УДК 681.31

А.Д. Иванников

АЛГЕБРАИЧЕСКАЯ МОДЕЛЬ ПРОГРАММНО-МИКРОПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ЦИФРОВЫХ СИСТЕМ

На основе алгебраического подхода формализуется понятие оператора над памятью цифровых систем, вводится понятие обобщенной памяти, включающей кроме собственно памяти и регистров состояние активной метки оператора и также значение времени, формализуется функционал оператора над обобщенной памятью. Проводится анализ структуры области определения и области значений оператора. Вводится понятие произведения операторов, формулируется условие существования введенного произведения. Показывается, что программа цифровой системы является произведением операторов, множество которых является полугруппой. Разработанную формальную модель предполагается использовать для создания эффективной системы автоматизированной генерации тестовых примеров для программного обеспечения цифровых систем.

Ключевые слова: *модель программного обеспечения, программа как произведение операторов, алгебраическая модель, операторы над обобщенной памятью.*

Введение. При проектировании современных цифровых систем в виде схемы соединения некоторых блоков и текста программного (или микропрограммного) обеспечения, которое осуществляется с использованием той или иной системы автоматизированного проектирования, весьма важной является задача отладки проектов [1].

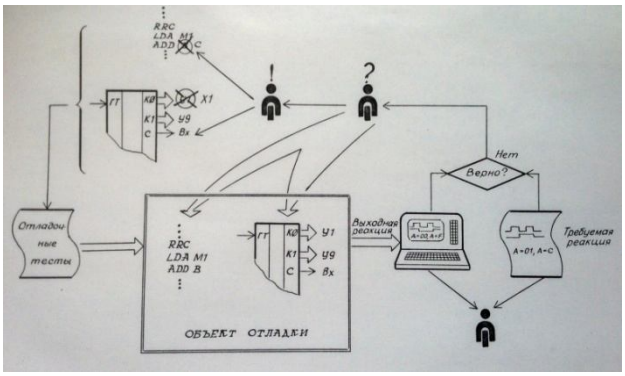


Рисунок 1 – Упрощенная схема отладки цифровой системы

Отладка весьма часто осуществляется с использованием ЭВМ путем подачи на модель цифровой системы некоторого множества входных взаимодействий [2] и анализа результирующих выходных воздействий на управляемые объекты или процессы, а также анализа внутренних состояний отлаживаемой цифровой системы (рисунок 1).

В связи с большой размерностью решения задачи отладки в общем, при проектировании сложных цифровых систем используются методы декомпозиции этой задачи [3]. При этом отладка осуществляется иерархически, то есть при использовании более подробных моделей компонентов цифровой системы на модель последней подаются более короткие тестовые последовательности, проверяющие правильность выполнения коротких базовых операций, а при использовании более грубых моделей компонентов осуществляется отладка последовательности операций.

В соответствии с принципом вертикальной структурной декомпозиции [3] программно-микропрограммное обеспечение является интерпретатором последовательности операций, определяемой входными взаимодействиями цифровой системы. При заданной схеме технических средств, то есть фиксированном интерпретаторе микрокоманд, команд или операторов программного обеспечения функции программно-микропрограммного обеспечения в целом совпадают с функциями цифровой системы [4-6].

Начальный выбор тестов для отладки программно-микропрограммного обеспечения методом моделирования аналогичен выбору отладоч-

ных тестов для цифровой системы в целом и должен обеспечить проверку всех выполняемых функций. Проверка каждой функции программно-микропрограммного обеспечения должна осуществляться на конечном наборе тестов, учитывающих структуру программ или микропрограмм.

Цель работы – разработка формальной модели программно-микропрограммного обеспечения и его фрагментов с учетом особенностей современных цифровых систем. Формальная модель позволит формализовать и автоматизировать составление отладочных тестов для каждой проверяемой функции программно-микропрограммного обеспечения. Основой предлагаемой модели является алгебраическое представление программного обеспечения, предложенное в [7].

При разработке модели необходимо учитывать особенности программно-микропрограммного обеспечения современных цифровых микроэлектронных систем, а именно:

- циклический характер работы программно-микропрограммного обеспечения, поскольку обычно цифровая система функционирует непрерывно, пока включено питание;
- выполнение определенных функций программно-микропрограммного обеспечения часто инициируется сигналами внешней среды;
- в современных цифровых системах широко используются различные типы прерываний.

Формализация понятия оператора. Рассмотрим множество элементов $\mathbf{B} = \{b_1, b_2, \dots\}$, каждый из которых может находиться в одном из состояний конечного множества $\mathbf{Z}_b^*$. Назовем эти элементы элементами памяти. Элементами памяти в нашем случае являются как собственно ячейки ЗУ, так и программно доступные регистры блоков. Множество состояний памяти есть $\mathbf{P} = \prod_{b \in \mathbf{B}} \mathbf{Z}_b^*$. Элементы p этого множества определяют состояния всех элементов памяти. Отображение $\varepsilon : \mathbf{P}_1 \to \mathbf{P}_2$, где $\mathbf{P}_1 \in \mathbf{P}$, $\mathbf{P}_2 \in \mathbf{P}$, есть оператор над памятью, $\mathbf{P}_1$ – область определения оператора, $\mathbf{P}_2$ – область значений.

Рассмотрим конечное множество операторов E. При выполнении программ и микропрограмм существенными являются не только изменения состояния элементов памяти (ячеек ЗУ и регистров), но также и переходы от одних команд (фрагментов) к другим. Введем в рассмотрение элемент $Сч$ с конечным множеством значений $\mathbf{N}$. Пусть задано отображение $\iota : \mathbf{N} \to \mathbf{E}$. Здесь $\mathbf{N}$ есть множество меток операторов, в частном случае подмножество натуральных чисел. Если элементы множества $\mathbf{E}$ есть отдельные команды, то в качестве элементов множества $\mathbf{N}$ могут рассматриваться адреса команд.

В общем случае фрагменты программы или микропрограммы, которые будут рассматриваться как операторы, могут иметь несколько входов. Обозначим конечное множество входов оператора ε через $\mathbf{A}_1^{\varepsilon}$. Отображение ι, ставящее в соответствие каждому входу всех операторов из $\mathbf{E}$ одно или несколько значений $Сч$, представим в виде:

$$\iota : \mathbf{N} \to \bigcup_{\varepsilon \in \mathbf{E}} \mathbf{A}_1^{\varepsilon}.$$

Назовем обобщенной памятью множество $\mathbf{B} \cup \{Сч\} \cup \mathbf{T}$, где $\mathbf{T}$ – множество моментов времени. Состоянием обобщенной памяти назовем $o = (p, n, t)$, где p – состояние памяти перед выполнением оператора; n – есть метка выполняемого оператора; t – время начала выполнения оператора. Различные элементы o образуют множество состояний обобщенной памяти $\mathbf{O}$, где $\mathbf{O} = \mathbf{P} \times \mathbf{N} \times \mathbf{T}$. Будем считать, что каждый оператор ε, кроме преобразования состояния памяти, осуществляет передачу управления на следующий оператор и требует для своего выполнения определенного времени. В соответствии с этим оператором ε будем называть отображение

$$\varepsilon : \mathbf{O}_1 \to \mathbf{O}_2;\ \mathbf{O}_1 \subset \mathbf{O};\ \mathbf{O}_2 \subset \mathbf{O},$$

где $\mathbf{O}_1$ – область определения оператора;

$\mathbf{O}_2$ – область значений оператора.

Сам оператор ε задается в виде команды, микрокоманды или фрагмента программы (макрокоманды).

Структура областей определения и значения операторов. Рассмотрим структуру множеств $\mathbf{O}_1$ и $\mathbf{O}_2$.

$$\mathbf{O}_1 = \bigcup_{a_1 \in \mathbf{A}_1} \mathbf{O}_1^{a_1};\ \mathbf{O}_1^{a_1} = \mathbf{P}_1^{a_1} \times \{a_1\} \times \mathbf{T},$$

где $\mathbf{A}_1$ – конечное множество состояний $Сч$, соответствующих входам в оператор ε (множество входных меток);

a_1 – элемент множества $\mathbf{A}_1$;

$\mathbf{O}_1^{a_1}$ – подобласть определения ε, соответствующая входу с меткой a_1;

$\mathbf{P}_1^{a_1}$ – подмножество допустимых состояний памяти перед выполнением оператора ε, соответствующее входу с меткой a_1;

$\mathbf{T}$ – множество моментов времени.

$$\mathbf{O}_2 = \bigcup_{a_2 \in \mathbf{A}_2} \mathbf{O}_2^{a_2};\ \mathbf{O}_2^{a_2} = \mathbf{P}_2^{a_2} \times \{a_2\} \times \mathbf{T};$$

$$\mathbf{A}_2 \cap \mathbf{A}_1 = \varnothing,$$

где $\mathbf{A}_2$ – конечное множество состояний $Сч$, соответствующих выходам оператора ε (множество выходных меток);

a_2 – элемент множества $\mathbf{A}_2$;

$\mathbf{O}_2^{a_2}$ – подобласть определения ε, соответствующая выходу с меткой a_2;

$\mathbf{P}_2^{a_2}$ – подмножество возможных состояний памяти после выполнения оператора ε, соответствующих выходу с меткой a_2;

$\mathbf{T}$ – множество моментов времени.

Оператор ε, заданный указанным образом, определяет новое состояние памяти:

$$p_2 = \varepsilon_p(p_1, a_1),\ p_1 \in \mathbf{P}_1^{a_1},\ a_1 \in \mathbf{A}_1,$$

где ε_p – отображение, задающее новое состояние памяти при выполнении оператора ε;

новое состояние $Сч$:

$$a_2 = \varepsilon_a(p_1, a_1),\ p_1 \in \mathbf{P}_1^{a_1},\ a_1 \in \mathbf{A}_1,$$

где ε_a – отображение, задающее метку выхода при выполнении оператора ε;

новое значение времени:

$$t_2 = t_1 + t(p_1, a_1),\ t_1 \in \mathbf{T},\ t_2 \in \mathbf{T},$$

где t_1 – значение времени перед выполнением оператора;

t_2 – значение времени после выполнения оператора;

$t(p_1, a_1)$ – время выполнения оператора.

Каждое выполнение оператора ε характеризуется парой (a_1, a_2), $a_1 \in \mathbf{A}_1$, $a_2 \in \mathbf{A}_2$. Структура множеств $\mathbf{O}_1$ и $\mathbf{O}_2$ может быть задана следующим образом:

$$\mathbf{O}_1 = \prod_{\substack{a_1 \in \mathbf{A}_1 \\ a_2 \in \mathbf{A}_2}} \mathbf{O}^{a_1 a_2};\ \mathbf{O}^{a_1 a_2} = \mathbf{P}_1^{a_1 a_2} \times \{a_1\} \times \mathbf{T};$$

$$\mathbf{P}_1^{a_1 a_2} \cap \mathbf{P}_1^{a_1 a_2'} = \varnothing \text{ при } a_2 \neq a_2';$$

$$\mathbf{O}_2 = \bigcup_{\substack{a_1 \in \mathbf{A}_1 \\ a_2 \in \mathbf{A}_2}} \mathbf{O}_2^{a_1 a_2};\ \mathbf{O}_2^{a_1 a_2} = \mathbf{P}_2^{a_1 a_2} \times \{a_2\} \times \mathbf{T},$$

где $\mathbf{O}_1^{a_1 a_2}$, $\mathbf{O}_2^{a_1 a_2}$ – множества состояний обобщенной памяти перед и после выполнения оператора ε при входе в него по метке a_1 и выходе по метке a_2;

$\mathbf{P}_1^{a_1 a_2}$, $\mathbf{P}_2^{a_1 a_2}$ – множества состояний памяти перед и после выполнения оператора ε при входе в него по метке a_1 и выходе по метке a_2.

В случае, если прохождение оператора ε по пути $(a_1 a_2)$ невозможно, $\mathbf{P}_1^{a_1 a_2} = \varnothing$, $\mathbf{P}_2^{a_1 a_2} = \varnothing$.

Оператор ε определяет конечное множество подоператоров $\varepsilon^{a_1 a_2}$, $a_1 \in \mathbf{A}_1$, $a_2 \in \mathbf{A}_2$, соответствующих входу в оператор по метке a_1 и выходу по метке a_2:

$$\varepsilon^{a_1 a_2} : \mathbf{P}_1^{a_1 a_2} \times \{a_1\} \times \mathbf{T} \to \mathbf{P}_2^{a_1 a_2} \times \{a_2\} \times \mathbf{T}.$$

Каждый подоператор $\varepsilon^{a_1 a_2}$ определяет новое состояние памяти:

$$p_2 = \varepsilon_p^{a_1 a_2}(p_1),\ p_1 \in \mathbf{P}_1^{a_1 a_2},\ p_2 \in \mathbf{P}_2^{a_1 a_2},$$

где $\varepsilon_p^{a_1 a_2}$ – отображение, задающее новое состояние памяти, новое состояние $Сч$, равное a_2, и новое значение времени:

$$t_2 = t_1 + t^{a_1 a_2}(p_1),\ t_1 \in \mathbf{T},\ t_2 \in \mathbf{T},\ p_1 \in \mathbf{P}_1^{a_1 a_2},$$

где $t^{a_1 a_2}(p_1)$ – время выполнения оператора ε при входе в него по метке a_1 и выходе по метке a_2.

Программно-микропрограммное обеспечение как полугруппа операторов. Определим произведение операторов $\varepsilon^{\mathbf{I}}$ и $\varepsilon^{\mathbf{II}}$, $\varepsilon^{\mathbf{I}} \in \mathbf{E}$, $\varepsilon^{\mathbf{II}} \in \mathbf{E}$ как оператор $\varepsilon^{\mathbf{III}} = \varepsilon^{\mathbf{I}} \varepsilon^{\mathbf{II}}$. Необходимым условием существования этого произведения является $\mathbf{A}_1^{\mathbf{I}} \cap \mathbf{A}_1^{\mathbf{II}} = \varnothing$, где $\mathbf{A}_1^{\mathbf{I}}$, $\mathbf{A}_1^{\mathbf{II}}$ – множества $\mathbf{A}_1$ операторов $\varepsilon^{\mathbf{I}}$ и $\varepsilon^{\mathbf{II}}$ соответственно.

Множества входных $\mathbf{A}_1^{\mathbf{III}}$ и выходных $\mathbf{A}_2^{\mathbf{III}}$ меток оператора $\varepsilon^{\mathbf{III}}$ есть $\mathbf{A}_1^{\mathbf{III}} = \mathbf{A}_1^{\mathbf{I}} \cup \mathbf{A}_1^{\mathbf{II}}$; $\mathbf{A}_2^{\mathbf{III}} = (\mathbf{A}_2^{\mathbf{I}} \cup \mathbf{A}_2^{\mathbf{II}}) / ((\mathbf{A}_1^{\mathbf{I}} \cap \mathbf{A}_2^{\mathbf{II}}) \cup (\mathbf{A}_1^{\mathbf{II}} \cap \mathbf{A}_2^{\mathbf{I}}))$.

На рисунке 2 показаны возможные четыре случая связей между операторами $\varepsilon^{\mathbf{I}}$ и $\varepsilon^{\mathbf{II}}$.

Соответственно, при выполнении операторов $\varepsilon^{\mathbf{I}}$, $\varepsilon^{\mathbf{II}}$ существует определенная последовательность их работы, зависящая от начального состояния памяти p и входа $a_1^{\mathbf{III}}$. Любое выполнение оператора $\varepsilon^{\mathbf{III}}$ характеризуется следом s, который есть слово в алфавите

$$\left\{ (a_1^{\mathbf{I}}, a_2^{\mathbf{I}}) \middle| a_1^{\mathbf{I}} \in \mathbf{A}_1^{\mathbf{I}}, a_2^{\mathbf{I}} \in \mathbf{A}_2^{\mathbf{I}} \right\} \cup \left\{ (a_1^{\mathbf{II}}, a_2^{\mathbf{II}}) \middle| a_1^{\mathbf{II}} \in \mathbf{A}_1^{\mathbf{II}}, a_2^{\mathbf{II}} \in \mathbf{A}_2^{\mathbf{II}} \right\}.$$

То есть

$$s = \left(a_{1j_1}^{i_1}, a_{2k_1}^{i_1}\right)\left(a_{1j_2}^{i_2}, a_{2k_2}^{i_2}\right)..\left(a_{1j_m}^{i_m}, a_{2k_m}^{i_m}\right), \qquad (1)$$

где $i_1, i_2, ..., i_m$ принимают значения из множества $\{\mathbf{I}, \mathbf{II}\}$;

$i_1 = i_3 = i_5 = ...;\ i_2 = i_4 = i_6 = ...;\ i_1 \neq i_2$;

$a_{1j_1}^{i_1}, a_{1j_3}^{i_3}, ...$ принадлежат множеству $\mathbf{A}_1^{i_1}$;

$a_{2k_2}^{i_1}, a_{2k_2}^{i_3}, ...$ принадлежат множеству $\mathbf{A}_2^{i_1}$;

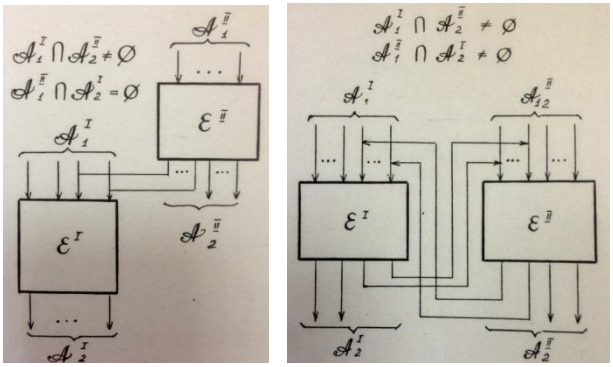
$a_{1j_2}^{i_2}, a_{1j_4}^{i_4}, ...$ принадлежат множеству $\mathbf{A}_1^{i_2}$;

$a_{2k_2}^{i_2}, a_{2k_4}^{i_4}, ...$ принадлежат множеству $\mathbf{A}_2^{i_2}$;

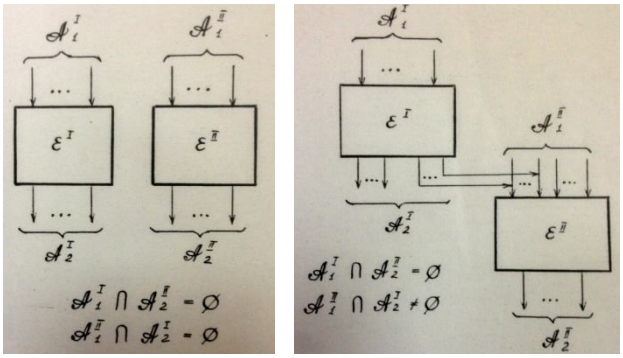
$a_{1j_1}^{i_1} = a_1^{\mathbf{III}};\ a_{2k_1}^{i_1} = a_{1j_2}^{i_2}$;

$a_{2k_2}^{i_2} = a_{1j_3}^{i_3}; \ldots; a_{2k_m}^{i_m} = a_2^{\mathbf{III}}$;

m – количество букв в следе s, то есть количество прохождений через операторы $\varepsilon^{\mathbf{I}}$, $\varepsilon^{\mathbf{II}}$.



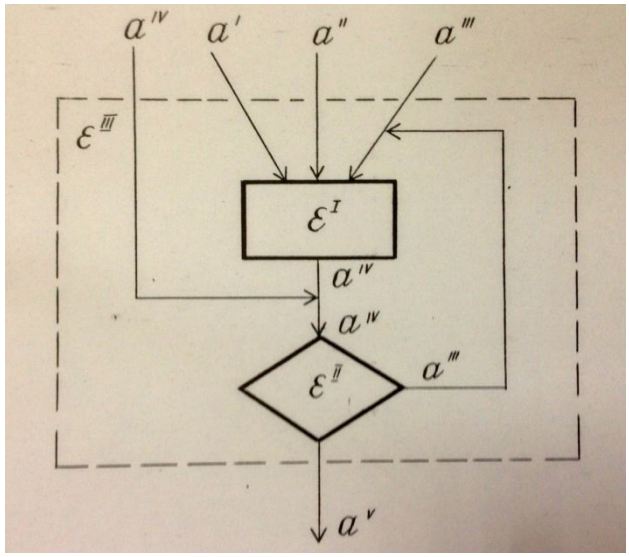
а) б)



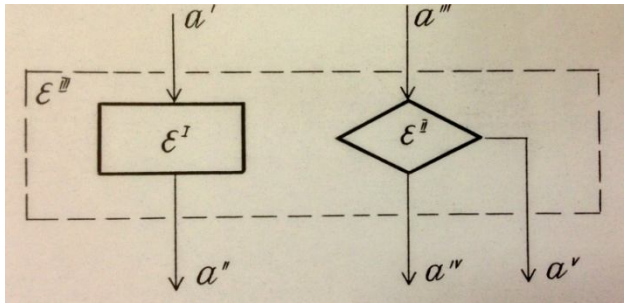
в) г)

Рисунок 2 – Различные случаи взаимодействия операторов $\varepsilon^{\mathbf{I}}$ и $\varepsilon^{\mathbf{II}}$

Каждой паре $\left(a_1^{\mathbf{III}}, a_2^{\mathbf{III}}\right)$ оператора $\varepsilon^{\mathbf{III}}$ соответствует множество следов вида (1). Необходимым условием того, что $p \in \mathbf{P}_1^{\mathbf{III}a_1a_2}$, где $\mathbf{P}_1^{\mathbf{III}a_1a_2}$ – множество состояний памяти, на котором определен оператор $\varepsilon^{\mathbf{III}a_1a_2}$, то есть оператор $\varepsilon^{\mathbf{III}}$ при входе в него по метке a_1 и выходе по метке a_2 является конечность следа s. Для каждой пары $\left(a_1, a_2\right)$, $a_1 \in \mathbf{A}_1^{\mathbf{III}}$, $a_2 \in \mathbf{A}_2^{\mathbf{III}}$ рассмотрим множество $\mathbf{S}^{a_1a_2}$ конечных следов. Каждый след $s \in \mathbf{S}^{a_1a_2}$ определяет выполнение последовательности операторов и задает преобразование состояния памяти, элемента *Сч* и времени.



а)



б)

Рисунок 3 – Пары операторов, произведения которых существуют

На рисунке 3 показаны пары операторов $\varepsilon^{\mathbf{I}}$ и $\varepsilon^{\mathbf{II}}$, произведения которых существуют. Для рисунка 3, а $\mathbf{A}_1^{\mathbf{I}} = \{a^{\mathbf{I}}, a^{\mathbf{II}}, a^{\mathbf{III}}\}$, $\mathbf{A}_2^{\mathbf{I}} = \{a^{\mathbf{IV}}\}$, $\mathbf{A}_1^{\mathbf{II}} = \{a^{\mathbf{IV}}\}$, $\mathbf{A}_2^{\mathbf{II}} = \{a^{\mathbf{III}}, a^{\mathbf{V}}\}$. Входными и выходными множествами меток оператора $\varepsilon^{\mathbf{III}}$ являются $\mathbf{A}_1^{\mathbf{III}} = \{a^{\mathbf{I}}, a^{\mathbf{II}}, a^{\mathbf{III}}, a^{\mathbf{IV}}\}$, $\mathbf{A}_2^{\mathbf{III}} = \{a^{\mathbf{V}}\}$.

Для рисунка 3, б $\mathbf{A}_1^{\mathbf{I}} = \{a^{\mathbf{I}}\}$, $\mathbf{A}_2^{\mathbf{I}} = \{a^{\mathbf{II}}\}$, $\mathbf{A}_1^{\mathbf{II}} = \{a^{\mathbf{III}}\}$, $\mathbf{A}_2^{\mathbf{II}} = \{a^{\mathbf{IV}}, a^{\mathbf{V}}\}$, $\mathbf{A}_1^{\mathbf{III}} = \{a^{\mathbf{I}}, a^{\mathbf{III}}\}$, $\mathbf{A}_2^{\mathbf{III}} = \{a^{\mathbf{II}}, a^{\mathbf{IV}}, a^{\mathbf{V}}\}$.

Таким образом, на множестве операторов $\mathbf{E}$ определена полугруппа, причем произведение любой последовательности операторов, если $\mathbf{A}_2^i \cap \mathbf{A}_1^j \neq \varnothing$ при $i \neq j$, для всех операторов существует и является оператором. Отметим, что при использовании введенной операции умноже-

ния результат произведения набора операторов не зависит от последовательности умножения.

В качестве одного из операторов можно рассматривать команду останова. Эта команда в программах реального времени используется для останова в режиме ожидания запроса на прерывание. В этом случае команду останова можно рассматривать как оператор, выполняющийся за один такт, при отсутствии запроса на прерывание передающий управление на свою входную метку и, как и все остальные команды при немаскированном прерывании, в качестве одного из аргументов проверяющий состояние регистра запроса на прерывание. Отметим, что для вычислительных программ, в которых прерывания могут не использоваться, команда останова может рассматриваться как оператор с $\mathbf{A}_2 = \varnothing$.

Программой является оператор, у которого $\mathbf{A}_2 = \varnothing$. Программа может содержать или не содержать оператор останова.

После того, как программа или микропрограмма сформирована в виде произведения некоторого множества операторов, выделим в множестве входных меток некоторое множество внешних входов программы или микропрограммы. Хотя в общем случае возможно наличие нескольких входов в программу или микропрограмму, в программе или микропрограмме цифровой системы в целом, как правило, имеется одна точка, с которой начинается ее выполнение при включении питания.

Учет действия прерываний осуществляется выделением в множестве $\mathbf{B}$ регистров запроса на прерывание и регистров векторов прерывания.

Заключение. Если отладка программного или микропрограммного обеспечения осуществляется с использованием моделирования технических средств, она может быть выполнена на машинных моделях как функционально-логического уровня, так и уровня архитектуры или микроархитектуры.

Составление набора тестов в этом случае должно осуществляться, прежде всего, исходя из выполняемых функций, а затем – исходя из структуры программы или микропрограммы [8-10]. В последнем случае могут быть использованы различные методы: метод покрытия путей, линейных участков или переходов, тестирование по особым значениям, иерархическое тестирование и др. [11-13]. Использование предложенной модели позволит учесть особенности программно-микропрограммного обеспечения современных цифровых систем и разработать систему его автоматизированного тестирования с высокой эффективностью. При этом возможна реализация автоматизированной генерации тестовых входных взаимодействий как на основе структуры программно-микропрограммного обеспечения, так и на основе анализа заданных в спецификации алфавита выполняемых функций и их возможных последовательностей.

Библиографический список

1. *Иванников А.Д., Стемпковский А.Л.* Формализация задачи отладки проектов цифровых систем // Информационные технологии. 2014. № 9. С. 3-10.
2. *Иванников А.Д., Стемпковский А.Л.* Математическая модель отладки проектов сложных цифровых систем и микросистем на основе представления последних в виде семейства стационарных динамических систем // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2014. Часть II. С. 123-128.
3. *Иванников А.Д.* Методы декомпозиции задачи отладки проекта цифровых систем с помощью моделироваеия // Вестник Рязанского государственного радиотехнического университета. 2014. № 49. С. 73–76.
4. *Иванников В.П., Камкин А.С., Косачев А.С., Кулямин В.В., Петренко А.К.* Использование контрактных спецификаций для представления требований и функционального тестирования моделей аппаратуры // Программирование. 2007. Т. 33. № 5. С. 47-62.
5. *Nguen M.D.* Hardware/software formal co-verification using hardware verification techniques // Fourth Int. Conf. on Communications and Electronics (ICCE). 2012. Pp. 465-470.
6. *Вишнеков А.В., Ерохин В.В.* Верификация СНК: выбор стратегии // Нано- и микросистемная техника. 2014. № 12. С. 30-36.
7. *Ляпунов А.А.* К алгебраической трактовке программирования // В кн.: Проблемы кибернетики. – М.: Наука, 1962. Вып. 2. С. 235-241.
8. *Lee J., Kang S., Lee D.* Survey on Software Testing Practices // Software, IET. 2012. Vol. 6. № 3. Pp. 275-282.
9. *Voas J., Miller K.W.* Software Testing: What Goes Around Comes Around // IT Professional. 2012. Vol. 14. № 3. Pp. 4-5.
10. *Bryce R.C., Sampath S., Memon A.M.* Developing a Single Model and Test Prioritization Strategies for Event-Driven Software // IEEE Transactions on Software Engineering. 2011. Vol. 37. № 1. Pp. 48-64.
11. *Кулямин В.В., Петухов А.А.* Обзор методов построения покрывающих наборов // Программирование. 2011. Т. 37. № 3. С. 3-41.
12. *Moonzoo Kim, Yunho Kim, Hotae Kim.* A Comparative Study of Software Model Checkers as Unit Testing Tools: An Industrial Case Study // IEEE Transactions on Software Engineering. 2011. Vol. 37. № 2. Pp. 146-160.
13. *Arcaini P., Gargantini A., Vavassori P.* Validation of Models and Tests for Constrained Combinational Interaction Testing // IEEE Seventh Int. Conf. on Software Testing, Verification and Validation Workshops (ICSTW) 2014. Pp. 98-107.