УДК 681.3

С. В. Мухов, Г. Л. Муравьев

КОМПЬЮТЕРНАЯ ПОДДЕРЖКА СПЕЦИАЛИЗИРОВАННЫХ СТРУКТУР ДАННЫХ

Сформулированы требования к компьютерной поддержке специфичных классов объектов обработки и их типов, используемых при моделировании описаний цифровых устройств. Рассмотрены структуры данных, обеспечивающие компактность хранения и эффективность обработки указанных объектов применительно к языку VHDL

Введение

Рост сложности проектов цифровых устройств привел к переносу акцента в их моделировании на верхний, микросхемный уровень проектирования. Это снижает размерность задач, ослабляет требования на более детальных и трудоемких уровнях проектирования. Моделирование систем, разрабатываемых с использованием языков спецификации верхнего уровня, базируется на использовании специфичных объектов обработки. Это сигналы, для представления которых применяют специфические типы данных. Это многозначная логика, битовые строки произвольной длины. Последний тип задает векторное представление сигналов, являющееся одним из средств повышения компактности описаний, скорости моделирования.

Система моделирования в составе САПР верхнего уровня реализуется как взаимодействие подсистем типа: БИБЛИОТЕКА для информационного обеспечения моделирования, хранения формирователей; ГЕНЕРАЦИЯ для получения результативных моделей, настроенных на библиотеку формирователей; МОДЕЛИРОВАНИЕ для анализа изменения сигналов на выходах формирователей или на входах портов, ведущих к срабатыванию процессов, продвижения модельного времени, обработки событий, обновления формирователей; АРИФМЕТИКА для поддержки в виде специализированных библиотек обработки специфических типов данных.

Эффективность моделирования значительно определяется организацией используемых структур данных. Здесь рассматривается подход к организации информационного обеспечения моделирования, организации формирователей сигналов и обработки специфических типов применительно к языку VHDL (стандарты VHDL'93 - ANSI/IEEE Std 1076–1993 и VHDL–AMS – Std 1076.1–1999 [1, 2]).

Особенности описания формирователей

В VHDL сигналы относятся к категории объектов данных. Это внешние сигналы, декларированные путем объявления портов проекта и обеспечивающие каналы динамической связи проекта с окружающей средой. И внутренние сигналы, декларированные в телеархитектуры проекта.

С каждым сигналом связан формирователь, являющийся источником его значений и содержащий упорядоченные во времени списки прошлых и будущих значений. Формирователи составляют информационную базу для осуществления моделирования проекта.

Указанное требует организации хранения формирователей, направленной на их эффективное использование в процессе моделирования, включая сканирование содержимого, применение операций атрибутивной арифметики, выбор нужных элементов, внесение изменений. Организация формирователей должна производиться с учетом их привязки к проектам, портам, версиям архитектурной реализации, с учетом способа представления входных воздействий и типов VHDL [3- 5].

Соответственно БИБЛИОТЕКА информационного обеспечения моделирования включает следующие разделы:

- Логическое оглавление верхнего уровня с ключевым полем – описанием наборов данных проекта (НД). Представляет список имен проектов, соответствующих им имен описаний структур наборов данных (СНД) и списков имен самих наборов данных. Идентификация НД осуществляется с использованием префиксации и обеспечивает иерархическое упорядочивание объектов.
- Библиотеку описаний СНД для статичного (СНДС) и динамического способов представления значений сигналов (СНДД).
- Библиотеку НД для статичного (НДС) и динамического способов представления значений сигналов – формирователей (НДД).
- Библиотеку выходных реакций, которая строится по аналогии с библиотекой НДС.

Связывание с заданным на входе вектором обеспечивается заданием его идентификатора в виде номера в соответствующем префиксе. Значения формирователей (в виде соответствующих векторов в НД) описывают входные, выходные значения моделируемых проектов. Используется разный синтаксис описания и различная интерпретация при моделировании. Примерные форматы в форме Бэкуса – Наура приведены ниже

```
<Набор> ::= <Заголовок> <Вектор> { < Вектор> } ,
<Заголовок> ::= <Идентификатор> <Тип_набора> <Число_векторов>
                <Размер_вектора> <Сигнал> { <Сигнал> } <Погрешность> ,
<Сигнал> ::= <Имя> <Режим_использования> <Тип> ,
<Погрешность> ::= < Тип> <Значение> { <Значение> }.
```

Для статических последовательностей

```
<Вектор> ::= <Номер> <Значение> { <Значение> } .
```

Для динамических последовательностей в целях удобства обработки формат хранения векторов приводится к формату статичного варианта. Диаграммы сигналов заменяют множеством векторов, каждый из которых соответствует временному срезу – точке, где происходит изменение хотя бы одного сигнала. Соответственно значение времени вносится в вектор

```
<Вектор> ::= <Имя_сигнала> <Режим_использования> <Тип>
             <Размерность_времени> <Компонент> { <Компонент> } ,
<Компонент> ::= <Значение_сигнала> <Время> .
```

Особенности описания типов

Специфический тип данных языка VHDL - *bit_vector* (строка бит), составляющий вместе с типами *bit* (бит) и *boolean* (логический) основу предопределенной двоичной арифметики языка. При обработке сигналов этого типа и их сечений должна обеспечиваться эффективная поддержка типового набора операций регистрового уровня [1–4].

Соответственно задача сводится к выбору внутреннего представления объектов этого типа, использование которого должно обеспечивать: – реализацию произвольных выражений из комбинаций битовых векторов, битовых и литеральных значений; – корректное взаимодействие с объектами других типов, эффективное преобразование типов; - минимальное по времени выполнение основных операций двоичной арифметики за счет использования машинных команд и минимизации числа преобразований типов.

Кроме этого, необходимо обеспечить: – возможность контроля результатов выполнения операций, изменения диапазона значения в результате операций, контроля ситуации потери значимости и переполнения промежуточных и конечного результатов; – поддержку аппарата предопределенных атрибутов VHDL; – адаптацию логического формата хранения под длину конкретной битовой строки.

Структуры данных библиотеки формирователей

Оглавление библиотеки состоит из описательных структур:

```
struct      def_dir
{
  char dir_sw; /* флаги */
    char[dir_prjl] dir_prj;  /* идентификатор проекта */
    char[dir_textl] dir_text;  /* комментарий */
    char[dir_mtdsl] dir_mtds; /* маска */
    char[dir_dtdsl] dir_dtds;  /* идентификатор НД */
}   .
```

Здесь поле *dir_sw* – флаг записи (0x80 – последний элемент набора; 0x40 – элемент свободен; 0x20 – проект содержит НДС; 0x10 – проект содержит НДД), *dir_prj* - имя проекта, *dir_text* – поле комментария, *dir_mtds* – имя Маски_НДх, *dtds* – имя Описания_НДх.

Описание_НД содержит структуры следующего вида:

```
struct def_dtds
{
  char dtds_sw;       /* флаги */
    char dtds_swx;    /* флаги */
    char dtds_swz;    /* флаги */
```

```
    char dtds_swe;     /* флаги */
    int dtds_error;    /* погрешность */
    char[dir_mtdsl] dtds_mtds; /* маска */
    int dtds_nmbx;     /* количество сигналов */
    int dtds_recl;   /* длина записи вектора */
    int dtds_prfl;   /* длина префикса вектора */
    struct def_sig  dtds_signal [dtds_nmb]; /* описания сигналов */
}    .
```

Здесь поле *dtds_sw* – флаг записи (0x80 – последний элемент набора; 0x40 – элемент свободен; 0x08 – описание НДС; 0x04 – описание НДД; 0x02 – номер такта; 0x01 – время типа *real*). Поле *dtds_swx* – флаг записи (0x80 – нет выходных реакций), *dtds_swe* – флаг (0x80 – общая погрешность; 0x40 – число разрядов погрешности; 0x20 – абсолютное значение погрешности; 0x10 – относительное значение погрешности; 0x08 – формат *int* погрешности; 0x04 – формат *float* погрешности; 0x02 – погрешность по каждому сигналу). Поле *dtds_error* – значение погрешности, *dtds_mtds* – имя Маски_НДх, *dtds_nmbx* – количество сигналов, *dtds_recl* – длина вектора, *dtds_prfl* – длина префикса вектора, *dtds_signal* – массив описаний сигналов.

Описание сигналов выполняется с помощью следующей структуры:

```
struct def_sig
{
  char sig_sw;    /* флаги */
    char sig_swf;    /* флаги */
    char sig_swtv;     /* флаги */
    char sig_swe;   /* флаги */
    char[sig_idtxtl] sig_idtxt; /* идентификатор сигнала */
    int sig_value;   /* значение сигнала */
    int sig_error;     /* значение погрешности */
}    .
```

Здесь поле *sig_sw* – флаг записи (0x80 – последний элемент набора; 0x40 - элемент свободен; 0x20, 0x10, 0x08 режим сигнала IN, OUT, INOUT; 0x04 - снятие выходных реакций). Поле *sig_swf*, *sig_swtv* – флаги формата внутреннего и экранного представления данных, *sig_swe* – флаг погрешности (см. *dtds_swe*), *sig_idtxt* – идентификатор сигнала, *sig_value* – значение сигнала, *sig_error* – значение погрешности (см. *dtds_swe*).

При необходимости преобразования данных во время вызова для моделирующей программы может быть определена структура, аналогичная *def_dtds*, но с замененными полями, что позволит с минимальными затратами обеспечивать их преобразование.

Реализация битовых структур

Учитывая соображения в пользу независимости специализированных библиотек, в качестве средства хранения значений переменных типа *bit_vector* произвольной длины предлагаются специальные структуры данных языка С. Они включают массив элементов базового типа (базовые элементы) и служебную структуру. Такими элементами могут служить целочисленные слова. Массив элементов $A[0], A[1],..., A[AL-1]$ базового типа используется для хранения значений *bit_vector* в упакованном формате, что обеспечивает необходимую эффективность реализации. А служебная структура, описывающая область и формат хранения данных, обеспечивает их мобильность, самоопределение и реализацию пересылок путем простого изменения описательных структур.

Биты элемента A[0] могут быть использованы частично, поэтому в служебной структуре хранится значение длины неиспользованной области, задающее смещение от начала поля данных до начала битовой последовательности и указывающее начальную позицию размещения битового вектора. Место хранения переменной запоминается указателем начала данных.

Таким образом, для отображения переменной типа *bit_vector* произвольной длины необходим: массив базовых элементов для хранения значений; структура (описание строки бит) для хранения служебной информации

```
#define   define_bit_vector  dbv
#define   bit_vector              dbv
struct dbv                              /* описание строки бит */
{
     unsigned char dbv_t;               /* тип данных */
     unsigned char dbv_s;               /* текущее состояние данных */
```

```
union dbv_b                          /* база данных */
{
     char *p;                        /*  указатель начала данных */
     unsigned int       w;           /* строка бит dbv_tp */
     unsigned long      d;           /* строка бит dbv_tp */
     unsigned char  c[4];            /* строка бит dbv_tp */
} ;
unsigned int  dbv_d;                 /* смещение данных  */
unsigned int  dbv_l;                 /* длина данных */
} ;    .
```

Здесь атрибут *dbv_b* (типа объединение) служит для указания на начало поля, предназначенного для размещения данных, т. е. собственно самой битовой последовательности. Поле *dbv_d* доопределяет указание места расположения битовой последовательности. Поле *dbv_l* хранит количество разрядов, используемых в битовой последовательности.

Поле *dbv_t* указывает вариант реализации строки бит, т. е. используемый для хранения языковой тип данных, например: символ, логический, слово, двойное слово, строка байт, массив базовых элементов. Возможные варианты описания поля *dbv_s*: некорректные данные; отрицательные данные в прямом коде; переполнение; использование дополнительного кода; без выравнивания на правую границу.

Нумерация битов выполняется с нуля, а битовая строка (подстрока) выравнивается по правому краю на границу байта с целью эффективного перехода на использование машинных операций. Соответственно при применении используются указатели на структуры *dbv* и при обработке данных типа *bit_vector* в качестве параметров пересылаются адреса соответствующих служебных структур.

Самоопределение данных, их совместимость с другими типами данных обеспечивается полем *dbv_t*. При этом формат хранения конкретной битовой строки адаптируется под ее длину, позволяя в ряде случаев использовать просто базовые элементы языка и связанный с ними набор операций. Индикация текущего (динамического) состояния данных, указывающая, в частности, на их корректность, обеспечивается полем *dbv_s*.

Заключение

В работе применительно к языку VHDL сформулированы требования к реализации компьютерной поддержки специфических объектов обработки и связанных с ними формирователей и типов данных. Рассмотрены структуры данных для хранения и обработки формирователей, битовых векторов, используемых при моделировании.

Литература

1. *Поляков, А. К.* Языки VHDL и VERILOG в проектировании цифровой аппаратуры / А. К. Поляков. – М. : СОЛОН-Пресс, 2003. – 320 с.

2. *Бибило, П. Н.* Основы языка VHDL / П. Н. Бибило. – М. : СОЛОН-Р, 2000. – 210 с.

3. *Прихожий, А. А.* Система автоматизированного проектирования СБИС. Процедуры проектирования / А. А. Прихожий, Г. Л. Муравьев. – Минск, 1992. - 98 с. – (Материалы по математическому обеспечению ЭВМ / Академия наук Беларуси, Ин-т техн. кибернетики).

4. *Дудкин, А. А.* Алгоритмы и подсистемы автоматизированного логического проектирования цифровых СБИС / А. А. Дудкин, В. А. Головко, Г. Л. Муравьев. – Минск, 1994. – 120 с. - (Материалы по математическому обеспечению ЭВМ / Академия наук Беларуси, Ин-т техн. кибернетики).

5. *Муравьев, Г. Л.* Интерпретация VHDL-описаний, согласованная с процессным способом моделирования // Вестник БГТУ. – 2006. – № 5. - С. 39–42.

Мухов Сергей Владимирович, доцент кафедры информатики и вычислительной техники Брестского государственного технического университета, кандидат технических наук, доцент, tcm-brest@mail.ru

Муравьев Геннадий Леонидович, профессор кафедры интеллектуальных информационных технологий Брестского государственного технического университета, кандидат технических наук, доцент, mgl0251@mail.ru