СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ
CAD-SYSTEMS

УДК 004.31

А. Д. Иванников, д-р техн. наук, e-mail: 9287743@gmail.com; зам. директора по научной работе, Институт проблем проектирования в микроэлектронике Российской академии наук

Анализ методов декомпозиции задачи отладки проектов цифровых систем

В связи со сложностью и большой размерностью задачи отладки сложных цифровых систем методом моделирования в целом предлагаются алгебраические модели методов декомпозиции этой задачи, а именно: вертикальной и горизонтальной структурной декомпозиции, функциональной декомпозиции, декомпозиции по типам ошибок. Приводится алгебраическая модель программного обеспечения цифровых систем. Программное обеспечение представляется как полугруппа операторов.

Ключевые слова: *модель сложных цифровых систем, отладка методом моделирования на ЭВМ, проектирование систем на кристалле, логическое моделирование, логико-временной анализ, алгебраические модели, модель программного обеспечения, программа как полугруппа операторов*

Введение

В настоящее время проектирование сложных цифровых схем и микросистем на кристалле возможно только с использованием средств автоматизации проектирования. Причем обычно сжатые сроки проектирования делают необходимым широкое использование уже спроектированных ранее блоков.

Спроектированная сложная цифровая схема или микросистема (далее — цифровая система) задается в виде принципиальной электрической схемы Cx технических средств и текста программного обеспечения $П$, т. е. в виде пары (Cx, $П$).

Основой отладки цифровых систем является отладочный эксперимент, в котором на входы объекта отладки (компьютерной модели цифровой системы) подается отладочный тест, определяется выходное воздействие и, возможно, какие-то внутренние переменные, в зависимости от времени. Если выходное воздействие не соответствует требуемому, осуществляются локализация ошибки и ее исправление.

Сравнительно малое число выводов и высокая сложность технических средств и программного обеспечения реальных цифровых систем не дают возможности локализовать ошибку исходя лишь из наблюдения выходных воздействий цифровой системы. В целях получения более полной информации о работе отлаживаемой системы разработчик анализирует изменения содержимого ячеек запоминающего устройства (ЗУ) и регистров блоков, а также логических сигналов на выходах блоков, из которых состоит цифровая система, если модель цифровой системы, используемая в качестве объекта отладки, это позволяет. Однако и при этом условии высокая сложность современных цифровых схем, большое число возможных источников ошибок затрудняют локализацию последних в связи с высокой размерностью задачи. Эффективным средством в этом случае является декомпозиция задачи отладки на ряд задач отладки меньшей размерности. Целью данной работы являются алгебраическая формулировка принципов декомпозиции задачи отладки современных цифровых схем методом моделирования, а также разработка алгебраической модели программного обеспечения цифровых систем.

Возможные методы декомпозиции задачи отладки методом моделирования

Структурная вертикальная декомпозиция. Цифровая система может быть представлена как вертикальная иерархическая композиция из нескольких компонентов различных уровней. Таких уровней, как минимум, два:

- технические средства, определяющие архитектуру или микроархитектуру цифровой системы и являющиеся интерпретатором последовательности команд или микрокоманд;
- программное или микропрограммное обеспечение, которое в комплексе с техническими средствами определяет переработку (является интерпретатором) последовательностей входных сигналов в выходные.

Цифровая система в ряде случаев содержит большее число уровней. Достаточно часто циф-

ровая система строится из отлаженных ранее блоков, которые в свою очередь представляют собой схемы соединения блоков более низкого уровня. Также в ряде цифровых систем можно выделить уровень технических средств, определяющих микроархитектуру системы (систему микрокоманд), уровень микропрограмм, определяющих архитектуру системы (систему команд, методы адресации, дисциплину обслуживания прерываний) и уровень программного обеспечения. Программное обеспечение может быть в ряде случаев представлено как имеющее два и более уровней.

Цифровая система в процессе работы выполняет некоторую последовательность функций из конечного алфавита **K,** свойственного конкретной системе. Каждую такую последовательность $f \in \mathbf{F}$ будем считать программной, выполняемой (интерпретируемой) цифровой системой. "Командами" таких программ являются отрезки входных взаимодействий, вызывающих выполнение той или иной операции из алфавита **K**. Входными взаимодействиями мы называем временные диаграммы входных сигналов и выходных сигналов управления обменом, т. е. выходных сигналов квитирования и инициации ввода информации [1].

Цифровую систему в целом необходимо рассматривать совместно с множеством **F** выполняемых последовательностей функций.

Цифровая система может быть представлена как

$$(\mathbf{L}_1, \mathbf{L}_2, ..., \mathbf{L}_n, \mathbf{F}_n), \qquad (1)$$

где $\mathbf{L}_1$ — схема технических средств; $\mathbf{L}_2$, ..., $\mathbf{L}_n$ — программы или микропрограммы i-го уровня, представляющие собой определенные последовательности команд этого уровня; n — число уровней, имеющихся в системе; $\mathbf{F}_n$ — множество программ, выполняемых цифровой системой в целом.

Кортеж ($\mathbf{L}_1$, ..., $\mathbf{L}_i$) задает правила интерпретации команд i-го уровня, т. е. виртуальную машину этого уровня с системой команд $\mathbf{K}_i$.

Отладка цифровой системы в целом представляет собой выбор из множества $\mathbf{F}_n$ конечного множества конечных по времени отладочных тестов $\mathrm{TST}_n \subset \mathbf{F}_n$, выполнения TST_n с помощью интерпретатора ($\mathbf{L}_1$, $\mathbf{L}_2$, ..., $\mathbf{L}_n$), определение правильности или ошибочности интерпретации, в случае наличия ошибки — ее идентификацию и исправление.

В соответствии с принципом структурной вертикальной декомпозиции задачу отладки цифровой системы ($\mathbf{L}_1$, $\mathbf{L}_2$, ..., $\mathbf{L}_n$, $\mathbf{F}_n$) можно заменить отладкой виртуальной машины первого уровня, т. е. отладкой интерпретатора ($\mathbf{L}_1$) на наборе отладочных тестов TST_1; отладкой виртуальной машины второго уровня, т. е. отладкой интерпретатора ($\mathbf{L}_1$, $\mathbf{L}_2$) на наборе отладочных тестов TST_2 и т. д., заканчивая отладкой машины n-го уровня (цифровой системы в целом), т. е. отладкой интерпретатора ($\mathbf{L}_1$, $\mathbf{L}_2$, ..., $\mathbf{L}_n$) на наборе отладочных тестов $\mathrm{TST}_n \subset \mathbf{F}_n$. Такая декомпозиция существенно снижает размерность задачи отладки в связи с меньшим числом возможных ошибок и их типов в каждом уровне $\mathbf{L}_i$ по сравнению с цифровой системой в целом.

При этом для каждого уровня необходимо:

- иметь возможность выделить множество допустимых программ $\mathbf{F}_i$ для виртуальной машины i-го уровня;
- построить набор отладочных тестов $\mathrm{TST}_i \subset \mathbf{F}_i$;
- знать множество допустимых реакций для всех возможных программ i-го уровня или, по крайней мере, для всех отладочных тестов TST_i;
- иметь машинную модель интерпретатора i-го уровня (модель технических средств, модели цифровой системы на уровнях микроархитектуры, архитектуры, базовых макрокоманд, из которых формируется общий алгоритм работы системы) с возможностью слежения за внутренними переменными модели этого уровня.

В качестве такой модели всегда может быть использован интерпретатор первого уровня ($\mathbf{L}_1$), т. е. модель технических средств, в сочетании с программами ($\mathbf{L}_2$, ..., $\mathbf{L}_i$). Однако такие модели требуют больших затрат машинного времени в связи с моделированием внутренних переменных всех уровней. В этих случаях более экономично в смысле затрат времени инструментальной ЭВМ использовать специально созданные и отлаженные эмуляторы i-го уровня, моделирующие внутренние переменные только этого уровня. Например, для отладки программного обеспечения могут быть использованы функционально-логические модели системы на уровне соединения типовых блоков. Но во многих случаях для отладки программ информация о логических сигналах на выводах блоков не является необходимой. Тогда целесообразно использование эмуляторов цифровых систем на уровне архитектуры (системы команд), позволяющих следить только за изменениями содержимого программно-доступных регистров блоков и ячеек ЗУ.

Структурная горизонтальная декомпозиция. Во многих случаях определенный уровень $\mathbf{L}_i$ цифровой системы можно представить как некоторое множество связанных между собой блоков. Возможны случаи, когда такие блоки функционируют одновременно (процессорные блоки в мультипроцессорной системе; блоки каналов ввода-вывода и другие специализированные периферийные блоки, функционирующие параллельно с выполнением основной программы) или в определенной очередности (блоки ЗУ, ввода-вывода, подключенные к общей шине; процессорный блок и блок контроллера прямого доступа к памяти).

На микропрограммном уровне могут быть выделены микропрограммы, реализующие отдельные команды, микропрограмма обслуживания пульта, микропрограмма, реализующая дисциплину обслу-

живания прерываний. В программном обеспечении цифровых систем реального времени могут быть выделены фрагменты, запускаемые при поступлении сигнала прерывания с заданным приоритетом.

Разбиение уровня $\mathbf{L}_i$ на блоки обычно осуществляется таким образом, чтобы их интерфейс был наиболее простым и легко описываемым.

Итак, каждый иерархический уровень $\mathbf{L}_i$ цифровой системы может быть представлен как

$$\mathbf{L}_i = \begin{pmatrix} u_i^1 \\ u_i^2 \\ \dots \\ u_i^j \end{pmatrix},$$

где u_i^j — блок, выделенный в уровне $\mathbf{L}_i$; j — число блоков, на которые разбит уровень $\mathbf{L}_i$.

В соответствии с принципом структурной горизонтальной декомпозиции отладка $\mathbf{L}_i$ может быть заменена автономной отладкой блоков u_i^1, u_i^2, ..., u_i^j и отладкой их взаимодействия. Так, в случае уровня технических средств возможна отдельная отладка блоков ОЗУ, ПЗУ, блока микропрограммного управления и отладка обмена информацией между блоками по общей шине. В случае микропрограммного обеспечения — отдельная отладка микропрограмм, реализующих каждую команду.

Функциональная декомпозиция. Цифровая система обычно реализует конечный набор функций, например, контроль температуры объекта, индикацию состояния объекта на табло, управление шаговыми двигателями и т. д. Виртуальные машины более низких уровней ($\mathbf{L}_1$, $\mathbf{L}_2$, ...) или их блоки также реализуют конечный набор функций или команд. Так, блок микропрограммного управления может реализовать переход к микропрограмме с адресом, который на единицу больше текущего адреса, условный или безусловный переход к указанному адресу микрокоманды, переход к микроподпрограмме или возврат из нее. Какой-либо контроллер внешнего устройства, подключенный к общей шине системы, может осуществлять только три операции (цикла): чтение, запись, чтение — пауза — запись, что определяется стандартом на шину.

В соответствии с принципом функциональной декомпозиции при отладке цифровой системы или ее составляющих необходимо проводить последовательную отладку функций, выполняемых системой или ее частью, а также проверку правильности выполнения последовательности функций.

На принципе функциональной декомпозиции, в частности, основаны многие современные методы выбора отладочных тестов для цифровых систем [2]. Следует отметить, что результат применения функциональной декомпозиции может совпадать с результатом структурной горизонтальной декомпозиции, например, в случае отладки микропрограмм, реализующих отдельные команды процессорного блока.

Декомпозиция по типам ошибок. При отладке цифровых систем методом моделирования могут быть выделены группы отладочных экспериментов для выявления ошибок определенных типов. Так, при проведении отладки могут использоваться машинные модели на уровне архитектуры системы и на уровне сети составляющих ее блоков, что в свою очередь подразделяется на модели для логической отладки и модели для верификации временных диаграмм. Более того, верификация временных диаграмм может проводиться с различной степенью подробности [3, 4]. Модели каждой степени детализации ориентированы на выявление определенных типов ошибок. Разделение задачи проверки технических средств цифровой системы на верификацию временных диаграмм и логическую отладку является результатом применения принципа декомпозиции задачи отладки по типам ошибок.

Принцип декомпозиции по типам ошибок может быть использован совместно с другими принципами декомпозиции. Примером такого подхода может служить моделирование на определенном уровне для выявления и устранения ошибок проектирования какого-либо типа, но проводимого не для всего проекта сразу, а для его отдельных структурных частей.

Алгебраическая модель программного обеспечения цифровых систем

Из представления цифровой системы в виде (1) видно, что программное обеспечение может иметь несколько уровней. Для формализованной обработки разработанного программного обеспечения необходимо иметь его формальную алгебраическую модель. При разработке этой модели были использованы идеи, изложенные в работе [5].

Формализация понятия оператора. Рассмотрим множество элементов $\mathbf{B} = \{b_1, b_2, ...\}$, 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 ε: $\mathbf{P}_1 \to \mathbf{P}_2$, 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 $\mathbf{E}$. 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 δ: $\mathbf{N} \to \mathbf{E}$. Здесь $\mathbf{N}$ есть множество меток

операторов, в частном случае — подмножество натуральных чисел. Если элементы множества **E** есть отдельные команды, то в качестве элементов множества **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 δ, ставящее в соответствие каждому входу всех операторов из **E** одно или несколько значений *Сч*, представим в виде

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

Назовем обобщенной памятью множество $\mathbf{B} \cup \{Сч\} \cup \mathbf{T}$, где **T** — множество моментов времени. Состоянием обобщенной памяти назовем $o = (p, n, t)$, где p — состояние памяти перед выполнением оператора; n — метка выполняемого оператора; t — время начала выполнения оператора. Различные элементы o образуют множество состояний обобщенной памяти **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; **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; **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:

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

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

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

где $\mathbf{O}_1^{a_1 a_2}$, $\mathbf{O}_2^{a_1 a_2}$ — множества состояний обобщенной памяти до 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 до 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 $\varepsilon^{a_1a_2}$ определяет новое состояние памяти:

$$p_2 = \varepsilon_p^{a_1a_2}(p_1),\ p_1 \in \mathbf{P}_1^{a_1a_2},\ p_2 \in \mathbf{P}_2^{a_1a_2},$$

где $\varepsilon_p^{a_1a_2}$ — отображение, задающее новое состояние памяти, новое состояние *Сч*, равное a_2, и новое значение времени:

$$t_2 = t_1 + t^{a_1a_2}(p_1),\ t_1 \in \mathbf{T},\ t_2 \in \mathbf{T},\ p_1 \in \mathbf{P}_1^{a_1a_2},$$

где $t^{a_1a_2}(p_1)$ — время выполнения оператора ε при входе в него по метке a_1 и выходе по метке a_2.

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

Множества входных $\mathbf{A}_1^{\mathrm{III}}$ и выходных $\mathbf{A}_2^{\mathrm{III}}$ меток оператора $\varepsilon^{\mathrm{III}}$ есть

$$\mathbf{A}_1^{\mathrm{III}} = \mathbf{A}_1^{\mathrm{I}} \cup \mathbf{A}_1^{\mathrm{II}};$$

$$\mathbf{A}_2^{\mathrm{III}} = (\mathbf{A}_2^{\mathrm{I}} \cup \mathbf{A}_2^{\mathrm{II}})/((\mathbf{A}_1^{\mathrm{I}} \cap \mathbf{A}_2^{\mathrm{II}}) \cup (\mathbf{A}_1^{\mathrm{II}} \cap \mathbf{A}_2^{\mathrm{I}})).$$

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

$$\{(a_1^{\mathrm{I}}, a_2^{\mathrm{I}}) | a_1^{\mathrm{I}} \in \mathbf{A}_1^{\mathrm{I}},\ a_2^{\mathrm{I}} \in \mathbf{A}_2^{\mathrm{I}}\} \cup$$

$$\cup \{(a_1^{\mathrm{I}}, a_2^{\mathrm{II}}) | a_1^{\mathrm{II}} \in \mathbf{A}_1^{\mathrm{II}},\ a_2^{\mathrm{II}} \in \mathbf{A}_2^{\mathrm{II}}\},$$

т. е.

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

где $i_1, i_2, \ldots, i_m$ принимают значения из множества $\{\mathbf{I}, \mathbf{II}\}$; $i_1 = i_3 = i_5 = \ldots$; $i_2 = i_4 = i_6 = \ldots$; $i_1 \neq i_2$;

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

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

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

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

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

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

m — число букв в следе s, т. е. число прохождений через операторы ε^{I}, $\varepsilon^{\mathrm{II}}$.

Каждой паре $(a_1^{\mathrm{III}}, a_2^{\mathrm{III}})$ оператора $\varepsilon^{\mathrm{III}}$ соответствует множество следов вида (2). Необходимым условием того, что $p \in \mathbf{P}_1^{\mathrm{III}a_1a_2}$, является конечность следа s. Здесь $\mathbf{P}_1^{\mathrm{III}a_1a_2}$ — множество состояний памяти, на котором определен оператор $\varepsilon^{\mathrm{III}a_1a_2}$, т. е. оператор $\varepsilon^{\mathrm{III}}$ при входе в него по метке a_1 и выходе по метке a_2. Для каждой пары (a_1, a_2), $a_1 \in \mathbf{A}_1^{\mathrm{III}}$, $a_2 \in \mathbf{A}_2^{\mathrm{III}}$ 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.

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 умножения 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 выполняющийся за один такт оператор, который при отсутствии запроса на прерывание передает управление на свою входную метку и, как и все остальные команды при немаскированном прерывании, в качестве одного из аргументов проверяет 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 с $\mathbf{A}_2 = \varnothing$. 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 **B** регистров запроса на прерывание и регистров векторов прерывания.

Заключение

Рассмотренные методы декомпозиции отладки проектов цифровых систем на этапе проектирования могут модифицироваться и, как указывалось выше, использоваться в смешанном варианте. Примером этого является смешанное моделирование проектов цифровых систем, когда одни части проекта моделируются на более высоком уровне, скажем, на уровне просто логических сигналов, а другие, наиболее критические, — на более детальном уровне, например, с подробным анализом фронтов сигналов.

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.

Использование предложенных моделей 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.

Список литературы

1. **Иванников А. Д., Стемпковский А. Л.** Формализация задачи отладки проектов цифровых систем // Информационные технологии. 2014. № 9. С. 3—10.

2. **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.

3. **Гаврилов С. В., Гудкова О. Н., Щелоков А. Н.** Логико-временной анализ нанометровых схем на основе интервального подхода // Известия ЮФУ. Технические науки. 2012. № 7 (132). С. 85—91.

4. **Стемпковский А. Л., Гаврилов С. В., Глебов А. Л.** Методы логического и логико-временного анализа цифровых КМОП СБИС. М.: Наука. 2007. 220 с.

5. **Ляпунов А. А.** К алгебраической трактовке программирования // В кн.: Проблемы кибернетики. М.: Наука. 1962. Вып. 2. С. 235—241.

A. D. Ivannikov, Doctor of Sciences, Deputy Director, e-mail: ADI@ippm.ru,
Institute for Design Problems in Microelectronics of Russian Academy of Sciences

Decomposition Methods Analysis for Digital System Design Debugging

The task of digital systems project debugging is usually fulfilled by simulation. Due to the complexity and large dimensions of the task decomposition methods are proposed. The following methods are described and examined: vertical and horizontal structure decomposition, functional decomposition, decomposition based on error types.

The concept of an operator upon digital system memory is formally introduced on the algebraic approach bases. The concept of a generalized memory consisting of memory cells, register contents, active operator label and current time value is introduced. An operator functioning upon generalized memory is formalized. Structural analysis of an operator domain and operator range is fulfilled. The definition of operator's product is introduced, the condition of such a product existence is analyzed. It is shown that any digital system software is the product of operators, operator set is a semi group. The developed formal model is supposed to be used for the creation of digital systems software test generation.

Keywords: *complex digital system model, debugging by computer simulation, system on a chip design, logical simulation, logic timing analysis, algebraic models, algebraic model for software, program as a semi group of operators*

References

1. **Ivannikov A. D., Stempkovsky A. L.** Formalizaciya zadachi otladki proektov cifrovih sistem (Formal mathematical representation for the task of digital system projects debugging), *Informacionnie Technologii*, 2014, no. 9, pp. 3—10 (in Russian).

2. **Ivannikov V. P., Kamkin A. S., Kossachev A. S., Kuliamin V. V., Petrenko A. K.** Ispolzovanie kontraktnih specifikaciy dlya predstavleniya trbovaniy i funkcionalnogo testirovaniya apparaturi (The use of contract specifications for representing requirements and for functional testing of hardware models), *Programmirovanie*, 2007, vol. 33, no. 5, pp. 47—62 (in Russian).

3. **Gavrilov S. V., Gudkova O. N., Schelokov A. N.** Logiko-vremennoi analis nanometrovih shem na osnove intervalnogo podhoda (Logika-timing analysis of nanometer circuits on interval approach base, *Izvestia UFU, Technicheskie nauki*, 2012, no. 7 (132), pp. 85—91 (in Russian).

4. **Stempkovsky A. L., Gavriliv S. V., Glebov A. L.** *Metodi logicheskogo i logiko-vremennogo analiza cifrovih KMOS SBIS* (Methods of logical and logico-timing analysis for digital CMOS LSI). Moscow, Nauka, 2007. 220 pp. (in Russian).

5. **Liapunov A. A.** *K algebraicheskoy traktovke programmirovaniya* (For the algebraic treatment of programming). In the book: Problemi kibernetiki, Moscow, Nauka, 1962, Iss. 2, pp. 235—241 (in Russian).