12 показана обобщенная каскадированная структура, в которой базовый подграф мультиплицирован n раз, при этом базовый подграф имеет множество общих информационных входов A_1, A_2,….A_m.

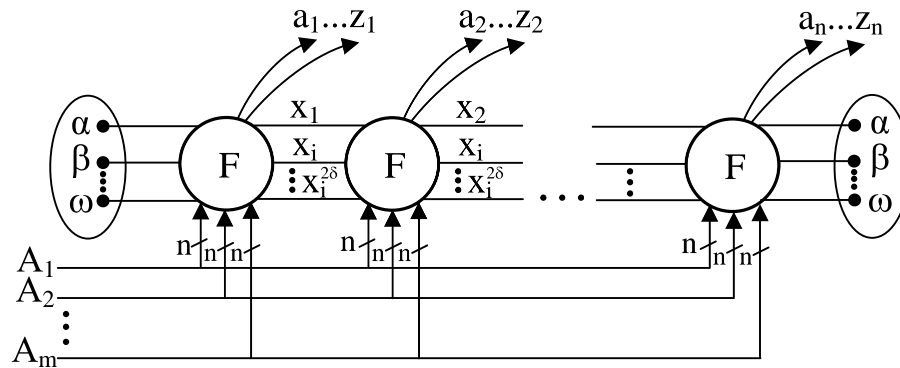


Рисунок 1.12 - Обобщенная каскадированная структура

Кроме того, каждый базовый подграф F_i имеет непересекаемые множества выходов $a_i \ldots z_i$. И, наконец, имеется множество какскадируемых выходов α, β, …., ω, которое преобразуется в результирующее множество α', β', …., ω'.

Данная вычислительная структура может быть описана с помощью оператора

$$F < n > (< \alpha, \alpha' >, < \beta, \beta' >, ..., < \omega, \omega' >, A_1, A_2, ..., A_m, a_{g(n)}, ..., z_{g(n)}) . \quad (1.28)$$

Здесь в угловых скобках стоят смежные входные и выходные каналы каскадной структуры. После описания каскадных структур следует список общих входов в базовые подграфы каскадной структуры, а далее - список уникальных выходов из каждого базового подграфа каскадной структуры.

Введение подобного описания позволяет значительно упростить построение каскадных структур, которые широко используются при распараллеливании по итерациям. Введение кадровых структур позволяет оставить неизменным число каналов для базовой реализации параллельно-конвейерной программы, в то время как для распараллеливания по данным, чему соответствует использование векторных массивов, приводит к пропорциональному увеличению числа задействованных информационных каналов. Каскадирование вычислений обладает значительно более высоким потенциалом распараллеливания в широком смысле этого слова обработки данных по сравнению с векторной обработкой, поскольку число каналов в РВС ограничено и существенно меньше числа блоков функциональной обработки информации.