УДК 681.58.011.56

В.Г. Сухоребрый, Д.В. Джулгаков

Принципы генерации текстовых эквивалентов для программ, созданных на графическом языке FBD

Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ»
Харьковское территориальное отделение Малой академии наук Украины

Современные системы автоматизированного проектирования программного обеспечения (САПР ПО) промышленных контроллеров систем управления, как правило, используют технологию визуального программирования на основе одного или нескольких графических языков [1]. Наиболее популярным среди разработчиков прикладных алгоритмов и программ является язык схем функциональных блоков – FBD (functional block diagram). Во многих современных системах автоматизированного визуального программирования, как зарубежных, так и отечественных, язык FBD заявлен в качестве основного средства описания функциональных алгоритмов для систем управления технологическим оборудованием.

Программа на языке функциональных схем (FBD) представляет собой разновидность схемы потоков данных. Согласно диалекту языка FBD, регламентированному международным стандартом IEC 61131-3, в графической программе отсутствует явная передача управления, т.е. функциональные блоки, как элементарные, так и процедурные, соединены только линиями передачи данных.

В связи с широкой популярностью систем визуального программирования перед разработчиками таких систем возникают вопросы оптимального и логически адекватного формирования текстовых эквивалентов для программ, созданных пользователями на графических языках, в частности FBD. Особые сложности возникают с трансляцией в текстовый эквивалент многоуровневых логических конструкций. Разрешение этой проблемы, а не только функциональные возможности графического редактора схем FBD, в конечном итоге определяют популярность и коммерческий успех той или иной САПР ПО.

Цель данной работы – определение и анализ возможных алгоритмов трансляции FBD-подобных программ с точки зрения оптимизации времени выполнения.

Элементарный алгоритм генерации текста для FBD-программы

В качестве математической модели схемы потоков данных используем ориентированный ациклический граф общего вида $G = (V, E)$, где $V = \{v_i\}$ – множество вершин, $E = \{(v_i, v_j) \mid v_i \neq v_j\}$ – множество дуг. При этом гипердуга рассматривается как множество одиночных простых дуг. Хотя явные циклы в FBD-схемах не запрещены стандартом [2], в большинстве реализаций языка их не используют, и поэтому циклический обмен данными между блоками программы необходимо реализовать неявно с помощью переменных. Такой подход дает возможность пользователю явно задавать порядок выполнения вершин в цикле.

Поскольку программа является динамическим объектом, важно определить порядок выполнения ее элементов, или, в терминах нашей модели, порядок обхода вершин графа *G*. Он устанавливается следующими правилами ([2], с. 251):

1) ни один из элементов схемы не может быть выполнен, пока не будут вычислены все его входы;
2) выполнение элемента схемы не может быть закончено, пока не будут вычислены все его выходы.

Таким образом, основной задачей генерации текстового эквивалента FBD-программы является формирование такого порядка на вершинах графа, при котором соблюдаются названные условия.

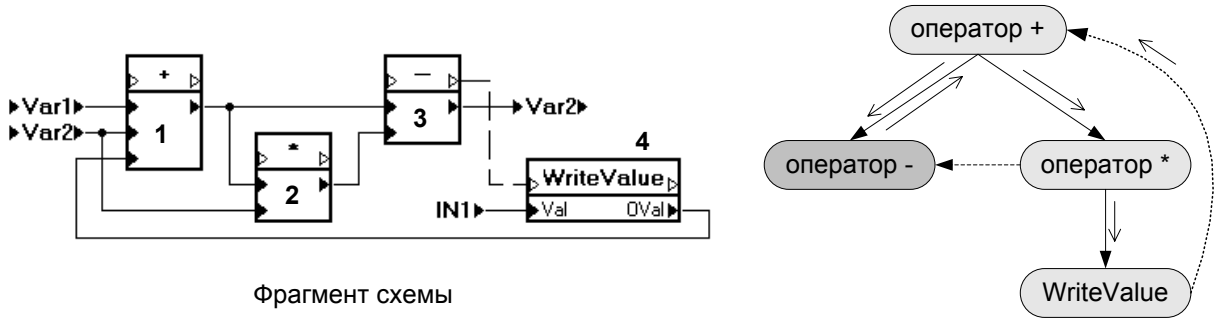
Для построения динамической части модели введем определение: для каждой вершины $a \in V$ существует множество вершин ***M(a)***, от которых *зависит* вершина ***a***:

$$M(a) = \{ v_i \in V \mid \exists (v_i,c_1)(c_1,c_2)\ldots(c_n,a),\ c_j \in V \}.$$

Множество ***M(a)*** указывает на необходимый порядок выполнения вершин схемы. Все вершины, входящие в ***M(a)***, должны выполняться раньше ***a***. Если же две вершины ***a*** и ***b*** таковы, что $a \notin M(b),\ b \notin M(a),$ то относительный порядок выполнения вершин ***a*** и ***b*** произволен. Таким образом, задача свелась к расположению всех вершин ациклического ориентированного графа на прямой так, чтобы каждая вершина ***a*** располагалась *правее* всех вершин из ***M(a)*** или чтобы все дуги графа шли слева направо. Такая задача является классической задачей на графах и известна под названием «топологическая сортировка» [3]. Алгоритм решения этой задачи может быть основан на поиске в глубину и состоит в следующем:

1) вызвать рекурсивную процедуру поиска в глубину;
2) при этом, завершая обработку вершины, добавлять ее в начало списка;
3) вернуть построенный список вершин.

Таким образом, образованный список вершин будет упорядочен по времени окончания обработки вершин и будет представлять собой искомый порядок. Следует отметить, что если в схеме существуют циклы, то поиск в глубину их обнаружит, поскольку каждый цикл образует обратное ребро в дереве поиска в глубину. Необходимо лишь добавить обработку этой ситуации в алгоритм. Схема работы этого алгоритма показана на рис.1. В приведенном примере схема содержит цикл, который обнаруживается в процессе поиска в глубину. Сложность алгоритма равна сложности поиска в глубину и составляет ***O(V+E).***



Фрагмент схемы

Дерево поиска в глубину, соответствующее фрагменту схемы

Рисунок 1 – Схема работы простого алгоритма анализа FBD-схемы

Расширенный диалект FBD-схем

Использование условных вычислений в соответствии со стандартом языка FBD ограничено, поскольку имеется возможность условного выполнения только одиночной вершины. Однако во многих задачах необходимы алгоритмы с условным выполнением фрагментов схемы. Поэтому в современных диалектах языка FBD предусмотрены условные дуги, отличающиеся от определенных стандартом.

При этом у каждой вершины-действия схемы добавляются вход и выход управления, которые отображаются в заголовке вершины. На рис. 2 представлен вариант изображения блока в соответствии с расширением диалекта FBD.

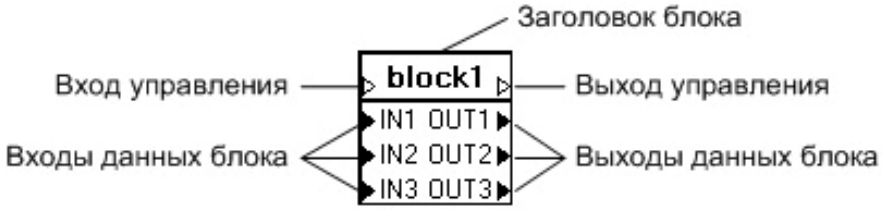


Рисунок 2 – Пример изображения блока в расширенной версии диалекта FBD

На расширенной схеме FBD все дуги делятся на несколько категорий. **Дуга данных** – это гипердуга, соединяющая один выход вершины с несколькими входами других вершин. Дуги данных, по сути, являются дугами, определенными стандартом. **Условной дугой** будем называть дугу, соединяющую выход данных логического типа одной вершины с входом управления некоторого блока. Условная дуга означает, что блок, на вход которого она заведена, будет выполняться только в том случае, если значение дуги истинно. **Дугой управления** будем называть дугу, соединяющую несколько выходов управления блоков с одним входом управления другого блока. При этом если источник дуги всего один, то это **простая дуга управления**, а если несколько – то **составная дуга**. Дуги управления дают возможность пользователю задать дополнительный порядок на вершинах схемы, т.е. указать, что вершина-получатель дуги управления должна выполняться строго после вершин-источников этой дуги. Кроме того, составная дуга управления позволяет указать, что вершина-получатель должна быть выполнена, если выполнялась хотя бы одна из вершин-источников дуги, т.е. реализовать операцию логического ИЛИ. Примеры различных категорий дуг показаны на рис. 3.

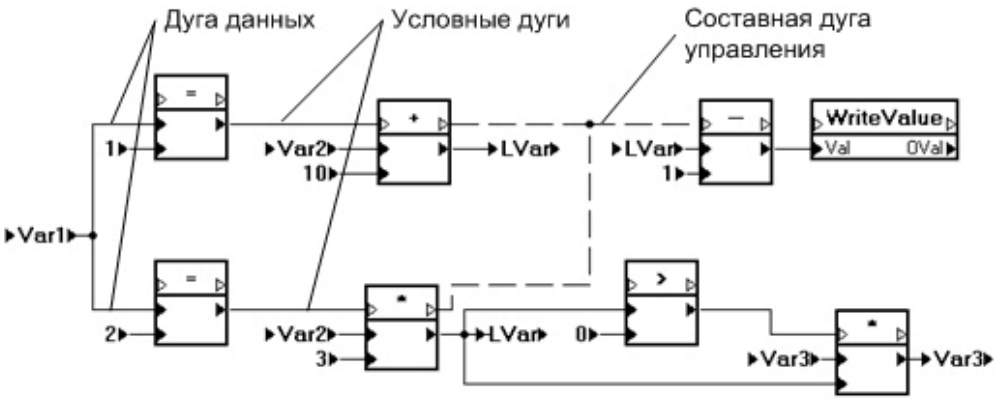


Рисунок 3 – Пример FBD-схемы на основе расширенного диалекта языка FBD

Алгоритм генерации с учетом условных фрагментов

В случае, если на схеме содержатся условные дуги или дуги управления, алгоритм определения порядка выполнения вершин схемы усложняется. Поскольку выходом генератора текстового эквивалента FBD является программа на каком-то из классических текстовых языков программирования, то схема должна однозначно представляться правильной структурой условных операторов. Основным требованием корректности FBD-схемы является необходимость того, чтобы к моменту выполнения каждой вершины схемы все ее входы были уже вычислены. Это накладывает ограничения на поток данных схемы между вершинами, которые могут выполняться в зависимости от некоторых условий. Таким образом, алгоритмическими задачами генератора FBD-схем являются:

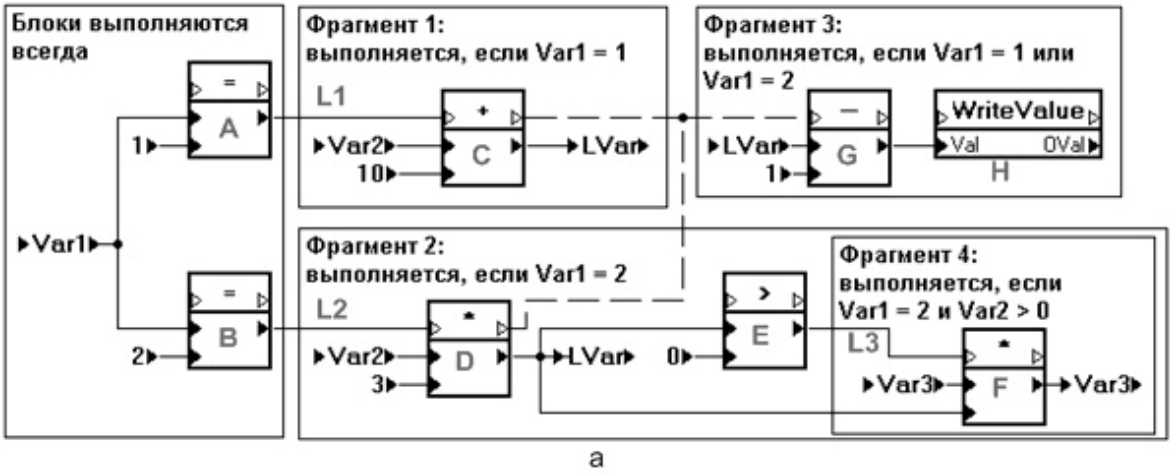
- определение необходимого порядка и условий выполнения всех вершин схемы;
- проверка корректности сочетания потока данных и потока управления.

Введем некоторые термины. **Условным фрагментом** схемы будем называть совокупность вершин схемы, которые при вызове схемы или всегда выполняются вместе, или не выполняются вообще. Две вершины схемы принадлежат к одному условному фрагменту тогда и только тогда, когда условия выполнения этих двух вершин эквивалентны. Аналогом условных фрагментов в текстовых языках является ветвь ***then*** оператора ***if.*** Кроме того, условные фрагменты могут быть вложенными. Например, условный фрагмент ***A*** должен выполняться, если истинно условие, значение которого вычисляется в условном фрагменте ***B***; тогда условный фрагмент ***A*** является вложенным в фрагмент ***B***. Таким образом, условные фрагменты схемы образуют **дерево условных фрагментов**, корнем которого является фрагмент, содержащий вершины, выполняемые всегда. Следует отметить, что стандартная схема FBD представляется деревом фрагментов, состоящим всего из одной вершины. Будем считать, что некоторый блок ***K*** схемы FBD **зависит** от блока ***P***, если ко входам блока ***K*** подключена хотя бы одна дуга, идущая от выходов блока ***P***.

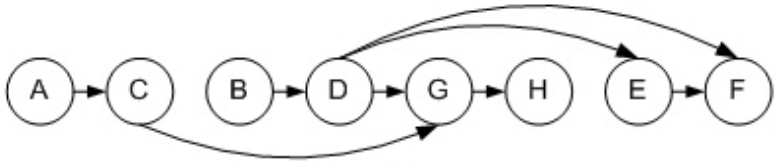
В начале работы алгоритма необходимо определить первичный порядок вершин схемы, используя топологическую сортировку. При этом нужно учитывать как дуги данных, так и дуги управления и условные дуги, поскольку все они указывают на зависимости между вершинами схемы. Будем рассматривать алгоритм разбиения схемы на условные фрагменты на примере схемы, изображенной на рис. 4,а. Топологическая сортировка дает порядок выполнения вершин, показанный на рис. 4,б. Этот порядок не образует правильное дерево условных фрагментов, поскольку, например, вершины ***D*** и ***E*** принадлежат фрагменту 2, хотя между ними выполняются вершины **G** и **H**, принадлежащие другому условному фрагменту. Поэтому генератор должен разбивать схему на дерево условных фрагментов, корректируя при этом первичный порядок выполнения вершин. Если построить такое дерево фрагментов для данной схемы невозможно, то пользователю должно выдаваться сообщение об ошибке.

Исходя из требований к схеме FBD и генерируемой программе, можно сформулировать основное требование к схеме, содержащей условные дуги: каждый блок, который принадлежит к некоторому фрагменту, может зависеть только от блоков, которые принадлежат или к тому же фрагменту, или к фрагментам, которые составляют путь от фрагмента с блоком до корня дерева фрагментов. Это требование связано с тем, что на момент истинности условия вычисления некото-

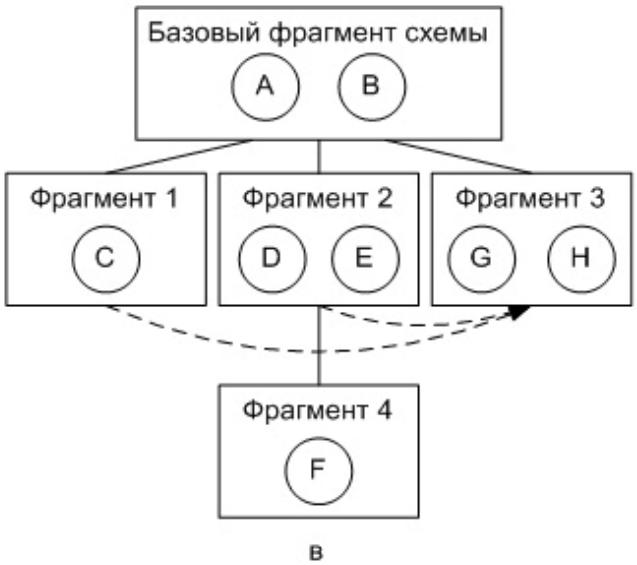
рого блока все его входы должны быть уже вычислены. Из этого следует, что если вершина **A** зависит от вершины **B**, то вершина **A** должна принадлежать либо к фрагменту, содержащему **B**, либо к фрагменту, дочернему по отношению к фрагменту, содержащему **B**. Это требование вытекает из того, что если при определенных условиях выполняется некий фрагмент **F**, то обязательно выполняются только фрагменты-родители фрагмента **F** в дереве условных фрагментов.



а



б



в

```
flag1 = 0;
A; B;
if L1
  C;
  flag1 = 1;
endif;
if L2
  D; E;
  if L3
    F;
  endif;
  flag1 = 1;
endif;
if flag1
  G; H;
endif;
```

г

Рисунок 4 – Генерация текста для схемы FBD, содержащей условные фрагменты:
а – пример схемы FBD с условными фрагментами;
б – порядок выполнения вершин без учета условных фрагментов;
в – дерево условных фрагментов;
г – текст программы, которая создается генератором

Рассмотрим алгоритм генерации текстового эквивалента схемы FBD, корректно решающий поставленную задачу. Генератор обрабатывает вершины схемы в порядке, сформированном топологической сортировкой. Это гарантирует, что на момент обработки каждого блока все блоки, от которых он зависит, уже были обработаны. По сути, первая часть алгоритма полностью совпадает с алгоритмом генерации простой FBD-схемы. Генератор по мере обработки вершин схемы строит дерево условных фрагментов. Изначально дерево фрагментов содержит только корневой фрагмент, который выполняется всегда. Для каждой вершины $\mathbf{v} \in \mathbf{V}$ осуществляется анализ тех вершин и содержащих их фрагментов, от которых вершина **v** зависит. Если вершину **v** нельзя отнести ни к одному из существующих условных фрагментов, то создается новый фрагмент. После этого осуществляется анализ зависимостей вершины **v** и, если нарушается основное требование к схеме с условными дугами, то генерируется ошибка.

Рассмотрим более детально принципы, которые реализует генератор при внесении очередной вершины **v** в дерево фрагментов. Если у вершины **v** не подключен вход управления, то, очевидно, она не образует нового условного фрагмента, а принадлежит к уже существующему. Рассмотрим условные фрагменты, содержащие вершины, от которых вершина **v** зависит. Выберем среди этих фрагментов тот, который лежит в дереве фрагментов дальше всего от корня, т.е. для выполнения которого должно удовлетворяться больше всего условий. Назовем этот фрагмент **фрагментом по данным** вершины **v**. Обозначим фрагмент по данным вершины **v** как $\mathbf{F_D}$. Этот фрагмент является первым кандидатом на формирование условного фрагмента, содержащего вершину **v**. Если к входу управления вершины **v** подходит простая дуга управления, то это лишь задает дополнительный порядок на вершинах схемы. Поэтому фрагмент, содержащий вершину, от которой к условному входу вершины **v** идет дуга, нужно учитывать, как и фрагменты, от которых вершина **v** зависит по данным.

Если же ко входу управления вершины **v** подходит условная дуга, то вершина **v** должна образовывать новый фрагмент $\mathbf{F_u}$. Однако условие выполнения вершины **v** также вычисляется в одном из условных фрагментов, поэтому фрагмент $\mathbf{F_u}$ должен быть дочерним по отношению к $\mathbf{F_D}$ в дереве фрагментов.

Если ко входу управления вершины **v** подходит составная дуга управления, то вершина **v** должна выполняться, если выполняется хотя бы одна из вершин, от которых к **v** идет дуга управления. Пусть множество фрагментов $\mathbf{M_C}$ содержит фрагменты, которым принадлежат вершины, соединенные с **v** составной дугой управления. Обозначим через $\mathbf{F_P}$ общего родителя в дереве фрагментов $\mathbf{F} \in \mathbf{M_C}$, т.е. такой фрагмент из дерева фрагментов, который является родителем всех фрагментов из $\mathbf{M_C}$ и лежит как можно дальше от корня дерева фрагментов. Тогда, исходя из основного правила генератора, можно сделать вывод, что вершина **v** должна образовывать новый фрагмент $\mathbf{F_C}$, дочерний по отношению к $\mathbf{F_P}$. При этом условие выполнения нового фрагмента $\mathbf{F_C}$ должно быть дизъюнкцией условий выполнения всех фрагментов $\mathbf{F} \in \mathbf{M_C}$. Кроме того, при генерации текстового эквивалента генератор должен обеспечивать такой порядок появления в тексте тел фрагментов, дочерних к $\mathbf{F_P}$, чтобы тело фрагмента $\mathbf{F_C}$ располагалось позже всех фрагментов $\mathbf{F} \in \mathbf{M_C}$.

После того, как для вершины **v** генератор выбрал или создал новый фрагмент, он проверяет выполнение основного условия для вершины **v**. Если оно не выполняется, то генерируется сообщение об ошибке. Таким образом, после пер-

вого этапа генерации окончательно построено дерево условных фрагментов, определен порядок выполнения вершин и их распределение по фрагментам. Для рассматриваемого примера дерево условных фрагментов показано на рис. 4,в.

Принципы генерации текстовых эквивалентов FBD-схем

После определения порядка выполнения вершин схемы FBD и распределения их по условным фрагментам генератор должен сформировать эквивалент схемы на одном из текстовых языков высокого уровня. В качестве примера будем использовать язык Си.

Текстовый эквивалент схемы FBD представляет собой правильную вложенную структуру условных блоков, полностью соответствующую дереву условных фрагментов. Сформировать такую структуру можно обходом в глубину сформированного дерева условных фрагментов. Для фрагментов, образованных условными дугами схемы, условие выполнения должно содержать значение соответствующей дуги. Для фрагментов, образованных составной дугой управления, условие должно быть дизъюнкцией условий выполнения фрагментов, от которых данный фрагмент зависит. Однако в этом случае возможны два подхода: или прямое объединение условий выполнения операцией «ИЛИ», или же реализация этой операции через промежуточную переменную логического типа. Второй подход экономит процессорное время с крайне малыми затратами памяти, поэтому является более предпочтительным для реализации в генераторе.

В теле каждого условного блока генератор создает текстовые эквиваленты всех вершин-действий схемы (операторов и блоков), которые входят в соответствующий условный фрагмент. При этом вершины внутри условного блока генерируются в порядке, сформированном на начальной стадии генерации топологической сортировкой. В конце каждого условного блока генератор создает реализации условных фрагментов, вложенных в данный. Для рассматриваемого примера общая структура текстового эквивалента показана на рис. 4,г.

Рассмотрим принцип построения генератором текстовых эквивалентов для каждой вершины схемы. Поскольку дуги показывают передачу данных между вершинами схемы, то в текстовом эквиваленте каждая дуга данных схемы представляется локальной переменной процедуры. Тогда каждая вершина-действие схемы представляется оператором языка Си, которая берет значения операндов из переменных или параметров охватывающей процедуры и возвращает результат в переменную или параметр. Каждая процедура или функция генерируется в соответствующий вызов:

$L_0 = F(L_1,L_2,\ldots,L_n);$ (функция)
$F(L_1,L_2,\ldots,L_n);$ (процедура)

Оператор генерируется в присваивание или несколько присваиваний:

$L_0 = L_1 \otimes L_2 \otimes \ldots \otimes L_n;$ (оператор $\otimes$)

Кроме того, на схеме могут встречаться дуги, соединяющие несколько переменных. В таком случае также генерируется присваивание:

$X = L_1;$ (переменная X)

Глобальные переменные должны определяться в некотором дополнительном файле, структура которого зависит от реализации. Кроме того, в соответствии

со стандартом [2] для блоков можно определить собственные данные, т.е. локальные переменные, значения которых будут сохраняться между вызовами для каждого из установленных экземпляров блока. Для реализации собственных данных удобно использовать классы. Однако, поскольку язык Си не поддерживает ООП, а трансляторы C++ отсутствуют для многих аппаратных платформ, особенно с ограниченными ресурсами, будем использовать другой подход, который по сути является ручной реализацией объектов. Так, для всех блоков, содержащих собственные данные, определяется структура, содержащая в качестве полей собственные данные этого блока и структуры с собственными данными вложенных блоков. Кроме того, для каждого такого блока добавляется параметр **self** типа указателя на структуру собственных данных. Доступ к собственным данным осуществляется через этот параметр.

Следует отметить, что для обеспечения уникальности имен в пределах текстовой программы необходимо добавлять к именам, создаваемым генератором, префикс, например «_». Имена локальных переменных, соответствующих дугам, можно строить из идентификаторов вершин и номеров выходов блоков.

Исходя из вышесказанного, для данного примера генератор может создавать следующий код на языке Си:

Файл block1.h:

```
#ifndef _INC_BLOCK1
#define _INC_BLOCK1
#include "WriteValue.h"              // используемый блок
typedef struct {                    //  структура собственных данных
      _data_WriteValue _WriteValue_17;    // вложенный блок
      unsigned char _Var3;    // переменная, сохраняемая между вызовами
} _data_block1;
void _block1 (STORED_AREA _data_block1* const self,
  const unsigned char _INPUT1, OUT_AREA unsigned char* _OUTPUT1);
#endif
```

Файл block1.c:

```
#include "block1.h"    // подключение заголовочного файла блока
void _block1(STORED_AREA _data_block1* const self,
  const unsigned char _INPUT1, OUT_AREA unsigned char* _OUTPUT1)
{
      unsigned char _LVar = 0;  // локальные переменные
      bit _run_flag_1 = FALSE;  // признак выполнения условного фрагмента
      bool _3_oper_out1_;       // локальные переменные для дуг схемы
      bool _4_oper_out1_;
      _3_oper_out1_ = _global_Var1 == 1;
      _4_oper_out1_ = _global_Var1 == 2;
      if (_3_oper_out1_)
      {
            unsigned char _8_oper_out1_;
            _run_flag_1 = TRUE;
            _8_oper_out1_ = _global_Var2 + 10;
            _LVar = _8_oper_out1_;
      }
      if (_4_oper_out1_)
      {
            unsigned char _10_oper_out1_;
            bool _27_oper_out1_;
            _run_flag_1 = TRUE;
            _10_oper_out1_ = _global_Var2 * 3;
```

```
                _LVar = _10_oper_out1_;
                _27_oper_out1_ = _10_oper_out1_ > 0;
                if (_27_oper_out1_)
                {
                      unsigned char _24_oper_out1_;
                      __fbdvertex(block1.fbd,24);
                      _24_oper_out1_ = self->_Var3 * _10_oper_out1_;
                      self->_Var3 = _24_oper_out1_;
                }
          }
          if (_run_flag_1)
          {
                unsigned char _15_oper_out1_;
                unsigned char _17_WriteValue_out1_OVal;
                _15_oper_out1_ = _LVar - 1;
                _WriteValue(&self->_WriteValue_17, _15_oper_out1_,
&_17_WriteValue_out1_OVal);
          }
     }
```

Следует отметить, что приведенный алгоритм генерирует в тексте программы много лишних операторов присваивания, количество которых можно оптимизировать. Например, для вычисления выражения x = sin(4*a+b/2)+2 генератор создаст четыре промежуточные переменные, хотя можно сгенерировать вычисление выражения одним оператором. Однако, поскольку генерация осуществляется на языке высокого уровня, компилятор Си оптимизирует эти переменные, используя вместо них регистры. Более того, архитектура большинства компилятором такова, что они строят на основе исходной программы промежуточное представление, в котором каждая команда реализует одну элементарную арифметическую операцию [4]. Поэтому оптимизировать в алгоритме генерации количество присваиваний в текстовом эквиваленте бессмысленно. Такая оптимизация будет увеличивать время работы как генератора, так и компилятора на приведение сложных выражений к внутреннему представлению.

Таким образом, предложенный алгоритм генерации корректно создает текстовый эквивалент схемы FBD расширенного диалекта, осуществляя правильное определение порядка и условий выполнения вершин схемы и может использоваться при разработке трансляторов в системах визуального проектирования прикладного ПО.

Список литературы

1. Патрахин В.А. Средства программирования PC-совместимых контроллеров // ПиКАД. 2003. № 1-2. – С. 34 - 40.
2. International standard 1131-3, Part 3: Programming languages, IEC, Division Automatismes Programmables, First edition, 1993.
3. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. М.: МЦНМО, 2001. – 960 с.
4. Ахо Альфред В., Сети Рави, Ульман Джеффри Д. Компиляторы: принципы, технологии и инструменты: Пер. с англ.– М.: Изд. дом «Вильямс», 2003.–768 с.