ИНФОРМАТИКА ИНФОРМАЦИОННЫЕ СИСТЕМЫ

УДК 51:33(075.8)

Программная реализация конечного автомата на языке СИ

С. Ф. Тюрин

Пермский национальный исследовательский политехнический университет
Россия, 614990, г. Пермь, ул. Комсомольский пр., 29
tyurinsergfe@yandex.ru; +7-952-32-02-510
Пермский государственный национальный исследовательский университет
Россия, 614990, г. Пермь, ул. Букирева, 15

О. В. Гончаровский

Пермский национальный исследовательский политехнический университет
Россия, 614990, г. Пермь, ул. Комсомольский пр., 29
35911953@mail.ru; +7-919-45-91-258

Конечный автомат описывается множествами входных, выходных символов и символов внутренних состояний. Кроме того, задаются функции переходов и выходов. Наиболее "быстрая" реализация этих функций выполняется аппаратно. Так, например, строятся устройства управления процессоров на основе так называемых программируемых логических матриц (ПЛИС). Однако часто необходима и "медленная", программная реализация, например при распознавании кодовых последовательностей (пин-код), проверке синтаксиса программ и пр. В статье рассматривается программная реализация автомата на языке СИ методом ПЛИС.

Ключевые слова: конечный автомат; функция переходов; функция выходов; программа на языке СИ.

DOI: 10.17072/1993-0550-2019-3-86-90

Введение

Реализация (вычисление) логических функций и автоматов [1, 2] – одна из важных областей информатики, которая развивается с 1930-х гг. XX в., если не учитывать релейно-контактные схемы, впервые использованные еще в XIX в. Аппаратные решения дали старт эре ЭВМ (компьютеров) с 1940-х гг. прошлого века. Команды логических операций, в том числе и битовые, являются одним из подклассов системы команд процессоров и микроконтроллеров, а также имеются в языках программирования высокого уровня, например, СИ [3]. Однако как таковых команд (операций) реализации автоматных отображений нет до сих пор.

Программную реализацию автоматов (не будем выделять реализацию логических функций, ибо она является частью реализации автоматов) можно разделить на *неуниверсальную* и *универсальную* [4]. *Неуниверсальная* реализует только данный автомат, а *универсальная* – некоторый класс автоматов в рамках некоторой размерности. Например, можно построить реализацию автомата путем использования таблицы переходов-выходов [4], но для больших размерностей такая таблица будет весьма громоздка.

Неуниверсальная реализация представляет собой программу вычисления нового состояния и выходных функций только для данного автомата.

Универсальная реализация использует отдельно задаваемую таблицу, которая меняется в зависимости от заданного автомата.

То есть, для нового автомата необходимо задать только размер таблицы и саму таблицу. Более экономный подход использует минимизированные логические функции в дизъюнктивной нормальной форме (ДНФ), в отличие от табличного, использующего совершенную дизъюнктивную нормальную форму (СДНФ). Например, используется алгоритм реализации программируемой логической матрицы (ПЛИМ), в которой функции заданы в ДНФ, в отличие от реализации на постоянной (оперативной) памяти (ПЗУ, ОЗУ), использующей СДНФ представление.

Такой алгоритм приведен в [5, 6] и примеры его использования на языке Ассемблер описаны в [7, 8]. Реализуем этот алгоритм на языке СИ, пользуясь ресурсом [9].

1. Подготовка исходных данных

Рассмотрим пример реализации автомата-распознавателя последовательности.

Пусть это последовательность на двух бинарных входах ab : 0,1,3,2. После получения правильной последовательности на выходе z_1 формируется единица. Если последовательность нарушается, на выходе z_2 формируется единица. Одновременное изменение сразу двух входов исключим из рассмотрения.

Состояния автомата обозначим $y(t)$ – текущее, $y(t+1)$ – следующее. Граф переходов автомата с дугами, обозначенными в

формате $\frac{ab}{z_2 z_1}$ изображен на рис. 1.

Рис. 1. Граф переходов автомата-распознавателя бинарной последовательности 00,01,11,10

По графу переходов получим таблицу переходов-выходов [7] (рис. 2).

$y_2 y_1(t)$	ab				
	00	01	11	10	
00	00H 00/00	01H 01/00	03H ~	02H 00/10	
01	04H 01/10	05H 01/00	07H 11/00	06H ~	
11	0CH ~	0DH 11/10	0FH 11/00	0EH 11/01	$\frac{y_2 y_1(t+1)}{z_2 z_1}$

Рис. 2. Таблица переходов-выходов

Таким образом, получаем систему логических функций:

$$\begin{cases} y_2(t+1)_{y_2 y_1 ab} = 4, 7, 12, 13, 15, 14[0, 1, 2, 5], \\ y_1(t+1)_{y_2 y_1 ab} = 1, 4, 5, 7, 12, 13, 15, 14[0, 2], \\ z_{2 y_2 y_1 ab} = 2, 4, 12, 13[0, 1, 5, 7, 15, 14], \\ z_{1 y_2 y_1 ab} = 14[0, 1, 2, 4, 5, 7, 12, 13, 15]. \end{cases} \quad (1)$$

Системе (1) соответствует массив масок-констант [7, 8]:

0FH, 01H, 04H, 0FH, 02H, 02H, 0FH, 04H, 06H, 0FH, 05H, 04H, 0FH, 07H, 0CH, 0FH, 0DH, 0EH, 0FH, 0FH, 0CH, 0FH, 0EH, 0DH, 00H.

В данном случае используем СДНФ, поскольку минимизация приводит к большему числу масок (9 вместо 8):

$$\begin{cases} y_2(t+1)_{y_2 y_1 ab} = y_2 \vee y_1 a \vee y_1 \bar{b}; \\ y_1(t+1)_{y_2 y_1 ab} = y_1 \vee b; \\ z_{1 y_2 y_1 ab} = y_1 \bar{a} \bar{b}; \\ z_{2 y_2 y_1 ab} = y_2 \bar{a} \vee y_1 \bar{a} \bar{b} \vee y_2 \bar{a} \bar{b}. \end{cases} \quad (2)$$

2. Разработка программы

Программа на языке СИ содержит блок задания констант для вычисления системы логических функций:

<pre>#include <stdio.h> #include <stdlib.h> #define _CRT_SECURE_NO_WARNINGS typedef struct { unsigned char A1; unsigned char A2; unsigned char A3; } A_type; A_type A[24]= { {0x0f,0x01,0x04}, {0x0f,0x02,0x02}, {0x0f,0x04,0x06}, {0x0f,0x05,0x04}, {0x0f,0x07,0x0c}, {0x0f,0x0d,0x0e}, {0x0f,0x0f,0x0c}, {0x0f,0x0e,0x0d}, {0,0,0} }; unsigned int x=0x0; //Input Vector unsigned char y=0x0; //Memory unsigned char z=0x0; //Output Vector unsigned char T=0x0; //Temporal</pre>	<p>Служебная информация</p> <p>Задание массива констант для кода 0132</p> <p>Длина массива (24) условная, рассчитана на максимальное количество констант.</p>
---	---

В основном блоке вводится символ с клавиатуры, вычисляются функции переходов и выходов и автомат переводится в новое состояние:

```
int main()
{
    do
    { scanf("%x",&x);//input console symbol
      printf ("x=%0x\n",x);//print of the console symbol
      {
        //y=0xc;
        x=y|x;//get new input vector with memory
        printf ("x=0x%x\n",x);
        int idx = 0;
        for (idx=0;idx<24;idx++)
        {
            if(A[idx].A1==0)//condition of the massive end
                break;
            T=x&A[idx].A1;//masc of variables
            T=T^A[idx].A2;//masc of not inverse variables
            if(T==0)//condition of the product term truth
            {
                z=z|A[idx].A3;//output vector
                printf ("z (%d)=0x%x\n",idx,z);//print output vector
                //y=z&0xc;// set new state
                //printf ("y (%d)=0x%x\n",idx,y);//print new state
                //break;
            }
            y=z&0xc;// set new state
            printf ("y (%d)=0x%x\n",idx,y);//print new state
        }
        while (1);//infinity cicle
    }
}
```

Ввод символа, вычисление функций переходов и выходов, перевод в новое внутреннее состояние и выдача выходных сигналов

Принцип вычисления системы логических функций следующий. Входной вектор маскируется первой константой данной конъюнкции — существенных переменных (выделяются существенные переменные) $T=x\&A[idx].A1$.

Далее результат складывается по модулю два со второй константой $T=T^A[idx].A2$.

Если результат нулевой, это свидетельствует об истинности конъюнкции, поэтому выполняется поразрядная дизъюнкция выходного вектора z с третьей маской (маской функций выходов) $z=z|A[idx].A3$.

Иначе выбирается следующая группа констант, если это не последняя константа, что фиксируется маркером окончания массива $\{0,0,0\}$.

3. Тестирование программы

В начале проведем тестирование одноконтного эквивалента автомата по таблице, изображенной на рис. 3.

y2(t)	y1(t)	a	b	Code Input	y2(t+1)	y1(t+1)	z1	z2	Code Output
1	1	0	0	0x0c	0	0	0	0	0
1	1	0	1	0x0d	1	1	1	0	0x0e
1	1	1	0	0x0e	1	1	0	1	0x0d
1	1	1	1	0x0f	1	1	0	0	0x0c
0	0	0	0	0x00	0	0	0	0	0
0	0	0	1	0x01	0	1	0	0	0x04
0	0	1	0	0x02	0	0	1	0	0x02
0	0	1	1	0x03	0	0	0	0	0
0	1	0	0	0x04	0	1	1	0	0x06
0	1	0	1	0x05	0	1	0	0	0x04
0	1	1	0	0x06	0	0	0	0	0
0	1	1	1	0x07	1	1	0	0	0x0c
1	0	0	0	0x08	0	0	0	0	0
1	0	0	1	0x09	0	0	0	0	0
1	0	1	0	0x0a	0	0	0	0	0
1	0	1	1	0x0b	0	0	0	0	0

Рис. 3. Таблица тестирования одноконтного эквивалента автомата

При этом код состояния y указываем вручную. Тестирование показывает, что таблица (рис. 3, соответствующий рис. 2) реализуется правильно.

Далее "включаем" перевод в новое внутреннее состояние. Проверяем также правильность ввода входного сигнала x , формирование внутреннего состояния y и выходного вектора z . Вводим последовательность 0132 — рис. 4. Видим, что формируется 0xd, это клетка 0ЕН в таблице, изображенной на рис. 2. То есть, формируется двоичный вектор 1101 (содержимое клетки 0ЕН). Это означает, что автомат находится в состоянии 11 ($y2(t+1)=1$, $y1(t+1)=1$), при этом $z2=0$, $z1=1$. Таким образом, последовательность "распознана".

```
C:\Projects\Project01\Project01.exe
0
x=0
x=0x0
y (8)=0x0
1
x=1
x=0x1
z (0)=0x4
y (8)=0x4
3
x=3
x=0x7
z (4)=0xc
y (8)=0xc
2
x=2
x=0xe
z (7)=0xd
y (8)=0xc
```

Рис. 4. Тестирование автомата на последовательности 0132

Тестирование на последовательности 02 приводит к получению $z=0x2$ (рис. 5).

```

C:\Projects\Project01\Project01.exe
0
x=0
x=0x0
y (8)=0x0
2
x=2
x=0x2
z (1)=0x2
y (8)=0x0

```

Рис. 5. Тестирование автомата на последовательности 02

Это означает формирование $z2=1$, $z1=0$. То есть последовательность нарушена. Аналогично выявляется нарушение последовательности при подаче 010 (рис. 6.).

```

C:\Projects\Project01\Project01.exe
0
x=0
x=0x0
y (8)=0x0
1
x=1
x=0x1
z (0)=0x4
y (8)=0x4
0
x=0
x=0x4
z (2)=0x6
y (8)=0x4

```

Рис. 6. Тестирование автомата на последовательности 010

Получаем формирование $z2=1$, $z1=0$. При этом $y2(t+1)=0$, $y1(t+1)=1$. Тестирование на последовательности 0131 (рис. 7).

```

C:\Projects\Project01\Project01.exe
0
x=0
x=0x0
y (8)=0x0
1
x=1
x=0x1
z (0)=0x4
y (8)=0x4
3
x=3
x=0x7
z (4)=0xc
y (8)=0xc
1
x=1
x=0xd
z (5)=0xe
y (8)=0xc

```

Рис. 7. Тестирование автомата на последовательности 0131

Получаем $y2(t+1)=1$, $y1(t+1)=1$, $z2=1$, $z1=0$. Таким образом, автомат работает правильно на всех возможных последовательностях, если учитывать возможные изменения только одного бита. Аналогично можно убедиться в правильности реакции на остальные варианты последовательностей.

Выводы

Таким образом, разработана универсальная программа реализации конечного автомата методом ПЛИС на языке СИ. Для реализации нового автомата в общем случае необходимо изменить только массив констант. Проведено тестирование программы на примере реализации автомата-распознавателя последовательности (пин-кода). В представленной программной реализации в явном виде код не указан, что может в какой-то мере повысить информационную безопасность.

В дальнейшем целесообразно разработать программу подготовки массива констант по заданному высокоуровневому описанию автомата.

Список литературы

1. Кузнецов О.П. Дискретная математика для инженера. 3-е изд., перераб. и доп. СПб: Лань, 2005. 395 с.
2. Новиков Ф.А. Дискретная математика для программиста. СПб.: Питер, 2008. 501 с.
3. Б. Керниган, Д. Ритчи. Язык программирования С. URL: <https://nsu.ru/xmlui/bitstream/handle/nsu/9058/kr.pdf?sequence=1&isAllowed=y> (дата обращения: 09.08.19).
4. Лазарев В.Г., Пийль Е.И. Синтез управляющих автоматов. 3-е изд., перераб. и доп. М.: Энергоатомиздат, 1989. 328 с.
5. Gutman F. 8080 Routine Emulates PLA Hardware. EDN. 1979. Vol. 24, № 18. P. 81–82.
6. Алексенко А.Г., Галицин А.А., Иванников А.Д. Проектирование радиоэлектронной аппаратуры на микропроцессорах: Программирование, типовые решения, методы отладки. М.: Радио и связь, 1984. 272 с.
7. Тюрин С.Ф. Вычислительная техника и информационные технологии. Руководство к лабораторным работам в системе Proteus 7.2. Пермь: Изд-во Перм. гос. техн. ун-та, 2010. 135 с.
8. Тюрин С.Ф., Гончаровский О.В., Громов О.А. Вычислительная техника и информационные технологии. Аппаратные средства вычислительной техники: конспект лекций. Пермь: Изд-во Перм. гос. техн. ун-та, 2011. 324 с.
9. Кавалеров М.В. Методические материалы по программированию. URL: <http://file.at.pstu.ru/materials/courses/pz/>

(дата обращения: 26.03.19).

Software implementation of the finite state machine in C

S. F. Tyurin

Perm National Research Polytechnic University; 29, Komsomolsky Av., Perm, 614990, Russia
Perm State University; 15, Bukireva st., Perm, 614990, Russia
tyurinsergfeo@yandex.ru; +7 952-320-02-510

O. V. Goncharovskij

Perm National Research Polytechnic University; 29, Komsomolsky Av., Perm, 614990, Russia
35911953@mail.ru; +7-919-45-91-258

A finite state machine is described by sets of input and output symbols and symbols of internal states. In addition, transition and output functions are set. The most "fast" implementation of these functions is performed in hardware. Therefore, for example, processor control devices are built based on the so-called programmable logic array (PLA). However, a "slow" software implementation is often necessary, for example, when recognizing code sequences ("pin"), checking the syntax of programs, etc. The article discusses the software implementation of the state machine by the C language using the PLA algorithm.

Keywords: *Finite state machine; Transition function; Output function; C program.*