**Методика разработки конечного автомата и дополнительные комментарии**

Вначале несколько определений.

Основой конечных автоматов является память. Значения в этой памяти называют кодами состояний автомата. Значение кода, который в конкретный момент времени, определяет поведение автомата называют текущим состоянием автомата.

1. Разработка конечного автомата начинается с таблицы переходов, начальное состояние которой можно изобразить следующим образом:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Символ  состояния | Наименование состояния | Входные сигналы и переходы  в новые состояния | | | | | |
| CLR | X1 | X2 | X3 | . . . | XN |
| S1 | Начальное состояние (сброс) | S1 |  |  |  |  |  |
| S2 |  | S1 |  |  |  |  |  |
| S3 |  | S1 |  |  |  |  |  |
| . . . |  | S1 |  |  |  |  |  |
| SN | Конечное состояние | S1 |  |  |  |  |  |

Где:

S1 . . . SN - это символы необходимых состояний управляемого объекта,

CLR, X1 . . . XN – это символы сигналов, которые приводят к изменению состояний

Пересечение строки состояния и столбца сигнала образует ячейку перехода. В этой ячейке отображается символ нового состояния, в которое должен перейти автомат при поступлении такого сигнала. Если ячейка не заполнена, это означает что в соответствующем состоянии соответствующий сигнал заблокирован (замаскирован), то есть, просто игнорируется.

В начальном состоянии таблицы переходов всегда следует определять начальное состояние (S1) и сигнал установки в начальное состояние (CLR - сброс), который не маскируется в любом другом состоянии и переводит автомат в состояние (S1). Кроме того, всегда следует определять конечное состояние, которое предназначено для остановки работы автомата по причинам завершения цикла управления или/и аварийных ситуаций.

1. Предполагается, что с каждым состоянием связано управляющее воздействие на объект управления. Для описания сигналов управления подготавливается таблица выходных сигналов:

|  |  |  |
| --- | --- | --- |
| Символ  Выходного  воздействия | Наименование выходного воздействия | Реализация выходного воздействия, при этом, символы состояний используются как символы одноименных сигналов |
| Y1 | Начальное состояние (сброс) | S1 |
| Y2 |  |  |
| Y3 |  |  |
| . . . |  |  |
| YN | Конечное состояние |  |

1. В соответствии с логикой управления, последовательно выполняется заполнение ячеек перехода автомата в новые текущие состояния при поступлении тех или иных входных сигналов. При этом, по необходимости, в соответствующие таблицы добавляются строки новых состояний, столбцы новых сигналов, строки выходных воздействий.
2. При заполнении таблицы переходов возможно несколько ситуаций с учетом особенностей объекта управления:

* В текущем состоянии предполагается значимость только одного входного сигнала для перехода в новое состояние. В этом случае все остальные сигналы просто маскируются (блокируются);
* В текущем состоянии предполагается значимость нескольких входных сигналов, которые гарантировано не могут появляться одномоментно. В этом случае также достаточно просто замаскировать лишние сигналы;
* Если в текущем состоянии могут иметь значимость несколько сигналов, появление которых невозможно предсказать, то при их одномоментном появлении необходимо задать им приоритетность. Такую приоритетность следует реализовать в приоритетном шифраторе маски входных сигналов. Особо следует подчеркнуть, что сигнал сброса (CLR) должен обладать наивысшим приоритетом.

1. С учетом сказанного выше блок-функциональную схему конечного автомата можно представить в следующем виде:

Шифратор выходных сигналов

Y = Function(S, X) автомат Мили

Y = Function(S) автомат Мура

Память конечного автомата и дешифратор в сигналы состояний

S =Function(S, W)

Приоритетный шифратор (маска) входных

Сигналов

W=Function(X, S)

X - Входные сигналы

Y - Выходные сигналы

W – сигналы установки

памяти

S – сигнал текущего состояния

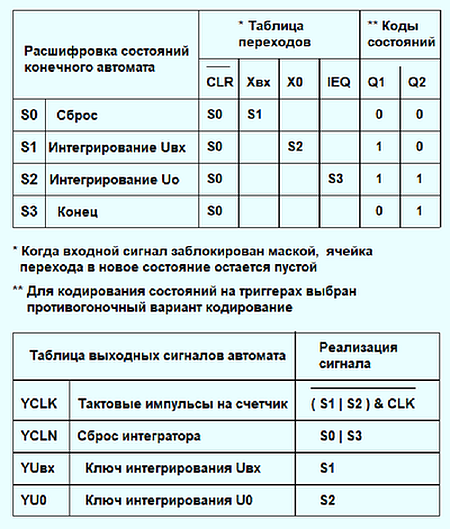
1. Ели вы проектируете конечный автомат с организацией памяти в виде линейки триггеров, то необходимо для каждого состояния определить уникальную комбинацию битов на выходах этих триггеров. Для этого используется таблица кодирования состояний автомата.

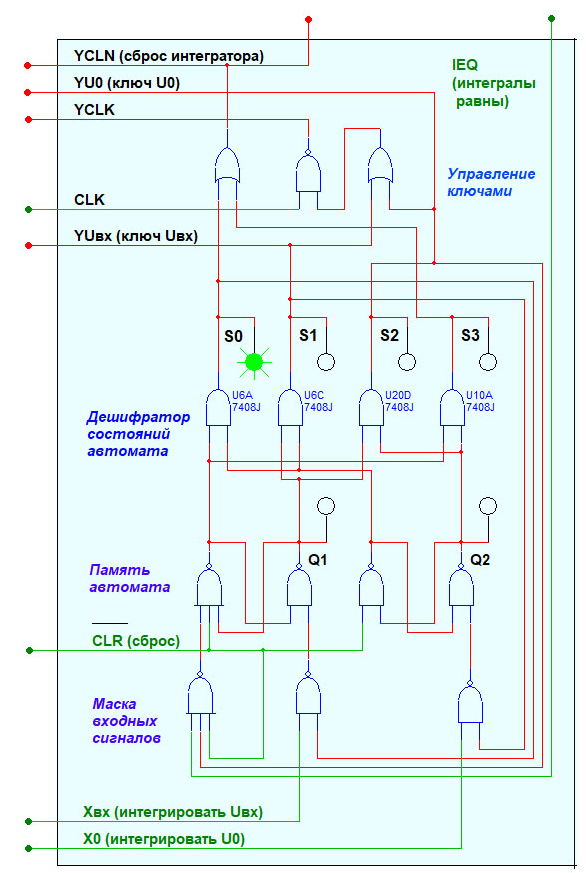
Основными особенностями выбора количества (M) триггеров и комбинаций битов для кода каждого состояния являются следующие правила:

* Любой переход между текущим и новым состоянием должен изменять не более одного бита в соответствующих кодах. Это правило позволяет исключить относительные задержки при переключении триггеров, а значит избежать пусть кратких, но непредвиденных промежуточных состояний. Такое кодирование называется противогоночным.
* Количество триггеров выбирается с учетом количества состояний автомата плюс выполнение правила противогоночного кодирования.

Примерный вид таблицы кодирования состояний автомата:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Символ  состояния | Наименование состояния | Код состояния | | | | | |
| Q1 | Q2 | Q3 | Q4 | . . . | QM |
| S1 | Начальное состояние (сброс) | 0 | 0 | 0 | 0 |  | 0 |
| S2 |  | 1 | 0 | 0 | 0 |  | 0 |
| S3 |  | 0 | 1 | 0 | 0 |  | 0 |
| . . . |  |  |  |  |  |  |  |
| SN | Конечное состояние | 1 | 1 | 1 | 1 |  | 1 |

1. ПРИМЕР построения таблиц достаточно простого автомата для АЦП двухкратного интегрирования:

Полная модель АЦП двухкратного интегрирования представлена в архиве моделей раздела конечных автоматов. На рисунке ниже показана реализация этого автомата:

1. КОММЕНТАРИЙ 1. Если вы проектируете группу автоматов, причем некоторые из них планируете использовать подобно процедурам в программировании, то вам потребуется специальное состояние «запуск и работа внешнего автомата». При переходе в такое состояние формируется выходной сигнал «старт внешнего автомата» и блокируются все входные сигналы кроме «конец работы внешнего автомата». Такая архитектура неявно реализует стековую организацию вызовов, однако не позволяет параллельную работу любого из автоматов-процедур.
2. КОММЕНТАРИЙ 2. Следует также отметить возможность применения техники конечных автоматов в традиционном программировании. Обычно программируемые конечные автоматы используются в виде сложных (а часто и алгоритмически изменяемых) операциях ветвления. В этом случае логика разработки таких программ имеет вид:

* Определяем и инициализируем глобальные данные для программного автомата, пусть имя этих данных будет **AData**. Для инициализации будем использовать глобальную константу **DefATab,** которая будет совмещать в себе таблицы переходов, маску входных сигналов и таблицу выходов. Для простоты предположим, что названые таблицы будут представлены в составе **AData** как двухмерный массив записей **ATab** следующей структуры (индекс строки состояния, индекс столбца состояния, косвенная ссылка на процедуру, реализующую выходной сигнал (пусть имена таких процедур будут **YPR**) с примерным синтаксисом вызова: **YPR** (**AData**). Детальнее, описание таблицы смотрите в примере исходного текста;
* Для работы автомата будем использовать процедуру (пусть ее имя будет **RunAutomat**) с примерным синтаксисом: **RunAutomat** **(****RqXAr)**, где **RqXAr** это однострочный массив входных сигналов:
* Если во внешней алгоритмической среде возникают изменения в **RqXAr**, то процедура **RunAutomat** вызывается этой средой. В этом случае **RunAutomat** сканирует вектор входных сигналов (возможно с некоторым приоритетным порядком) и первый из найденных сигналов (значение отличное от нуля) используется для перехода в новое состояние, которое обновляет поле «текущее состояния автомата», кроме того, вызывается выполнение соответствующей процедуры **YPR**, если такая процедура задана. На этом работа процедура **RunAutomat** завершается и управление возвращается во внешнюю среду.
* ПРИМЕЧАНИЕ. Количество столбцов массива переходов должно быть равно количеству столбцов в векторе входных сигналов.

1. КОММЕНТАРИЙ 3. Понятно, что автомат, реализованный как программа, можно выполнить в технологии объектно-ориентированного программирования.

Например:

// ====================================================================

// СЕГМЕНТ ОПИСАНИЙ СТРУКТУРЫ ДАННЫХ

// ====================================================================

// Максимальные индексы таблиц переходов и входных сигналов

const MaxNumR = 4; // Число строк или состояний

const MaxNumC = 4; // Число столбцов или входных сигналов

// Описание сигнала

type TXValue = integer;

// Описание таблицы - вектора входных сигналов

type TXAr = array[1..MaxNumC] of TXValue;

// Описание интерфейса процедур выходных воздействий

type TYProc = procedure (pAData : Pointer);

// Описание совмещенной таблицы маски, переходов и выходов

type TATab = array[1..MaxNumR, 1..MaxNumC] of

record

NS : integer; // Индекс перехода в новое состояние

YP : TYProc; // Процедура вызываемая в новом состоянии

end;

// ---------------------------------------------------------------------------------------------------------------------

// Описание системы данных для конечного автомата

type TAData = record

XAr : TXAr; // Вектор входных сигналов

IX : integer; // Индекс сигнала, который вызвал переход

CS : integer; // Индекс текущего состояния

ATab : TATab; // Таблица переходов, маски, выходов

end;

// ====================================================================

//

// ====================================================================

// Процедура 1 имитации выходного воздействия

procedure YPR1(pAData : Pointer);

var wStr : string;

begin

with TAData(pAData^)do

begin

wStr := 'Переход в начальное состояние';

ShowMessage(wStr);

end;

end;

// ---------------------------------------------------------------------------------------------------------------------

// Процедура 2 имитации выходного воздействия

procedure YPR2(pAData : Pointer);

var wStr : string;

begin

with TAData(pAData^)do

begin

wStr := 'Переход в состояние: ' + IntToStr(CS) + #13#10;

wStr := wStr + 'по сигналу X' + IntToStr(IX);

wStr := wStr + ' с значением: ' + IntToStr(XAr[IX]);

ShowMessage(wStr);

end;

end;

// ====================================================================

// ТАБЛИЦА ПЕРЕХОДОВ, МАСОК, ВЫХОДОВ

// ====================================================================

// NS – Номер состояния для перехода, если 0, то такой переход заблокирован (замаскирован)

// YP – Имя процедуры выходного воздействия, если nil, то тереход без вызова процедуры

const DefATab : TATab =

(

((NS:2; YP:YPR2), (NS:0; YP:nil), (NS:0; YP:nil), (NS:0; YP:nil) ), // S1

((NS:0; YP:nil), (NS:0; YP:nil), (NS:3; YP:YPR2), (NS:0; YP:nil) ), // S2

((NS:0; YP:nil), (NS:2; YP:YPR2), (NS:0; YP:nil), (NS:4; YP:YPR2)), // S3

((NS:0; YP:nil), (NS:1; YP:YPR1), (NS:0; YP:nil), (NS:0; YP:nil) ) // S4

);

// ====================================================================

// НАЧАЛО ОПИСАНИЕ КЛАССА TAutomat

// ====================================================================

type TAutomat = class(TObject)

private

// Система данных объекта

AData : TAData;

public

// Создание объекта

constructor Create(RqATab : TATab);

// Выполнение автомата

procedure RunAutomat (RqXAr : TXAr);

// Чтение текущего состояния объекта

property GetCurrStat : Integer read AData.CS;

end;

// ---------------------------------------------------------------------------------------------------------------------

// Выделение памяти объекту.

// Очистка вектора входных сигналов в объекте.

// Копирование внешней таблицы переходов, масок, выходов во таблицу объекта.

constructor TAutomat.Create(RqATab : TATab);

var wR, wC : Integer;

begin

inherited Create(); // Получить память для объекта

with AData do

begin

CS := Low(ATab); // Установить номер начального состояния

// Очистить вектор входных сигналов

// Скопировать таблиу переходов, масок, выходов

for wC := Low(XAr) to High(XAr) do

begin

XAr[wC] := 0;

for wR := Low(ATab) to High(ATab) do ATab[wR,wC] := RqATab[wR,wC];

end;

end;

end;

// ---------------------------------------------------------------------------------------------------------------------

// Копирование входных сигналов из внешней среды

// Сканирование входных сигналов для поиска ненулевого и не заблокированного сигнала

// Выполнение перехода по найденному сигнаду или выход если сигнал не найден

// Выполнение процедуры выхода, если она определена

procedure TAutomat.RunAutomat(RqXAr : TXAr);

var wInd : Integer;

wFound : Boolean;

wPR : TYProc;

wP : Pointer;

begin

with AData do

begin

for wInd := Low(XAr) to High(XAr) do XAr[wInd] := RqXAr[wInd];

wFound := False;

// Приоритетный поиск сигнала

for wInd := Low(XAr) to High(XAr) do

begin

if (AData.XAr[wInd] > 0) and (AData.ATab[CS,wInd].NS > 0)

then begin

IX := wInd;

wFound := True;

Break;

end;

end;

if wFound

then begin

wPR := ATab[CS,IX].YP;

CS := ATab[CS,IX].NS;

if Assigned(wPR)

then begin

wP := Addr(AData);

wPR(wP);

end;

end;

end;

end;

// ====================================================================

// КОНЕЦ ОПИСАНИЯ КЛАССА TAutomat

// ====================================================================

// ====================================================================

// НАЧАЛО СЕГМЕНТА ВНЕШНЕЙ СРЕДЫ ДЛЯ РАБОТЫ С ОЪЕКТОМ

// ====================================================================

// Вектор входных сигналов

var XArr : TXAr;

// Ссылка на объект автомата

var AUTOMAT : TAutomat;

// Создать объект автомата

procedure TForm1.btn1Click(Sender: TObject);

begin

if not Assigned(AUTOMAT)

then begin

AUTOMAT := TAutomat.Create(DefATab);

edt1.Text := 'S' + IntToStr(AUTOMAT.GetCurrStat);

end;

end;

// Изменить выбранные входные сигналы и выполнить автомат

procedure TForm1.btn2Click(Sender: TObject);

begin

XArr[1] := StrToInt(cbb1.Text);

XArr[2] := StrToInt(cbb2.Text);

XArr[3] := StrToInt(cbb3.Text);

XArr[4] := StrToInt(cbb4.Text);

if Assigned(AUTOMAT)

then begin

AUTOMAT.RunAutomat(XArr);

edt1.Text := 'S' + IntToStr(AUTOMAT.GetCurrStat);

end

else begin

ShowMessage('Вначале создайте объект автомата');

end;

end;

// Удалить объект автомата

procedure TForm1.btn3Click(Sender: TObject);

begin

if Assigned(AUTOMAT) then

begin

AUTOMAT.Free;

AUTOMAT := nil;

edt1.Text := '';

end;

end;

// ====================================================================

// КОНЕЦ СЕГМЕНТА ВНЕШНЕЙ СРЕДЫ ДЛЯ РАБОТЫ С ОЪЕКТОМ

// ====================================================================

Полный исходный текст тестового приложения, которое использует объект автомата, приведен в архиве раздела в папке с именем «Delphi\_модель». Дополнительно в папке присутствует исполняемый в среде Windows файл Automat01.exe и две вспомогательных библиотеки rtl70.bpl и vcl70.bpl. Эти библиотеки позволяют исполнять Automat01.exe без установки в Windows системы Delphi.

Воронов С.И., Киев, 2023г.