Модел. и анализ информ. систем. Т. **20**, № **6** (2013) 78–94

УДК 517.9

Построение и верификация LD-программ ПЛК по LTL-спецификации

Кузьмин Е. В.[1], Соколов В. А.[2], Рябухин Д. А.

Ярославский государственный университет им. П. Г. Демидова
150000 Россия, г. Ярославль, ул. Советская, 14

e-mail: {*kuzmin,sokolov*}*@uniyar.ac.ru, dmitriy_ryabukhin@mail.ru*

получена 28 октября 2013

Ключевые слова: программируемые логические контроллеры, технология программирования, спецификация и верификация программ, релейные диаграммы

Предлагается подход к построению и верификации LD-программ логических контроллеров (ПЛК) для «дискретных» задач. Спецификация программного поведения проводится на языке темпоральной логики линейного времени LTL. Программирование осуществляется на языке LD (Ladder Diagram) по LTL-спецификации. Анализ корректности LTL-спецификации производится с помощью программного средства символьной проверки модели Cadence SMV. Подход к программированию и верификации LD-программ ПЛК демонстрируется на примере. Для дискретной задачи приводятся LD-программа, ее LTL-спецификация и SMV-модель.

Целью статьи является описание подхода к программированию ПЛК, который бы обеспечивал возможность анализа корректности LD-программ ПЛК с помощью метода проверки модели.

Поэтому изменение значения каждой программной переменной описывается с помощью пары LTL-формул. Первая LTL-формула описывает ситуации, при которых происходит возрастание значения соответствующей переменной, вторая LTL-формула задает условия, приводящие к уменьшению значения переменной. Рассматриваемые для спецификации поведения переменных LTL-формулы являются конструктивными в том смысле, что по ним производится построение ПЛК-программы, которая соответствует темпоральным свойствам, выраженным этими формулами. Таким образом, программирование ПЛК сводится к построению LTL-спецификации поведения каждой программной переменной. Кроме этого, по LTL-спецификации строится SMV-модель LD-программы ПЛК, которая затем проверяется на корректность (относительно дополнительных общепрограммных LTL-свойств) методом проверки модели с помощью средства верификации Cadence SMV.

[1]Работа проводилась при финансовой поддержке РФФИ, грант №12-01-00281-а.

[2]Работа проводилась при финансовой поддержке Минобрнауки РФ в рамках ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы, соглашение №14.B37.21.0392 от 06.08.2012.

Введение

Применение программируемых логических контроллеров (ПЛК) в системах управления сложными производственными процессами предъявляет строгие требования корректности к программам ПЛК. Любая программная ошибка считается недопустимой. Вместе с тем программирование ПЛК представляет собой прикладную область, в которой существующие наработки в области формальных методов моделирования и анализа программных систем могут иметь успешное применение.

Ранее в работах [1, 2, 3] был предложен подход к построению и верификации «дискретных» ПЛК-программ, обеспечивающий возможность анализа их корректности с помощью метода проверки модели (model checking) [7], — программирование и верификация по LTL-спецификации. При этом подходе в качестве языка спецификации программного поведения используется язык темпоральной логики LTL. Анализ корректности LTL-спецификации производится с помощью программного средства символьной проверки модели Cadence SMV [10].

В этой статье описывается технология построения и верификации LD-программ ПЛК на основе подхода к программированию и верификации по LTL-спецификации. Технология демонстрируется на примере LD-программы ПЛК управления библиотечным подъемником.

ПЛК — «реагирующая» система, имеющая множество входов, подключенных посредством датчиков к объекту управления, и множество выходов, подключенных к исполнительным устройствам [9, 6]. Программа ПЛК выполняется в рабочем цикле: считывание входов, выполнение программы и выставление выходов.

Язык релейных диаграмм LD (Ladder Diagram) — графический язык, реализующий структуры электрических цепей. LD-диаграмма представляется в виде двух вертикальных шин питания. Между ними расположены цепи, образованные соединением контактов. Нагрузкой каждой цепи служит реле. Реле имеет контакты, которые можно использовать в других цепях. Каждому контакту ставится в соответствие логическая переменная, определяющая его состояние. Если контакт замкнут, то переменная имеет значение 1. Если разомкнут – 0. Имя переменной пишется над контактом и служит его названием. Последовательное соединение контактов или цепей равноценно операции «И». Параллельное соединение образует «ИЛИ». Нормально разомкнутый контакт обозначается —| |—, нормально замкнутый, инверсный, контакт имеет обозначение —|/|— и замыкается, если значение переменной 0. Инверсный контакт равнозначен операции «НЕ». Обмотка реле изображается как —( )—. Помимо обычных реле, в релейных схемах применяются реле с самофиксацией. Такое реле имеет две обмотки, переключающие его из одного положения в другое, что реализует элементарную ячейку памяти. Такое реле реализуется при помощи двух специальных обмоток SET и RESET. Обмотки типа SET обозначаются —(S)—. Обмотки типа RESET имеют обозначение —(R)—. Если соответствующая обмотке —(S)— переменная принимает значение 1, то сохраняет его бесконечно. Вернуть данную переменную в 0 можно только обмоткой —(R)—.

LD-программа ПЛК выполняется последовательно слева направо и сверху вниз. В каждом рабочем цикле однократно выполняются все цепи диаграммы. Если реле в цепи изменит переменную, то цепи ниже получат новое значение переменной сразу, а цепи, расположенные выше, — только в следующем цикле.

1. Метод проверки модели для ПЛК-программ

Задача проверки модели (Model Checking) состоит в определении выполнимости для конечной модели программы (в виде структуры Крипке) свойства, выраженного формулой темпоральной логики.

Структурой Крипке над множеством элементарных высказываний P называется система переходов $\mathcal{S} = (S, s_0, \to, L)$, где S — конечное множество состояний (модели программы), $s_0 \in S$ — начальное состояние, $\to \subseteq S \times S$ — отношение переходов, $L : S \to 2^P$ — функция, помечающая каждое состояние множеством элементарных высказываний, истинных в этом состоянии.

Путь в структуре Крипке из состояния s_0 — это бесконечная последовательность состояний $\pi = s_0 s_1 s_2 \dots$ такая, что $\forall\, i \geq 0$ выполняется $s_i \to s_{i+1}$.

В качестве языка спецификации поведенческих свойств программной модели рассматривается язык темпоральной логики линейного времени LTL (Linear-Time Temporal Logic). Выбор логики LTL связывается с тем, что программа ПЛК является классической реактивной (реагирующей) управляющей системой, которая, будучи однажды запущенной, должна иметь корректное бесконечное поведение. Условия корректности удобно задавать в виде шаблонов свойств, которым должны соответствовать корректные исполнения программы. В темпоральной логике LTL каждая формула по сути представляет собой такой шаблон.

Формулы логики LTL строятся по следующей грамматике при $p_i \in P$:

$$\varphi, \psi ::= \mathit{true} \mid p_0 \mid p_1 \mid \dots \mid p_n \mid \neg\, \varphi \mid \psi \wedge \varphi \mid X\varphi \mid \psi\, U\, \varphi \mid F\, \varphi \mid G\, \varphi.$$

Формула логики LTL описывает свойство одного пути структуры Крипке, выходящего из некоторого выделенного текущего состояния. Темпоральные операторы X, F, G и U имеют следующую интерпретацию: $X\varphi$ означает, что формула φ должна выполняться в следующем состоянии, $F\varphi$ — φ должна выполняться в некотором будущем состоянии пути, $G\varphi$ — φ должна выполняться в текущем состоянии и во всех будущих состояниях пути, $\psi\, U\, \varphi$ — φ должна выполняться в текущем или будущем состоянии при том, что во всех состояниях (начиная с текущего) до этого момента должна выполняться формула ψ. Операторы F и G являются производными и вводятся для удобства спецификации свойств: $F\, \varphi = \mathit{true}\, U \varphi$, $G\, \varphi = \neg F \neg \varphi$. Кроме того, далее будут использоваться классические логические связки $\vee$ и $\Rightarrow$.

Структура Крипке удовлетворяет формуле (свойству) φ логики LTL, если φ выполняется для всех путей, выходящих из начального состояния s_0.

Модель Крипке для программы ПЛК строится естественным образом. Состояние модели — это состояние программы (вектор значений всех переменных) ПЛК после одного полного прохода рабочего цикла. Начальное состояние модели — состояние программы после инициализации значений переменных. Таким образом, в качестве перехода из одного состояния модели в другое (или то же самое) рассматривается полное исполнение программы за один проход рабочего цикла ПЛК.

Отметим, что модель от исходной ПЛК-программы отличается лишь дискретным представлением работы таймеров (см. [3]). Если таймеры в программе не применяются, то поведение модели полностью совпадает с поведением программы.

В качестве элементарных высказываний модели будут рассматриваться логические (с применением арифметических операторов и операторов сравнения) выражения над переменными ПЛК-программы.

2. Концепция программирования и верификации

В соответствии с концепцией построения и верификации программ ПЛК по LTL-спецификации [1, 2, 3], смысл которой состоит в обеспечении возможности анализа корректности ПЛК-программ методом проверки модели, значение каждой переменной должно изменяться только в одном месте программы и не более одного раза за одно полное выполнение программы при прохождении рабочего цикла ПЛК. Поэтому изменение значения каждой программной переменной описывается с помощью пары LTL-формул. Первая LTL-формула описывает ситуации, при которых происходит возрастание значения соответствующей переменной, вторая LTL-формула задает условия, приводящие к уменьшению значения переменной. Рассматриваемые для спецификации поведения переменных LTL-формулы являются конструктивными в том смысле, что по ним производится построение ПЛК-программы, которая соответствует темпоральным свойствам, выраженным этими формулами. Таким образом, программирование ПЛК сводится к построению LTL-спецификации поведения каждой программной переменной. Кроме этого, по LTL-спецификации строится SMV-модель, которая затем проверяется на корректность (относительно дополнительных общепрограммных LTL-свойств) методом проверки модели с помощью средства верификации Cadence SMV.

Поскольку в языке LD все переменные принадлежат к бинарному типу данных, для описания ситуаций, которые приводят к увеличению значения бинарной переменной V, предлагается использовать LTL-формулу вида

$$\mathbf{G}\,\mathbf{X}(\,\neg\_V \wedge V \Rightarrow \mathit{FiringCond}\,), \tag{1}$$

которая означает, что всякий раз, когда новое значение переменной V оказывается больше ее предыдущего значения, записанного в переменной $\_V$, из этого следует, что было выполнено условие соответствующего внешнего воздействия *FiringCond*.

Символ лидирующего подчеркивания «_» в обозначении переменной $\_V$ удобно воспринимать как псевдооператор, позволяющий обратиться к значению переменной V, которое она имела в предыдущем состоянии. При этом псевдооператор может использоваться только под действием темпорального оператора **X**. Условие *FiringCond* является логическим выражением над программными переменными и константами, которые строятся с применением операторов сравнения, логических и арифметических операторов и псевдооператора «_» (который может быть применим только к переменным). Выражение *FiringCond* описывает ситуации, при которых возникает необходимость изменения значения переменной V.

Аналогичным образом описываются ситуации, приводящие к уменьшению значения логической переменной V:

$$\mathbf{G}\,\mathbf{X}(\,\_V \wedge \neg V \Rightarrow \mathit{FiringCond}'\,). \tag{1'}$$

Рассмотрим особый случай спецификаций вида (1) и (1′), при котором для V имеем $\mathit{FiringCond} = \neg\mathit{FiringCond}'$. Такая спецификация может быть заменена на одну LTL-формулу вида

$$\mathbf{G}\,\mathbf{X}(\,V = \mathit{FiringCond}\,). \tag{2}$$

Переменную V из спецификации вида (1) и (1′) будем называть *переменной-регистром*. Если для переменной V строится спецификация вида (2), назовем ее *переменной-функцией*. В особом случае спецификации (2), при котором условие

FiringCond не содержит псевдооператора лидирующего подчеркивания «_», переменную V будем называть *переменной-подстановкой*.

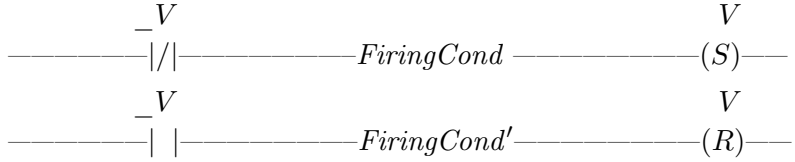
Условие изменения значения переменной только в одном месте программы облегчает отладку, дает возможность простой оценки степени готовности и объема текста программы. При этом стадия написания кода программы заканчивается, как только для каждой выходной и вспомогательной переменной создана спецификация. Отметим, что количество и смысл выходных переменных определяются интерфейсом ПЛК исходя из постановки задачи.

Спецификация программы делится на две части: 1) спецификацию поведения всех программных переменных (кроме входов), 2) спецификацию общепрограммных свойств. При этом вторая составляющая спецификации оказывает влияние на количество и смысл внутренних вспомогательных переменных ПЛК-программы.

При построении спецификации важно учитывать то, в каком порядке располагаются темпоральные формулы, описывающие поведение переменных. Некоторая переменная без псевдооператора «_» может быть задействована в спецификации поведения другой переменной, только если спецификация ее поведения уже произведена и находится выше по тексту.

В спецификации блок темпоральных формул, описывающий поведение некоторой переменной, будем по необходимости сопровождать указанием начального значения этой переменной, которое она получает при инициализации. Для этого задействуется ключевое слово Init. Например, $\mathrm{Init}(V) = 1$ означает, что при инициализации переменная V получает значение 1. Если в спецификации явно не указывается начальное значение переменной, то считается, что это значение равняется нулю.

Рассмотрим способ построения LD-диаграмм по конструктивной LTL-спецификации поведения программных переменных. Схема трансляции LTL-формул в LD-диаграммы следующая. Двум темпоральным формулам для переменной V (выставление сигнала и снятие сигнала) ставятся в соответствие следующие LD-цепи



где *FiringCond* и *FiringCond*′ представляются наборами контактов, которые соединены между собой последовательным и/или параллельным способом. Отметим, что по построению поведение полученной LD-программы будет полностью удовлетворять формулам LTL-спецификации.

Для спецификации поведения переменной-функции V имеем простую цепь вида

Каждая программная переменная должна быть определена в разделе (локальном или глобальном) описания переменных и проинициализирована в соответствии со спецификацией. Отметим, что, например, в среде разработки CoDeSys [8] по умолчанию все переменные инициализируются нулем.

Кроме того, необходимо реализовать идею псевдооператора лидирующего подчеркивания «_». Для этого в самом конце программы выделяется место для псевдооператорного раздела, куда после задания поведения всех переменных специфи-

кации для каждой такой переменной V, к прошлому значению которой обращались как $\_V$, строится цепь, реализующая присваивание $\_V := V$.

При этом переменную $\_V$ также необходимо определить в разделе описания переменных с такой же инициализацией, как и для переменной V.

Отметим, что подход к программированию по спецификации, которая, по сути, описывает причину изменения значения каждой программной переменной, выглядит естественным и оправданным, поскольку выходной сигнал ПЛК является управляющим, а смена значения этого управляющего сигнала обычно несет в себе дополнительную смысловую нагрузку. Например, важно четко представлять себе, почему двигатель должен быть запущен/выключен, а некоторая лампа зажжена/погашена. Поэтому кажется очевидным, что каждая переменная должна сопровождаться двумя свойствами, по одному на каждое направление изменений. При этом предполагается, что если условия изменений не выполнены, то переменная сохраняет свое прежнее значение.

В качестве программного средства анализа корректности методом проверки модели мы рассматриваем верификатор Cadence SMV [10]. После создания спецификации предлагается осуществлять построение модели Крипке на языке SMV с последующей проверкой выполнимости для этой модели общепрограммных свойств. Если некоторое общепрограммное свойство не выполняется для модели, то верификатор строит пример некорректного пути в модели Крипке, по которому вводятся исправления в спецификацию. Когда все программные свойства проверены с положительным результатом, по спецификации строится LD-программа ПЛК.

Средства языка SMV позволяют с помощью оператора next задавать значение переменной в следующем состоянии модели Крипке, т. е. на следующем шаге (после разового прохождения рабочего цикла ПЛК), относительно новых входных сигналов и последних значений выходов и внутренних переменных. Оператор next позволяет также обращаться к значению переменной, которое уже было вычислено для нового (следующего) состояния модели. Ветвление отношения переходов обеспечивается «недетерминированным» присваиванием. Например, присваивание вида $\mathrm{next}(V) := \{0, 1\}$ означает, что будут порождены состояния и переходы в них как со значением входа $V = 0$, так и со значением $V = 1$. На языке SMV символы «&», «|», «$\sim$» и «->» означают соответственно логические «и», «или», «не» и импликацию.

Язык SMV ориентирован на построение следующих состояний модели Крипке из текущего состояния. Начальным текущим состоянием модели является состояние программы после инициализации. Поэтому спецификацию поведения переменной V (1) и (1′) будет удобнее переписать в следующем равнозначном виде

$$\mathbf{G}(\,\neg V \wedge \mathbf{X}(V) \Rightarrow \mathbf{X}(\mathit{FiringCond})\,), \qquad \mathbf{G}(\,V \wedge \mathbf{X}(\neg V) \Rightarrow \mathbf{X}(\mathit{FiringCond}')\,).$$

И уже затем, ставя в соответствие темпоральному оператору $\mathbf{X}$ оператор next, естественным образом получаем SMV-модель поведения переменной V:

$$\begin{aligned}\mathrm{case}\{\ &\sim V\ \&\ \mathrm{next}(\mathit{FiringCond}\ ) &&: \mathrm{next}(V) := 1;\\ &V\ \&\ \mathrm{next}(\mathit{FiringCond}') &&: \mathrm{next}(V) := 0;\\ &\mathrm{default} &&: \mathrm{next}(V) := V;\ \}.\end{aligned}$$

Здесь ключевое слово default соответствует тому, что должно происходить, если не выполняются условия срабатывания первых двух веток операторного блока case.

В случае с переменной-функцией V при спецификации (2) SMV-модель ее поведения задается как $\mathrm{next}(V) := \mathrm{next}(\mathit{FiringCond})$. Поведение переменной-подстановки описывается просто в виде присваивания $V := \mathit{FiringCond}$.

Верификатор Cadence SMV позволяет проверять программные модели, содержащие до 59 двоичных переменных (переменные других типов представляются наборами двоичных переменных). При этом переменные-подстановки в это число не включаются, т. е. считаются только переменные-регистры и переменные-функции.

3. Библиотечный подъемник

Библиотечный подъемник, схема которого представлена на рис. 1, предназначен для подъема книг по требованию (заполненному формуляру требования) из хранилища подвального этажа на первый и второй этажи библиотеки и возврата книг на подвальный этаж. Вызов кабины подъемника на верхние этажи и отправка кабины с верхних этажей на подвальный этаж осуществляется нажатием кнопок «Вызов 2», «Вызов 1», «Вниз 2» и «Вниз 1» соответственно. Отправка кабины на верхние этажи из подвального этажа производится нажатием кнопок «Вверх 2» и «Вверх 1». Если соответствующая нажатой кнопке команда была принята, зажигается лампа этой кнопки, которая гасится после выполнения команды.

При заходе кабины на заданный этаж включается лампа этого этажа «Этаж 2», «Этаж 1» или «Этаж 0». Кабина подъемника своей двери не имеет, а есть только двери шахты, которые открываются и закрываются вручную. Датчики «Д2», «Д1» и «Д0» служат для определения положения дверей. Датчик двери выдает сигнал, если дверь закрыта. Когда дверь открыта, сигнал снимается. Визуально сигнал от датчика двери определяется с помощью лампы этого датчика.

Датчик этажей «ДЭ», который располагается на кабине подъемника, используются для определения положения кабины в шахте. Датчик этажа выставляет сигнал только тогда, когда кабина полностью находится на том или ином этаже, реагируя на специальную металлическую пластину. В противном случае сигнал снимается.

Управление библиотечным подъемником осуществляется с помощью ПЛК, получающего входные сигналы от датчиков и кнопок и подающего выходные сигналы на двигатель подъемника и лампы (см. рис. 1).

Задача состоит в написании программы для ПЛК с 10 входами и 14 выходами, предназначенного для управления подъемником. Интерфейс ПЛК управления библиотечным подъемником представлен на рис. 1.

Программа ПЛК должна удовлетворять следующим требованиям.

1. Начальное расположение кабины подъемника в шахте — подвальный этаж.

2. Выходы контроллера на лампы сигнализации положения дверей должны передавать входные сигналы соответствующих датчиков дверей.

3. Лампы «Этаж 2», «Этаж 1» и «Этаж 0» должны гореть тогда и только тогда, когда двигатель подъемника выключен, а кабина подъемника находится точно на втором, первом и подвальном этажах соответственно.

4. Команда «Вверх на этаж 1» принимается и загорается лампа подвального этажа «Вверх 1», если кабина подъемника находится на подвальном этаже и на этом

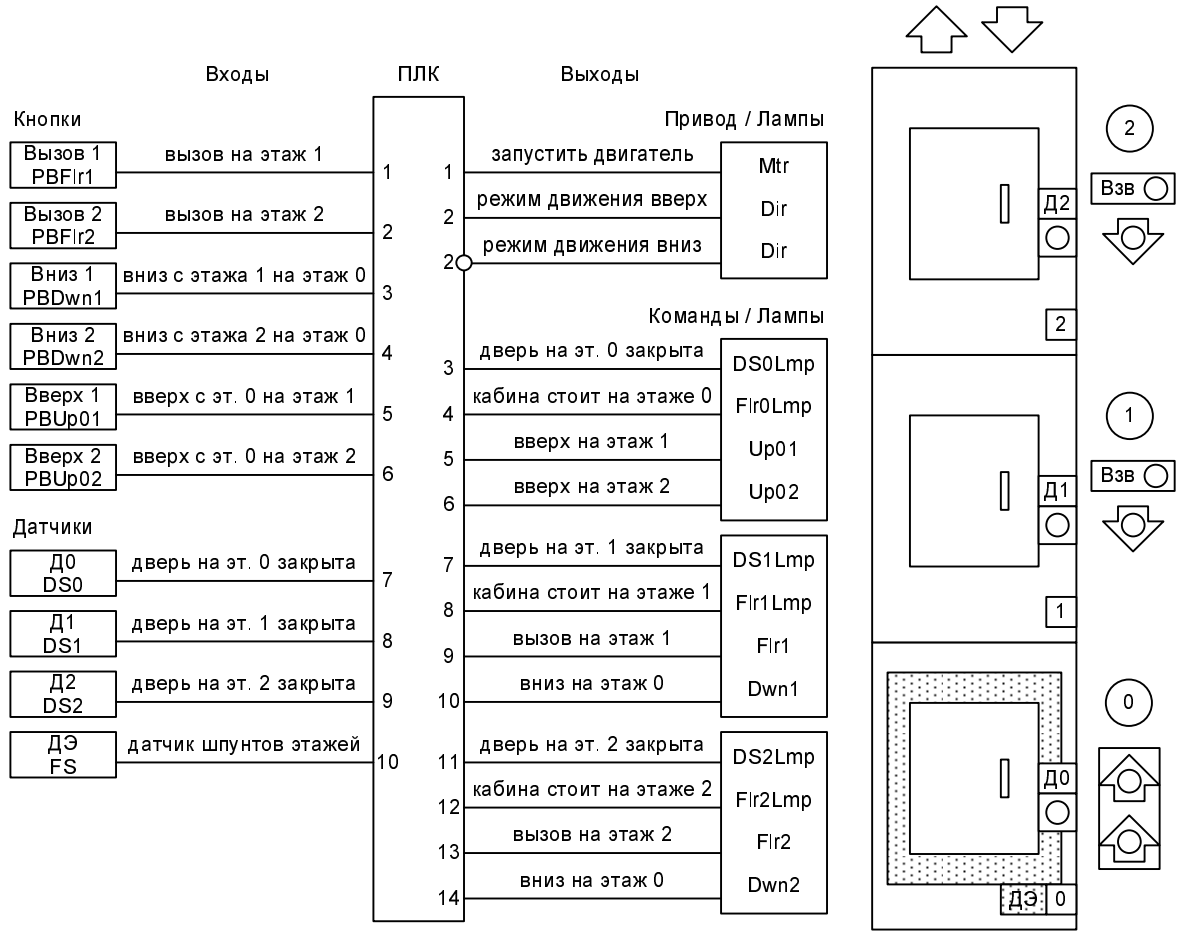


Рис. 1. Интерфейс ПЛК управления и схема библиотечного подъемника

этаже нажата кнопка «Вверх 1». Команда считается выполненной и соответствующая лампа гасится, если кабина находится на первом этаже при выключенном двигателе подъемника. Аналогично исполняется команда «Вверх на этаж 2».

5. Команда вызова кабины на первый этаж принимается и загорается лампа «Вызов 1», если нажата кнопка «Вызов 1». Команда считается выполненной, если кабина находится на первом этаже при выключенном двигателе подъемника. Аналогичным образом исполняется команда вызова кабины на второй этаж.

6. Команда отправки кабины с первого этажа на подвальный этаж принимается и загорается лампа первого этажа «Вниз 1», если двигатель подъемника выключен, кабина находится на первом этаже и либо нажата кнопка «Вниз 1», либо кабина находится на первом этаже с закрытой дверью более 10 секунд. Команда считается выполненной, если кабина находится на подвальном этаже при выключенном двигателе подъемника. Аналогичным образом принимается и выполняется команда отправки кабины со второго этажа на подвальный этаж.

7. Выходной сигнал контроллера на двигатель подъемника «Режим движения вверх» выставляется, если кабина подъемника находится на подвальном этаже и двигатель выключен. Этот сигнал снимается, т. е. выставляется инверсный сигнал «Режим движения вниз», если двигатель выключен и либо кабина подъемника находится на втором этаже, либо 1) кабина находится на первом этаже, 2) принята

команда движения вниз с первого этажа, 3) нет команды движения на второй этаж с подвального и 4) или нет вызова на второй этаж, или принята команда двигаться вниз со второго этажа на подвальный.

8. Двигатель подъемника запускается в установленном режиме, если двери на всех этажах закрыты и либо 1) кабина не находится точно на этаже и есть хотя бы одна еще не выполненная команда, либо 2) кабина находится точно на подвальном этаже и поступили команды движения с подвального этажа вверх, либо 3) кабина находится точно на подвальном этаже с закрытой дверью более 10 секунд и поступили вызовы с верхних этажей, либо 4) кабина находится точно на первом этаже и поступила команда двигаться вниз с первого на подвальный этаж, либо 5) кабина находится точно на втором этаже и поступила команда двигаться вниз со второго на подвальный этаж.

9. Двигатель подъемника останавливается, если либо 1) одна из дверей открыта, либо 2) в режиме движения вниз кабина находится точно на подвальном этаже, либо 3) в режиме движения вверх кабина находится точно на втором этаже, либо 4) кабина находится точно на первом этаже и или имеется вызов на первый этаж, но нет команды двигаться вниз на подвальный, или имеется команда двигаться вверх с подвального этажа на первый.

3.1. Программные свойства и вспомогательные переменные

Глобальные переменные программы определяются интерфейсом ПЛК. Кроме необходимости реализации алгоритма решения задачи, введение вспомогательных переменных во многом диктуется необходимостью выражать на языке темпоральной логики LTL общепрограммные свойства, вытекающие из постановки задачи. В данном разделе в качестве примера рассматривается ряд общепрограммных свойств для задачи «Библиотечный подъемник», которые затем задаются в виде LTL-формул с опорой на вводимые по необходимости вспомогательные переменные.

Важно отметить, что описание некоторых внутренних переменных, а также таймеров, следует непосредственно из постановки задачи. Например, для понимания того, на каком этаже находится кабина подъемника, введем соответствующие «флаги» этажей *Ctr0*, *Ctr1* и *Ctr2* (в совокупности образующие счетчик этажей). Другие внутренние переменные могут появляться исходя из удобства программирования и читаемости программы. Например, $DS = DS0 \wedge DS1 \wedge DS2$.

Явными общепрограммными свойствами в случае с библиотечным подъемником являются те свойства, которые выражают обязательность выполнения команд вызова/отправки на этажи. Прежде чем рассмотреть примеры общепрограммных свойств, по описанию задачи построим конструктивную спецификацию программы управления библиотечным подъемником в соответствии с концепцией программирования и верификации по LTL-спецификации.

```
Ctr0+: GX(~_Ctr0 &  Ctr0 -> FS & ~_FS & _Ctr1 & ~_Dir);
Ctr0-: GX( _Ctr0 & ~Ctr0 -> FS & ~_FS );
Ctr1+: GX(~_Ctr1 &  Ctr1 -> FS & ~_FS & (_Ctr2 & ~_Dir | _Ctr0 & _Dir));
Ctr1-: GX( _Ctr1 & ~Ctr1 -> FS & ~_FS ); init(Ctr1)=1;
Ctr2+: GX(~_Ctr2 &  Ctr2 -> FS & ~_FS & _Ctr1 &  _Dir);
Ctr2-: GX( _Ctr2 & ~Ctr2 -> FS & ~_FS );
Up01+: GX(~_Up01 &  Up01 -> FS & Ctr0 & PBUp01);
```

```
Up01-: GX( _Up01 & ~Up01 -> FS & Ctr1 & ~_Mtr );
Up02+: GX(~_Up02 &  Up02 -> FS & Ctr0 & PBUp02);
Up02-: GX( _Up02 & ~Up02 -> FS & Ctr2 & ~_Mtr );
Dwn2+: GX(~_Dwn2 &  Dwn2 -> FS & Ctr2 & ~_Mtr & (PBDwn2 | _Tmr.Q));
Dwn2-: GX( _Dwn2 & ~Dwn2 -> FS & Ctr0 & ~_Mtr);
Dwn1+: GX(~_Dwn1 &  Dwn1 -> FS & Ctr1 & ~_Mtr & (PBDwn1 | _Tmr.Q));
Dwn1-: GX( _Dwn1 & ~Dwn1 -> FS & Ctr0 & ~_Mtr );  Init(Dwn1)=1;
Flr2+: GX(~_Flr2 &  Flr2 -> PBFlr2);  Flr2-: GX( _Flr2 & ~Flr2 -> FS & Ctr2 & ~_Mtr);
Flr1+: GX(~_Flr1 &  Flr1 -> PBFlr1);  Flr1-: GX( _Flr1 & ~Flr1 -> FS & Ctr1 & ~_Mtr);
Dir+: GX(~_Dir &  Dir -> FS & Ctr0 & ~_Mtr);
Dir-: GX( _Dir & ~Dir -> FS & (Ctr2 | Ctr1 & Dwn1 & ~Up02 & (~Flr2 | Dwn2)) & ~_Mtr);
DS:   GX(DS = DS0 & DS1 & DS2);
Mtr+: GX(~_Mtr &  Mtr -> DS & (~FS & (Dwn1 | Dwn2 | Up01 | Up02 | Flr1 | Flr2)     |
                               FS & Ctr0 & (Up01 | Up02 | _Tmr.Q & (Flr1 | Flr2)) |
                               FS & Ctr1 & Dwn1 | FS & Ctr2 & Dwn2)               );
Mtr-: GX( _Mtr & ~Mtr -> (~DS | FS & Ctr0 & ~_Dir | FS & Ctr2 &  _Dir |
                               FS & Ctr1 & (Flr1 & ~Dwn1 | Up01))     );
Tmr.In:  GX(Tmr.In  = ~Mtr & FS & (Ctr0 & DS0 | Ctr1 & DS1 | Ctr2 & DS2));
Flr0Lmp: GX(Flr0Lmp = ~Mtr & FS & Ctr0);  DS0Lmp: GX(DS0Lmp = DS0);
Flr1Lmp: GX(Flr1Lmp = ~Mtr & FS & Ctr1);  DS1Lmp: GX(DS1Lmp = DS1);
Flr2Lmp: GX(Flr2Lmp = ~Mtr & FS & Ctr2);  DS2Lmp: GX(DS2Lmp = DS2).
```

Далее приведены примеры общепрограммных свойств программы управления библиотечным подъемником.

1. Свойство $\mathbf{G}(\mathit{Ctr0} + \mathit{Ctr1} + \mathit{Ctr2} = 1)$ означает, что в любой момент времени кабина лифта обязательно находится только на одном этаже.

2. «Выход за пределы шахты». Не существует ситуаций, при которых кабина лифта находится точно на подвальном этаже, а двигатель подъемника включен в режиме спуска: $\mathbf{G}\neg(\mathit{FS} \wedge \mathit{Ctr0} \wedge \mathit{Dir} = 0 \wedge \mathit{Mtr})$. Не существует ситуаций, при которых кабина лифта находится точно на втором этаже, а двигатель подъемника включен в режиме подъема: $\mathbf{G}\neg(\mathit{FS} \wedge \mathit{Ctr2} \wedge \mathit{Dir} = 1 \wedge \mathit{Mtr})$.

3. «Движение». Всегда верно, что если двигатель подъемника включен (работает), то двери шахты закрыты: $\mathbf{G}(\mathit{Mtr} \Rightarrow \mathit{DS})$. И если двери шахты закрыты и кабина не находится точно на одном из этажей, то двигатель подъемника включен (работает): $\mathbf{G}(\mathit{DS} \wedge \neg\mathit{FS} \Rightarrow \mathit{Mtr})$. Не существует ситуаций, при которых двигатель подъемника выключен, двери шахты закрыты, а кабина не находится точно на одном из этажей: $\mathbf{G}\neg(\neg\mathit{Mtr} \wedge \mathit{DS} \wedge \neg\mathit{FS})$.

4. «Выключение двигателя». Всякий раз, когда двигатель подъемника запускается, то обязательно рано или поздно он будет выключен, т. е. двигатель не может работать бесконечно долго: $\mathbf{G}(\mathit{Mtr} \Rightarrow \mathbf{F}(\neg\mathit{Mtr}))$.

5. «Выполнение команды». Если рассмотреть только те программные исполнения, при которых после поступления некоторой команды вызова/отправки на этаж *Cmd* до ее выполнения двери шахты будут открываться/закрываться только конечное число раз, то всегда при поступлении команды *Cmd* она рано или поздно будет выполнена, где *Cmd* — это *Flr1*, *Flr2*, *Up01*, *Up02*, *Dwn1* или *Dwn2*: $\mathbf{G}(\mathit{Cmd} \Rightarrow \neg\mathbf{G}(\mathit{Cmd}\ \mathbf{U}\ \neg\mathit{DS})) \Rightarrow \mathbf{G}(\mathit{Cmd} \Rightarrow \mathbf{F}(\neg\mathit{Cmd}))$.

По конструктивной спецификации далее строится SMV-модель программы (см. следующий раздел), для которой проверяется справедливость общепрограммных свойств. После проверки свойств происходит построение (также по конструктивной LTL-спецификации) LD-программы ПЛК, которая представлена на рис. 2, 3 и 4.

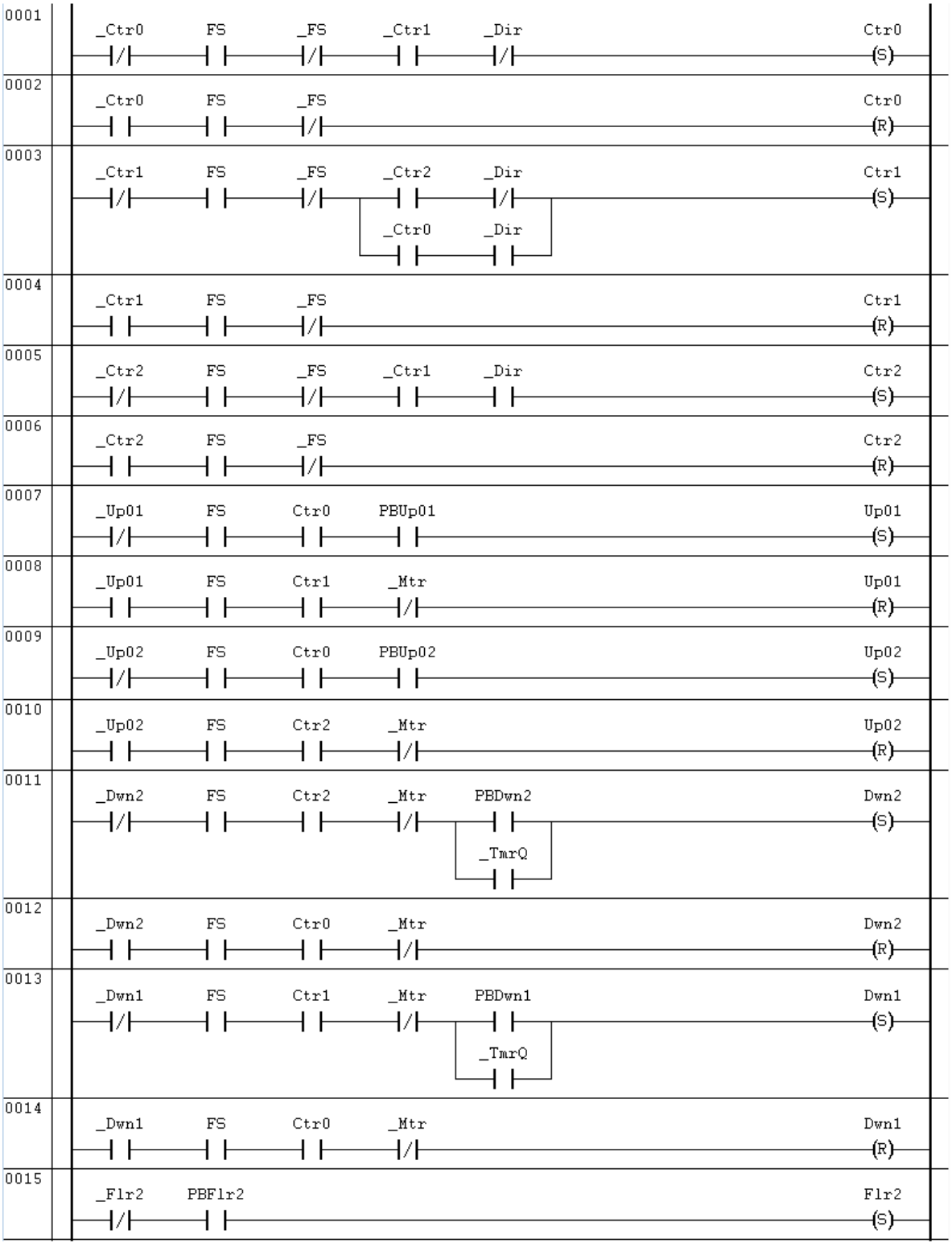


Рис. 2. LD-программа ПЛК управления библиотечным подъемником

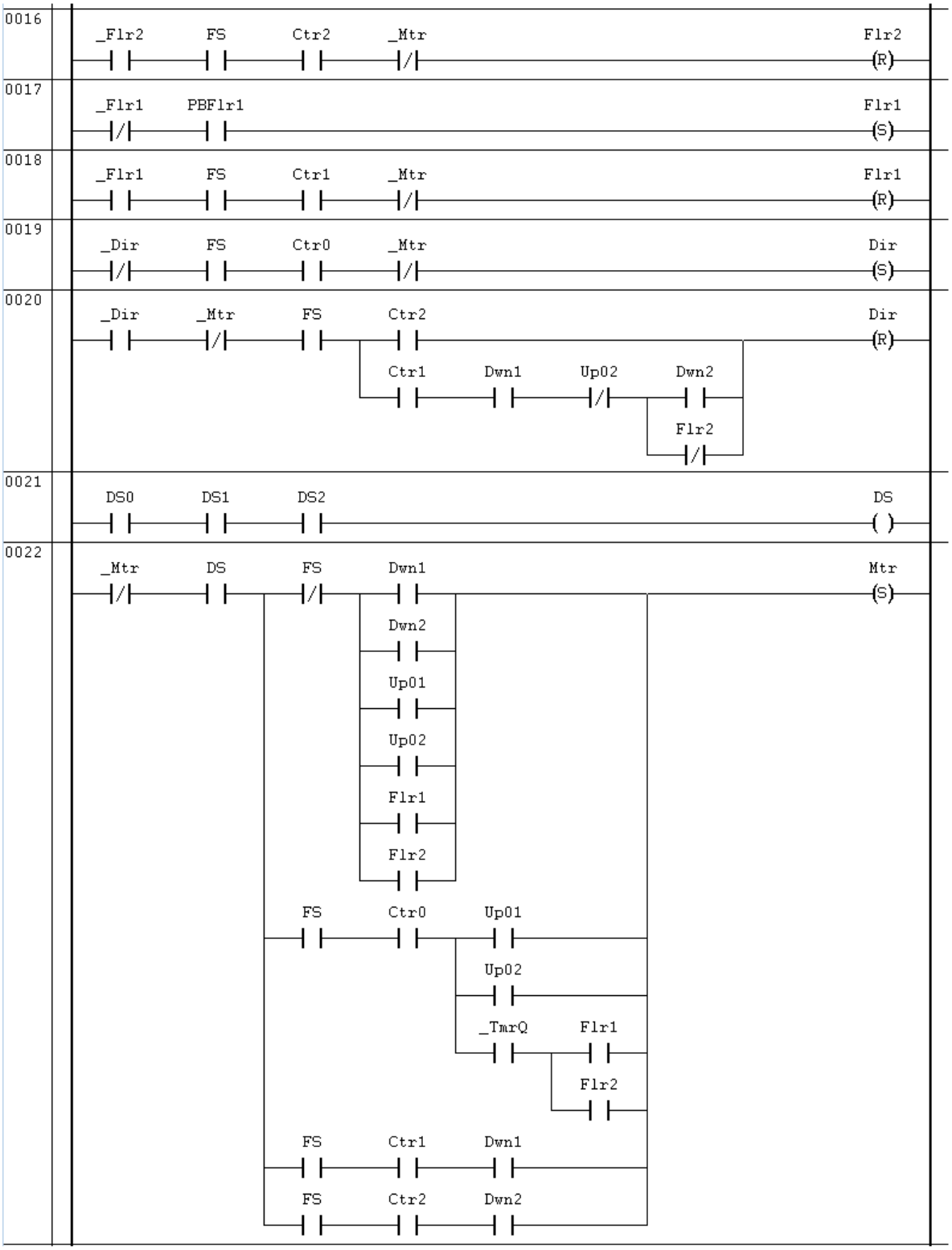


Рис. 3. LD-программа ПЛК управления библиотечным подъемником (продолжение)

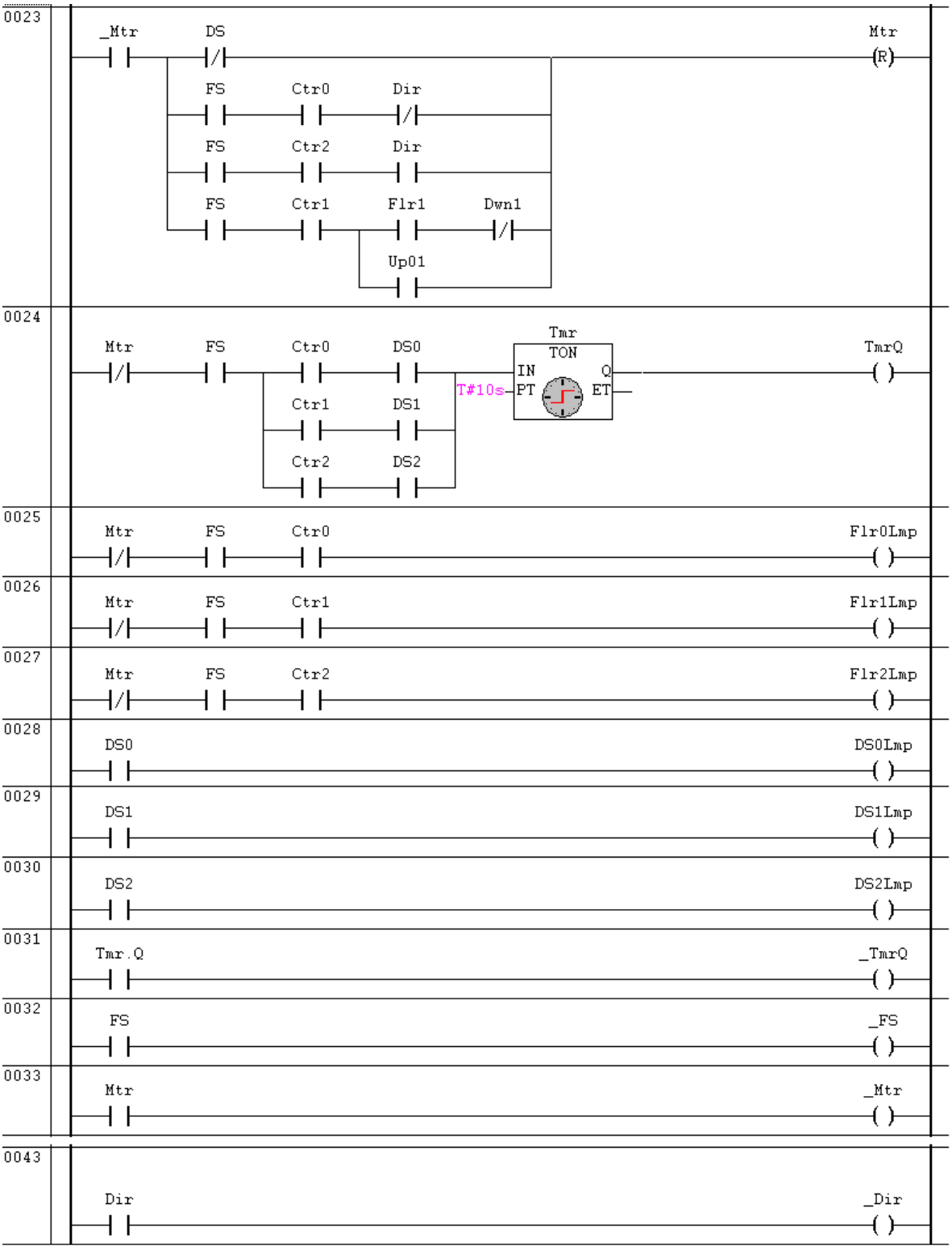


Рис. 4. LD-программа ПЛК управления библиотечным подъемником (продолжение)

3.2. SMV-модель программы библиотечного подъемника

Моделирование работы таймера подробно описывается в статье [3]. Поясним модель поведения датчика этажей FS. При включенном двигателе подъемника значение датчика FS может оставаться неизменным или меняться на противоположное. При выключенном двигателе датчик в следующем состоянии системы выдает прежнее значение. Условия честности FAIRNESS для датчика FS означают, что при включенном двигателе этот датчик не может выставлять и удерживать бесконечно долго без изменения один и тот же сигнал.

```
module timer(){
  I : 0..1;   /* вход  */
  Q : 0..1;   /* выход */
  init(I):=0; init(Q):=0; /* инициализация */
  next(Q):= next(I) & (Q | {0, 1}); /* новое значение выхода */
  FAIRNESS I -> Q;  /* условие честного срабатывания таймера */
}
module main(){ /* Раздел описания переменных */  /* входы */
  PBFlr2, PBDwn2, PBFlr1, PBDwn1, PBUp02, PBUp01: 0..1;      /* кнопки  */
  DS2, DS1, DS0, FS: 0..1;                                   /* датчики */
     /* выходы */
  Mtr, Dir: 0..1;                                            /* привод  */
  Flr2Lmp, Flr1Lmp, Flr0Lmp, DS2Lmp, DS1Lmp, DS0Lmp: 0..1;   /* лампы   */
  Flr2, Flr1, Dwn2, Dwn1, Up02, Up01: 0..1;                  /* команды */
     /* внутренние */
  Ctr0, Ctr1, Ctr2: 0..1;                                 /* флаги этажей */
  DS: 0..1;                                            /* состояние дверей */
  Tmr : timer;  /* таймер */
 /* Раздел инициализации */
     /* входы */
  init(PBFlr2):=0; init(PBDwn2):=0; init(PBUp02):=0; init(DS1):=0; init(DS0):=0;
  init(PBFlr1):=0; init(PBDwn1):=0; init(PBUp01):=0; init(DS2):=0;  init(FS):=0;
     /* выходы */
  init(Mtr):=0; init(Flr1):=0; init(Dwn1):=1; init(Up01):=0;
  init(Dir):=0; init(Flr2):=0; init(Dwn2):=0; init(Up02):=0;
     /* внутренние */
  init(Ctr0):=0; init(Ctr1):=1; init(Ctr2):=0;
 /* Система переходов */
     /* входы */
  next(PBFlr2):={0, 1}; next(PBDwn2):={0, 1};
  next(PBFlr1):={0, 1}; next(PBDwn1):={0, 1};
  next(PBUp02):={0, 1}; next(PBUp01):={0, 1};
  next(DS2):={0, 1}; next(DS1):={0, 1}; next(DS0):={0, 1};
  case{ Mtr     : next(FS):={0,1};
        default : next(FS):=FS;};
  FAIRNESS Mtr ->  FS;
  FAIRNESS Mtr -> ~FS;
     /* выходы и внутренние переменные */
  case{~Ctr0 & next(FS) & ~FS & Ctr1 & ~Dir: next(Ctr0):=1;          /* Ctr0+ */
        Ctr0 & next(FS) & ~FS              : next(Ctr0):=0;          /* Ctr0- */
       default                             : next(Ctr0):=Ctr0;  };
  case{~Ctr1 & next(FS) & ~FS & (Ctr2 & ~Dir | Ctr0 & Dir)
                                           : next(Ctr1):=1;          /* Ctr1+ */
        Ctr1 & next(FS) & ~FS              : next(Ctr1):=0;          /* Ctr1- */
       default                             : next(Ctr1):=Ctr1;  };
```

```
  case{~Ctr2 & next(FS) & ~FS & Ctr1 &  Dir: next(Ctr2):=1;          /* Ctr2+ */
        Ctr2 & next(FS) & ~FS              : next(Ctr2):=0;          /* Ctr2- */
       default                             : next(Ctr2):=Ctr2;  };
  case{~Up01 & next(FS) & next(Ctr0) & next(PBUp01): next(Up01):=1;  /* Up01+ */
        Up01 & next(FS) & next(Ctr1) & ~Mtr        : next(Up01):=0;  /* Up01- */
       default                                     : next(Up01):=Up01;  };
  case{~Up02 & next(FS) & next(Ctr0) & next(PBUp02): next(Up02):=1;  /* Up02+ */
        Up02 & next(FS) & next(Ctr2) & ~Mtr        : next(Up02):=0;  /* Up02- */
       default                                     : next(Up02):=Up02;  };
  case{~Dwn2 & next(FS) & next(Ctr2) & ~Mtr & (next(PBDwn2) | Tmr.Q)
                                                   : next(Dwn2):=1;  /* Dwn2+ */
        Dwn2 & next(FS) & next(Ctr0) & ~Mtr        : next(Dwn2):=0;  /* Dwn2- */
       default                                     : next(Dwn2):=Dwn2;  };
  case{~Dwn1 & next(FS) & next(Ctr1) & ~Mtr & (next(PBDwn1) | Tmr.Q)
                                                   : next(Dwn1):=1;  /* Dwn1+ */
        Dwn1 & next(FS) & next(Ctr0) & ~Mtr        : next(Dwn1):=0;  /* Dwn1- */
       default                                     : next(Dwn1):=Dwn1;  };
  case{~Flr2 & next(PBFlr2)                        : next(Flr2):=1;  /* Flr2+ */
        Flr2 & next(FS) & next(Ctr2) & ~Mtr        : next(Flr2):=0;  /* Flr2- */
       default                                     : next(Flr2):=Flr2;  };
  case{~Flr1 & next(PBFlr1)                        : next(Flr1):=1;  /* Flr1+ */
        Flr1 & next(FS) & next(Ctr1) & ~Mtr        : next(Flr1):=0;  /* Flr1- */
       default                                     : next(Flr1):=Flr1;  };
  case{~Dir & next(FS) & next(Ctr0) & ~Mtr         : next(Dir):=1;    /* Dir+ */
        Dir & next(FS) & (next(Ctr2) | next(Ctr1) & next(Dwn1) & ~next(Up02) &
              (~next(Flr2) | next(Dwn2))) & ~Mtr   : next(Dir):=0;    /* Dir- */
       default                                     : next(Dir):=Dir;  };
  DS:= DS0 & DS1 & DS2;  /* DS */
  case{~Mtr & next(DS) &
              (~next(FS) & (next(Dwn1) | next(Dwn2) | next(Up01) |
                            next(Up02) | next(Flr1) | next(Flr2)) |
                            next(FS) & next(Ctr0) & (next(Up01) | next(Up02) |
                                        Tmr.Q & (next(Flr1) | next(Flr2) ))  |
                            next(FS) & next(Ctr1) &  next(Dwn1)              |
                            next(FS) & next(Ctr2) &  next(Dwn2) )
                                                   : next(Mtr):=1;    /* Mtr+ */
        Mtr &(~next(DS) | next(FS) & next(Ctr0) & ~Dir |
              next(FS) & next(Ctr2) &  Dir |
              next(FS) & next(Ctr1) & (next(Flr1) & ~next(Dwn1) | next(Up01)))
                                                   : next(Mtr):=0;    /* Mtr- */
       default                                     : next(Mtr):=Mtr;  };
  next(Tmr.I):= next(~Mtr & FS & (Ctr0 & DS0 | Ctr1 & DS1 | Ctr2 & DS2));
  Flr0Lmp:= ~Mtr & FS & Ctr0;  /* Flr0Lmp */
  Flr1Lmp:= ~Mtr & FS & Ctr1;  /* Flr1Lmp */
  Flr2Lmp:= ~Mtr & FS & Ctr2;  /* Flr2Lmp */
  DS0Lmp:= DS0; DS1Lmp:= DS1; DS2Lmp:= DS2;  /* DS0Lmp, DS1Lmp, DS2Lmp  */
/* Раздел свойств */
  P_Ctr: assert G(Ctr0+Ctr1+Ctr2 = 1);
  P_Limit0: assert G~(FS & Ctr0 & Dir=0 & Mtr);
  P_Limit2: assert G~(FS & Ctr2 & Dir=1 & Mtr);
  P_Doors: assert G(Mtr -> DS);
  P_Stop: assert G~(~Mtr & DS & ~FS);
  P_Mtr: assert G( Mtr -> F(~Mtr));
  P_Flr2: assert G(Flr2 -> ~G(Flr2 U ~DS)) -> G(Flr2 -> F(~Flr2));
  P_Flr1: assert G(Flr1 -> ~G(Flr1 U ~DS)) -> G(Flr1 -> F(~Flr1));
  P_Up01: assert G(Up01 -> ~G(Up01 U ~DS)) -> G(Up01 -> F(~Up01));
```

```
  P_Up02: assert G(Up02 -> ~G(Up02 U ~DS)) -> G(Up02 -> F(~Up02));
  P_Dwn1: assert G(Dwn1 -> ~G(Dwn1 U ~DS)) -> G(Dwn1 -> F(~Dwn1));
  P_Dwn2: assert G(Dwn2 -> ~G(Dwn2 U ~DS)) -> G(Dwn2 -> F(~Dwn2));
  P_Move: assert G(DS & ~FS -> Mtr);
}
```

Список литературы

1. *Кузьмин Е. В.*, *Соколов В. А.*, *Рябухин Д. А.* Построение и верификация ПЛК-программ по LTL-спецификации // Моделирование и анализ информационных систем. 2013. Т. 20, №4. С. 5–22. (*Kuzmin E. V.*, *Sokolov V. A.*, *Ryabukhin D. A.* Construction and Verification of PLC-programs by LTL-specification // Modeling and analysis of information systems. 2013. V. 20, №4. P. 5–22 [in Russian]).

2. *Кузьмин Е. В.*, *Рябухин Д. А.*, *Шипов А. А.* Построение и верификация ПЛК-программ по LTL-спецификации // Международная научно-практическая конференция «Инструменты и методы анализа программ». Кострома, КГТУ, 2013. С. 17–34. (*Kuzmin E. V.*, *Ryabukhin D. A.*, *Shipov A. A.* Construction and Verification of PLC-programs by LTL-specification // Proc. of Int. Conf. «Tools and Methods of Program Analysis (TMPA-2013)». Kostroma, KSTU, 2013. P. 17–34 [in Russian]).

3. *Кузьмин Е. В.*, *Соколов В. А.* Моделирование, спецификация и построение программ логических контроллеров // Моделирование и анализ информационных систем. 2013. Т. 20, №2. С. 104–120. (*Kuzmin E. V.*, *Sokolov V. A.* Modeling, Specification and Construction of PLC-programs // Modeling and analysis of information systems. 2013. V. 20, №2. P. 104–120 [in Russian]).

4. *Кузьмин Е. В.*, *Соколов В. А.* О построении и верификации программ логических контроллеров // Моделирование и анализ информационных систем. 2012. Т. 19, №4. С. 25–36. (*Kuzmin E. V.*, *Sokolov V. A.* On Construction and Verification of PLC-Programs // Modeling and analysis of information systems. 2012. V. 19, №4. P. 25–36 [in Russian]).

5. *Кузьмин Е. В.*, *Соколов В. А.* О верификации LD-программ логических контроллеров // Моделирование и анализ информационных систем. 2012. Т. 19, №2. С. 138–144. (*Kuzmin E. V.*, *Sokolov V. A.* On Verification of PLC-Programs Written in the LD-Language // Modeling and analysis of information systems. 2012. V. 19, №2. P. 138–144 [in Russian]).

6. *Петров И. В.* Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования. М.: СОЛОН-Пресс, 2004. 256 с. (*Petrov I. V.* Programmiruemye kontrollery. Standartnye jazyki i priemy prikladnogo proektirovanija. M.: SOLON-Press, 2004. 256 p. [in Russian]).

7. *Clark E. M.*, *Grumberg O.*, *Peled D. A.* Model Checking. The MIT Press, 2001.

8. CoDeSys. Controller Development System. http://www.3s-software.com/

9. *Parr E. A.* Programmable Controllers. An engineer's guide. Newnes, 2003. 442 p.

10. SMV. The Cadence SMV Model Checker. http://www.kenmcmil.com/smv.html

Construction and Verification of PLC LD-programs by LTL-specification

Kuzmin E. V., Sokolov V. A., Ryabukhin D. A.

P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia

Keywords: programmable logic controllers, software engineering, specification and verification of PLC-programs, Ladder Diagram

An approach to construction and verification of PLC LD-programs for discrete problems is proposed. For the specification of the program behavior, we use the linear-time temporal logic LTL. Programming is carried out in the LD-language (Ladder Diagram) according to an LTL-specification. The correctness analysis of an LTL-specification is carried out by the symbolic model checking tool Cadence SMV. A new approach to programming and verification of PLC LD-programs is shown by an example. For a discrete problem, we give a LD-program, its LTL-specification and an SMV-model.

The purpose of the article is to describe an approach to programming PLC, which would provide a possibility of LD-program correctness analysis by the model checking method.

Under the proposed approach, the change of the value of each program variable is described by a pair of LTL-formulas. The first LTL-formula describes situations which increase the value of the corresponding variable, the second LTL-formula specifies conditions leading to a decrease of the variable value. The LTL-formulas (used for specification of the corresponding variable behavior) are constructive in the sense that they construct the PLC-program (LD-program), which satisfies temporal properties expressed by these formulas. Thus, the programming of PLC is reduced to the construction of LTL-specification of the behavior of each program variable. In addition, an SMV-model of a PLC LD-program is constructed according to LTL-specification. Then, the SMV-model is analysed by the symbolic model checking tool Cadence SMV.

Сведения об авторах:
Кузьмин Егор Владимирович,
Ярославский государственный университет им. П.Г. Демидова,
д-р физ.-мат. наук, профессор;
Соколов Валерий Анатольевич,
Ярославский государственный университет им. П.Г. Демидова,
д-р физ.-мат. наук, профессор;
Рябухин Дмитрий Александрович,
Ярославский государственный университет им. П.Г. Демидова,
аспирант.