УДК 681.3

Мухов С.В., Муравьев Г.Л.

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ БИТОВЫХ СТРУКТУР ПРИ МОДЕЛИРОВАНИИ VHDL-ПРОЕКТОВ

Введение

При проектировании цифровых систем необходимы эффективные средства реализации их моделей. Наиболее существенными элементами таких объектов моделирования, отображаемыми, как правило, при структурном, потоковом, поведенческом описаниях моделей проектов, являются сигналы и операционные средства для их обработки. Это видно на примере проектов, разрабатываемых с использованием специализированных языков спецификации верхнего уровня, таких как VERILOG-HDL, VHDL, языков регистровых передач и т.д., где для представления сигналов используются объекты данных специфических типов. Это многозначная логика (в языке VHDL типы std_logic, std_ulogic, подтип X01Z и т.п.), биты, битовые строки произвольной длины как векторное представление сигналов.

Векторное представление сигналов и операций над ними является одним из средств повышения компактности описаний моделей проектов, скорости моделирования. Соответственно эффективность моделирования ряда практических приложений определяется степенью обеспеченности функциональной полноты и скоростью обработки данных, представленных в виде битовых строк произвольной длины.

Одним из решений в плане обеспечения эффективной реализации такой обработки является создание специализированных библиотек. На примере языка проектирования VHDL (стандарты VHDL'93 - ANSI/IEEE Std 1076-1993 и VHDL-AMS - Std 1076.1-1999) [1-4] видно, что должна обеспечиваться эффективная поддержка операций над объектами типа bit_vector (СТРОКА_БИТ), составляющими вместе с типами bit (БИТ) и boolean (ЛОГИЧЕСКИЙ) основу предопределенной двоичной арифметики языка.

В работе анализируются требования к реализации компьютерной поддержки обработки данных указанных типов. Рассматриваются аспекты реализации обработки битовых векторов и связанных с ними типов данных средствами языков программирования высокого уровня. Все рассматриваемые вопросы иллюстрируются применительно к языку VHDL.

Требования к реализации битовых структур

В языке VHDL тип данных bit_vector относится к предопределенным индексированным типам, его значениями являются одномерные массивы типа bit, индексированные значениями типа natural:

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

Переменные типа bit_vector с границами, определяемыми значениями индексов i1, i2, описываются как

```
ИмяПеременной: bit_vector [i1 to i2] ,
ИмяПеременной: bit_vector [i1 downto i2]
```

Над переменными этого типа допустимы операции: а) логические операции and, or, nand, nor, xor, not; б) операции отношения =, /=, <, <=, >, >= с результатом типа boolean; в) аддитивная операция конкатенации, результатом применения которой является bit_vector суммарной длины, состоящий из элементов левого операнда, за которым следуют элементы правого операнда; г) операция вырезки (slicing) как получение непрерывного подмножества (сечения) переменной типа bit_vector из последовательности рядом расположенных битов; д) операция присваивания, в левой части которой может использоваться как переменная типа bit_vector так и любое ее сечение.

Остальные операции VHDL, допустимые для числовых типов, в том числе арифметические, не являются предопределенными для указанных типов и могут быть реализованы самостоятельно пользователем с помощью аппарата процедур и функций VHDL. Этот подход универсален, но может повлиять на эффективность моделирования при неоптимальных алгоритмах имитации отсутствующих операций.

Задача сводится к выбору внутреннего представления объектов типа СТРОКА_БИТ, обеспечивающего их корректное по синтаксису использование во всех допустимых конструкциях языка VHDL, обеспечивающего работу с описаниями моделей вплоть до регистрового уровня детализации и быструю обработку. При этом состав предопределенных операций должен быть расширен до типового набора операций регистрового уровня (например, арифметических операций, циклического сложения, присваивания по маске, сдвига и циклического сдвига) [5-8].

Соответственно реализация внутреннего представления переменных типа bit_vector должна обеспечивать [9]: - корректное взаимодействие указанных объектов с объектами других типов (bit, boolean, integer, array of bit, array of boolean) и эффективное преобразование объектов одних типов в другие; - минимальное по времени выполнение основных операций двоичной арифметики за счет возможности максимально использовать для выполнения операций соответствующих машинных (арифметических и др.) команд процессора, а также за счет минимального числа преобразований типов; - реализацию формул с использованием смесей битовых векторов произвольной длины, битовых и литеральных значений (с приведением термов формул и результатов выполнения операций в формулах к одному и тому же типу); - возможность контроля результатов выполнения операций, контроль изменения диапазона значения в результате операций, контроль ситуации потери значимости, переполнения промежуточных и конечного результата; - поддержку аппарата предопределенных атрибутов VHDL для сигналов, переменных индексированного типа; - адаптацию формата хранения под длину конкретной битовой строки.

Подходы к реализации битовых структур

В связи с необходимостью обеспечения эффективной обработки в качестве средства реализации рассматривается язык программирования С. Возможна реализация битовых структур с использованием средств библиотеки шаблонов С++. Они автоматизируют, в частности, выполнение только классических, логических операций над битовыми строками, удобны для работы с объектами, но в меньшей мере ориентированы на достижение их эффективной обработки.

Учитывая также соображения в пользу независимости специализированных библиотек от библиотек фирменных, в качестве средства хранения значений переменных битовой структуры произвольной длины предлагаются специализированные структуры данных.

Муравьёв Геннадий Леонидович, *к.т.н., профессор кафедры интеллектуальных информационных технологий БрГТУ.*
Мухов Сергей Владимирович, *доцент кафедры интеллектуальных информационных технологий БрГТУ.*
Беларусь, Брестский государственный технический университет, 224017, Беларусь, г. Брест, ул. Московская, 267.

Они включают массив элементов базового типа языка высокого уровня (далее просто – базовые элементы) и служебную структуру. Массив используется для хранения значений переменной, что обеспечивает необходимую эффективность реализации. А служебная структура, описывающая область (адрес и смещение) и формат хранения данных, другую служебную информацию, обеспечивает мобильность, самоопределение данных и реализацию пересылок данных путем простого изменения описательных структур.

При этом в качестве формата хранения значений переменной рассматривается упакованный формат, где для представления значения используется один бит в массиве элементов базового типа языка С (такими элементами могут служить целочисленные слова длиной 8, 16, 32 бита). Тогда длина переменной типа bit_vector в элементах базового типа определяется как AL = Al / Tl при делении нацело и как AL = | Al / Tl | + 1 в остальных случаях.

Здесь Tl - длина базового элемента в битах, а Al = (| i1 - i2 | + 1) - длина информационной части переменной типа bit_vector в битах. Формат хранения значения переменной типа bit_vector: массив элементов базового типа A[0], A[1],…,A[AL-1].

Биты элемента A[0] могут быть использованы частично, поэтому в служебной структуре хранится значение длины неиспользованной области Ad, выполняющее роль смещения от начала поля данных до начала битовой последовательности и указывающее начальную позицию размещения битового вектора. Соответственно AL x Tl = Al + Ad . Место хранения переменной запоминается значением Ap, выполняющим роль указателя на начало данных (поля данных).

Таким образом, для отображения переменной типа СТРОКА_БИТ произвольной длины необходим: - массив базовых элементов для хранения значений переменной; - структура ОПИСАНИЕ_СТРОКИ_БИТ для хранения всей служебной информации (Ap, Ad, Al и т.п.) о месте, характере размещения переменной и ее свойствах. Указанная структура может быть описана как

```
#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;   /* Ap - указатель на начало 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; /* Ad - СМЕЩЕНИЕ_ДАННЫХ */
     unsigned int dbv_l;       /* Al - ДЛИНА_ДАННЫХ */
     } ; .
```

Поле БАЗА_ДАННЫХ служит для указания на начало поля, предназначенного для размещения данных, т.е. собственно самой битовой последовательности. Поле СМЕЩЕНИЕ_ДАННЫХ доопределяет указание места расположения битовой последовательности. Поле ДЛИНА_ДАННЫХ хранит количество разрядов, использованных в битовой последовательности. Соответственно общая длина поля данных определяется как (СМЕЩЕНИЕ_ДАННЫХ + ДЛИНА_ДАННЫХ)/16, где (СМЕЩЕНИЕ_ДАННЫХ + ДЛИНА_ДАННЫХ) MOD 16 = 0.

Нумерация битов начинается с нуля, битовая строка (подстрока) выравнивается по правому краю на границу байта (выравнивание выполняется с целью эффективного перехода на использование операций процессора). Имеет место ориентация на конкретную машинную архитектуру (так, например, для машин типа DEC характерно использование обратного порядка нумерации бит при реализации арифметических операций, что приводит к необходимости переопределения типов данных).

Соответственно при применении переменных типа СТРОКА_БИТ используется указатель на структуру define_bit_vector (ОПИСАНИЕ_СТРОКИ_БИТ), а при обработке данных в качестве параметров пересылаются только адреса служебных структур, т.е. при вызове подпрограмм следует передавать (соответственно возвращать) адрес(а) этой структуры.

Сама обработка предполагает: - описание базовых типов; - объявление переменных типа СТРОКА_БИТ и инициализацию их служебных структур; - инициализацию значений переменных; - описание формул.

Ниже иллюстрируется: описание переменной типа СТРОКА_БИТ (выделение памяти для информационной части переменной битовой структуры)

```
unsigned int _ИмяПеременной [ AL ];
```

инициализация структуры ОПИСАНИЕ_СТРОКИ_БИТ

```
struct define_bit_vector ИмяПеременной = { <Тип>, 0, NULL, Ad, Al };
ИмяПеременной.dbv_b.p = _ИмяПеременной;
```

определение указателя на структуру ОПИСАНИЕ_СТРОКИ_БИТ

```
struct define_bit_vector *ИмяПеременной_;
```

инициализация указателя на структуру ОПИСАНИЕ_СТРОКИ_БИТ

```
ИмяПеременной_ = &ИмяПеременной; .
```

Возможный вариант описания поля ТИП_ДАННЫХ и поля ТЕКУЩЕЕ_СОСТОЯНИЕ_ДАННЫХ структуры ОПИСАНИЕ_СТРОКИ_БИТ приведен ниже

Здесь для хранения битовой строки используется поле указателя на строку бит – байт dbv_b.c [0] (последний разряд), dbv_b.w, dbv_b.d, dbv_b.c (4 байта), если СТРОКА_БИТ реализуется соответственно как тип данных ЛОГИЧЕСКИЙ, СЛОВО, ДВОЙНОЕ_СЛОВО или СИМВОЛ.

```
/* --- dbv_type ТИП_ДАННЫХ ----------- */
#define dbv_tb 0x80       /* СТРОКА_БИТ реализуется как тип данных ЛОГИЧЕСКИЙ */
#define dbv_tbv 0x40      /* СТРОКА_БИТ реализуется как массив базовых элементов */
#define dbv_tw 0x20       /* СТРОКА_БИТ реализуется как СЛОВО */
#define dbv_td 0x10       /* СТРОКА_БИТ реализуется как ДВОЙНОЕ_СЛОВО */
#define dbv_tc 0x08       /* СТРОКА_БИТ реализуется как СИМВОЛ */
#define dbv_ts 0x04       /* СТРОКА_БИТ реализуется как СТРОКА_БАЙТ */
#define dbv_tto 0x02      /* используется обратное направление индексации (downto) */

/* ----- dbv_state ТЕКУЩЕЕ_СОСТОЯНИЕ ---------- */
#define dbv_snok 0x80 /* некорректные данные */
#define dbv_smin 0x40 /* отрицательные данные в прямом коде */
#define dbv_sover 0x20 /* переполнение */
```

Самоопределение данных и их совместимость с другими типами данных (например, boolean) обеспечивается полем ТИП_ДАННЫХ структуры ОПИСАНИЕ_СТРОКИ_БИТ. 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 корректность данных, обеспечивается наличием поля ТЕКУЩЕЕ_СОСТОЯНИЕ_ДАН-НЫХ.

Внутреннее представление учитывает возможность применения битовых переменных в арифметических операциях, обеспечивая их эффективное преобразование в арифметические типы данных. Технологически обработка данных строится на использовании средств (подпрограмм), обеспечивающих преобразование типов, в качестве поля ответа используется рабочий буфер, который сбрасывается по завершении операции.

При использовании данных типа boolean необходимо учитывать, что при обработке он совместим с типом bit_vector (длиной один бит) в смысле возможности работы с ним как со строкой бит (при наличии соответствующего её описания). А для размещения данных этого типа требуется один байт (например, значение true кодируется как 00000001B, false - 00000000B).

Кроме этого при организации обработки значений переменных типа bit_vector необходимо также учитывать:

- если данные bit_vector представимы в прямом коде, то соответственно операции сравнения определяются знаком значения, а арифметические операции строятся как последовательности операций сравнения, сложения, вычитания;
- обработка данных производится только порциями базовых элементов;
- при согласовании данных типа bit_vector с данными типа bit и boolean используется их совместимость по внутреннему представлению данных;
- при представлении индексов битовых векторов в моделирующей программе в качестве их значений используются значения из описания на языке VHDL без каких-либо преобразований;
- при реализации операции вырезки в качестве параметров используются адрес структуры ОПИСАНИЕ_СТРОКИ_БИТ, а также номера первой и последней позиций битовой структуры, определяющие позиции вырезки;
- использование в описании переменной типа bit_vector указания направления нумерации бит приводит к формированию флага dbv_tto, обработка значений такой переменной производится справа налево в последовательности A[n], A[n-1], ...

Заключение

В работе применительно к языку VHDL сформулированы требования к реализации компьютерной поддержки обработки битовых векторов и связанных с ними типов данных, используемых в моделировании цифровой аппаратуры. Рассмотрен подход к реализации их обработки средствами языков программирования высокого уровня на базе использования специальных битовых структур.

Указанный подход реализован в виде библиотечных функций, которые могут использоваться автономно (например, для моделирования микропрограмм) либо совместно с препроцессором, обеспечивающим генерацию вызовов функций по исходному описанию модели на языке VHDL [10].

СПИСОК ЦИТИРОВАННЫХ ИСТОЧНИКОВ

1. Бибило П.Н. Основы языка VHDL. - М.: “СОЛОН-Р”, 2000. – 210 с.
2. Сергиенко А.М. VHDL для проектирования вычислительных устройств. - К.: “Корнейчук”, 2003. – 208 с.
3. Армстронг Дж. Р. Моделирование цифровых систем на языке VHDL. - М.: Мир, 1992. – 175 с.
4. Поляков А.К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры. - М.: “СОЛОН-Пресс”, 2003. – 320 с.
5. Прихожий А.А., Муравьев Г.Л. Система проектирования цифровых СБИС с языком VHDL на ПП ЭВМ. Разработка и использование ПЭВМ// Труды международного симпозиума INFO' 89. - Мн., 1989. - т.2, ч.1. – 6 с.
6. Прихожий А.А., Муравьев Г.Л. и др. Система автоматизированного проектирования СБИС. Процедуры проектирования. Материалы по математическому обеспечению ЭВМ. - Мн., ИТК АН РБ, 1992. – 98 с.
7. Дудкин А.А., Головко В.А. , Муравьев Г.Л. и др. Алгоритмы и подсистемы автоматизированного логического проектирования цифровых СБИС. Материалы по математическому обеспечению ЭВМ. - Мн., ИТК АН РБ, 1994. – 120 с.
8. Муравьев Г.Л., Мухов С.В. Подход к процессной интерпретации VHDL-моделей// Сборник материалов VII международной научно-методической конференции “Наука и образование в условиях социально-экономической трансформации общества”. – Брест: ИСЗ, 2004. - Ч.2.
9. Муравьев Г.Л., Мухов С.В. Требования к реализации компьютерной поддержки обработки битовых векторов// Сборник материалов 8 международной научно-методической конференции “Наука и образование в условиях социально-экономической трансформации общества”. – Витебск: ИСЗ, 19-20 мая, 2005 - Ч.1.
10. Муравьев Г.Л., Шуть В.Н. Интерпретация VHDL-описаний, согласованная с процессным способом моделирования// Вестник БГТУ.– 2005.– № 5(35).– С. 81-84.

Статья поступила в редакцию 21.12.2006

УДК 372.800.26.046.14

Шуть В.Н., Свирский В.М., Муравьев Г.Л., Анфилец С.В.

ГЕНЕРАЦИЯ РЕГУЛЯРНЫХ СВЯЗНЫХ ГРАФОВ

1. Введение

Генерация списков регулярных неизоморфных графов давно является проблемой конструктивной комбинаторики [1]. Уже к концу позапрошлого столетия предпринимались попытки построения списков регулярных графов со степенью вершины равной трем, графов K_n^3. К январю 1889 году удалось построить все графы K_{10}^3 – графы с десятью вершинами и степенью вершины три.

Регулярные графы применяются в peer-to-peer сетях, некоторые из которых базируются на связных регулярных неориентированных графах [2]. Поиск изоморфизма регулярных графов применяется в моделях механического вращения [3],

Шуть Василий Николаевич, *к.т.н., доцент кафедры интеллектуальных информационных технологий БрГТУ.*
Свирский В.М., *магистрант кафедры интеллектуальных информационных технологий БрГТУ.*
Анфилец Сергей Викторович, *магистрант кафедры интеллектуальных информационных технологий БрГТУ.*
Беларусь, Брестский государственный технический университет, 224017, г. Брест, ул. Московская 267.