УДК 004.3

ПРИМЕНЕНИЕ ТЕОРИИ КОНЕЧНЫХ АВТОМАТОВ В ПРОЕКТИРОВАНИИ СПЕЦИАЛИЗИРОВАННЫХ АППАРАТНО-ПРОГРАММНЫХ СРЕДСТВ ДЛЯ АВТОМАТИЗАЦИИ ПРОЦЕССА ЗАГРУЗКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И ТЕСТИРОВАНИЯ ВСТРОЕННЫХ СИСТЕМ

APPLICATION OF FINITE AUTOMATA THEORY FINITE AUTOMATA THEORY IN DESIGNING A SPECIALIZED HARDWARE AND SOFTWARE TOOLS TO AUTOMATE THE PROCESS OF SOFTWARE LOADING AND TESTING EMBEDDED SYSTEMS

Р.В. Закшевский, Национальный исследовательский университет «МИЭТ»,
г. Москва, г. Зеленоград

Аннотация. В данной работе рассматривается применение теории конечных автоматов при проектировании специализированных аппаратно-программных средств для автоматизации процесса загрузки программного обеспечения и тестирования встроенных систем. Анализируются принципы формального описания состояний и переходов между ними. Описаны ключевые функциональные модули проектируемых аппаратно-программных средств.

Ключевые слова: встроенные системы, автоматизация, теория конечных автоматов, контрольно-проверочная аппаратура (КПА), производство встроенных систем, аппаратно-программные средства.

Abstract. This paper considers the application of finite automata theory in the design of a specialized hardware-software tools for automating the process of software loading and testing embedded systems. The principles of formal description of states and transitions between them are analyzed. The key functional modules of the designed hardware and software are described.

Keywords: embedded systems, automation, finite automata theory, control and test equipment (CTA), production of embedded systems, hardware and software tools.

При проектировании специализированных аппаратно-программных средств (АПС) для автоматизации процесса загрузки программного обеспечения и тестирования встроенных систем, с целью повышения скорости производства, снижения влияния человеческого фактора и повышения точности выявления дефектов, одним из эффективных методов формального описания и управления данными процессами является

применение теории конечных автоматов. Данный подход позволяет строго определить состояния системы, возможные переходы между ними и условия их изменения, что обеспечивает предсказуемость и корректность работы разработанного средства.

Проектируемые аппаратно-программные средства представляют собой настольное приложение, предназначенное для автоматизированной загрузки программного обеспечения в встроенные системы на базе микроконтроллеров STM32, последующего тестирования, а также калибровки определённого датчика.

На блок-схеме (рис. 1) представлена последовательность этапов работы системы, включающая загрузку программного обеспечения, тестирование устройства и процесс калибровки pH-датчика. Предложенная архитектура объединяет автоматизированную загрузку прошивки, диагностику работоспособности устройства и его настройку.

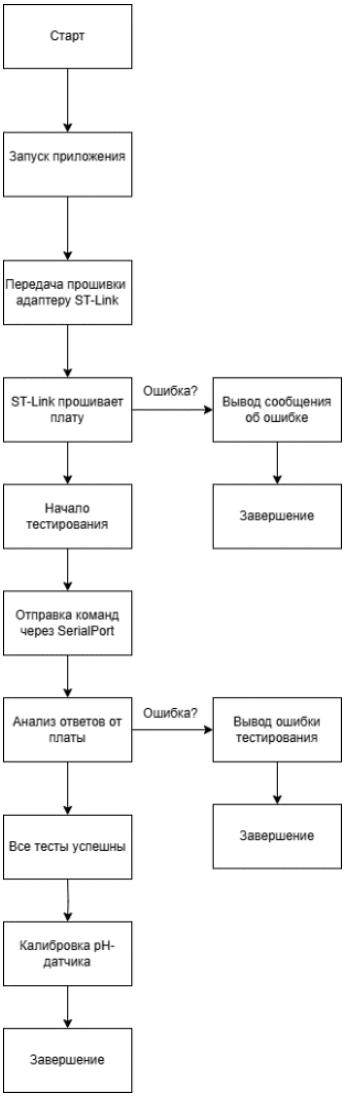

Рис. 1. Блок-схема основных этапов работы специализированных АПС.

В основе системы лежит несколько ключевых модулей, каждый из которых выполняет определённую задачу. Графический интерфейс обеспечивает удобное управление процессами прошивки, тестирования и калибровки, визуализирует ход выполнения операций и предоставляет пользователю результаты работы. Модуль взаимодействия с отладочным адаптером ST-Link отвечает за установление соединения с микроконтроллером, передачу прошивки и её верификацию. Модуль SerialPort управляет обменом данными с тестируемым устройством, отправляя команды и анализируя полученные ответы. Отдельный компонент занимается обработкой данных и генерацией отчётов о результатах работы системы.

Функциональность системы включает три ключевых этапа: загрузку программного обеспечения в микроконтроллер, тестирование устройства и калибровку pH-датчика.

В процессе разработки специализированных аппаратно-программных средств для автоматизированной прошивки и тестирования встроенных систем важной задачей является обеспечение надежного и предсказуемого управления процессами взаимодействия компонентов системы. Для этого применяется концепция конечного автомата (Finite State Machine, FSM), которая позволяет формально описать все возможные состояния системы и правила переходов между ними.

Конечные автоматы используются для вычисления переходов между состояниями, которые связаны с входными символами. Переходы в конечных автоматах зависят от текущего состояния и входных символов [1]. Использование конечных автоматов в разработке программного обеспечения для встроенных систем оправдано тем, что процессы взаимодействия с микроконтроллером, адаптером ST-Link и тестируемым устройством имеют четко определенные этапы и логические условия перехода между ними.

Конечный автомат можно описать как кортеж [2]:

$$M = (S, \sum, \delta, S_0, F) \quad (1)$$

где S – конечное множество состояний системы;
$\sum$ – конечное множество входных символов (событий, команд ил внешних воздействий);
$\delta: S \times \sum \to S$ – функция переходов, определяющая, в какое состояние перейдёт система при заданном входе;
$S_0 \in S$ – начальное состояние, в котором система находится при запуске;
$F \subseteq S$ – множество конечных состояний, в которых система может завершить выполнение.

Во входной алфавит $\sum$ включены события, влияющие на поведение системы:

$$\sum = \{CS, DB, TO, TF, TeO, TeF, CaS, CaF, CaSu\} \quad (2)$$

где CS – connect_stlink;
DB– detect_board;
TO – transfer_ok;
TF – transfer_fail;
TeO – test_ok;
$TeF -$ test_fail;
$CaS - calibration\_start$;
$CaF -$ calibration_fail;

$CaSu - calibration\_success$.

Множество состояний системы можно разделить на несколько групп:

— Состояния загрузки ПО:

$$S_{prog} = \{waiting, connecting, setup, transfer, verification, reboot, error\} \quad (3)$$

— Состояния тестирования устройства:

$$S_{test} = \{TW, SC, MT, AR, GR, TE\} \quad (4)$$

где TW – test_waiting;
SC – setup_connection;
MT – module_test;
AR – analyze_results;
GR – generate_report;
TE – test_error.

— Состояния калибровки pH-датчика:

$$S_{cal} = \{CaS, RS1, RS2, SD, AC, CD, CE\} \quad (5)$$

где $CaS - calibration\_start$;
$RS1$ – $read$_solution1;
$RS1$ – $read$_solution2;
$SD - send\_data$;
$AC - apply\_calibration$;
$CD - calibration\_done$;
$CE - calibration\_error$.

Функция переходов δ задаёт, как система реагирует на поступающие события:

$$\delta(\text{waiting}, \text{connect\_stlink}) = \text{connecting} \quad (6)$$

$$\delta(\text{verification}, \text{transfer\_ok}) = reboot \quad (7)$$

$$\delta(\text{module\_test}, \text{test\_ok}) = \text{analyze\_results} \quad (8)$$

$$\delta(\text{calibration\_start}, \text{read\_solution1}) = \text{read\_solution2} \quad (9)$$

Если во время калибровки произошла ошибка, система возвращается в состояние ошибки:

$$\delta(read\_solution1, calibration\_fail) = calibartion\_error \quad (10)$$

Граф конечного автомата можно представить в виде таблицы (матрицы переходов), где строки соответствуют текущим состояниям, а столбцы – входным событиям (табл.1).

Таблица 1. Матрица переходов.

Состояние / Событие	**Подключен ST-Link**	**Плата найдена**	**Перевод в режим программирования**	**Файл передан**	**Ошибка передачи**	**Плата подключена (SerialPort)**	**Данные собраны**
Ожидание	Установление соединения						
Установление соединения		Подготовка к загрузке					
Подготовка к загрузке			Передача данных				
Передача данных				Верификация записи	Ошибка загрузки		
Верификация записи						Перезагрузка микроконтроллера	Ошибка загрузки
Перезагрузка микроконтроллера						Тестирование	
Тестирование						Настройка соединения	
Настройка соединения							Тестирование модулей
Тестирование модулей							Анализ результатов

Применение конечных автоматов в проектировании системы обеспечивает следующие преимущества:

— Структурированное управление процессами – четкое разделение состояний позволяет легко отслеживать и понимать логику работы программы.

— Простота отладки и тестирования – благодаря явному описанию переходов между состояниями ошибки легче выявлять и исправлять.

— Предсказуемость поведения – система всегда находится в одном из заранее определенных состояний, что исключает неконтролируемые сценарии работы.

— Гибкость расширения – при необходимости можно добавить новые состояния или переходы, не изменяя основную логику работы.

Таким образом, использование конечных автоматов в проектировании специализированного аппаратно-программного средства позволяет повысить надежность системы, упростить управление процессами прошивки и тестирования.

Список литературы

1. Yunus, Shirjeel. Difference between Pushdown Automata and Finite Automata [Electronic resource] / Shirjeel Yunus // TutorialsPoint. – URL: https://translated.turbopages.org/proxy_u/en-ru.ru.5eb7073a-67caf91b-f21c9c72-74722d776562/https/www.tutorialspoint.com/difference-between-pushdown-automata-and-finite-automata (accessed: 07.03.2025).
2. Priya, Bhanu. Distinguish between Finite Automata and Turing Machine [Electronic resource] / Bhanu Priya // TutorialsPoint. – URL: https://translated.turbopages.org/proxy_u/en-ru.ru.23522479-67cadd41-627e4050-74722d776562/https/www.tutorialspoint.com/distinguish-between-finite-automata-and-turing-machine (accessed: 07.03.2025).