УДК 004.031.6

АДАПТАЦИЯ ПРОЦЕСС-ОРИЕНТИРОВАННОГО ПОДХОДА К РАЗРАБОТКЕ ВСТРАИВАЕМЫХ МИКРОКОНТРОЛЛЕРНЫХ СИСТЕМ

© **А. С. Розов**[1,2], **В. Е. Зюбин**[1,2]

[1] *Институт автоматики и электрометрии СО РАН, 630090, г. Новосибирск, просп. Академика Коптюга, 1*
[2] *Новосибирский государственный университет, 630090, г. Новосибирск, ул. Пирогова, 1*
E-mail: rozov@iae.nsk.su
zyubin@iae.nsk.su

Представлено описание адаптации процесс-ориентированного подхода к программированию микроконтроллеров во встраиваемых системах. Проведён анализ специфики алгоритмов управления и особенностей программирования микроконтроллеров. Предложена математическая модель алгоритма управления, предусматривающая механизм описания прерываний микроконтроллера в виде гиперпроцессов, и приведена её динамическая семантика. Разработанная модель предоставляет понятийный аппарат для создания специализированных языков процесс-ориентированного программирования встраиваемых систем.

Ключевые слова: встраиваемые системы, микроконтроллеры, процесс-ориентированное программирование, гиперпроцессы.

DOI: 10.15372/AUT20190212

Введение. Снижение стоимости микропроцессоров и тенденция к объединению вычислительных и периферийных устройств на одном кристалле обусловили активное развитие области встраиваемых систем. Аналоговые и специализированные цифровые схемы в электронных устройствах замещаются универсальными микроконтроллерами, и создание устройства преимущественно сводится к разработке программного обеспечения (ПО) микроконтроллера. Открытые микроконтроллерные платформы, такие как Arduino, предоставляют готовые аппаратные решения, исключая необходимость разработки специализированных электронных схем в большинстве задач. Открытость схем и исходных кодов наряду с низким порогом вхождения обеспечили высокую популярность Arduino-платформ, сделав их фактически стандартом платформ прототипирования. В настоящий момент на рынке доступен широкий выбор аналогов Arduino и совместимых модулей расширения с низкой стоимостью [1]. Аппаратная платформа для большинства встраиваемых систем может быть собрана из готовых модулей с незначительными финансовыми и временны́ми затратами. Таким образом, стоимость встраиваемых систем определяется затратами на разработку ПО микроконтроллеров, что обусловливает интерес к специализированным методикам программирования, позволяющим уменьшить эти затраты.

Применение наработок в области промышленной автоматизации может снизить затраты на разработку ПО микроконтроллеров и повысить качество встраиваемых систем. При этом необходимо учитывать отличия промышленных вычислительных платформ от микроконтроллеров, используемых во встраиваемых системах. Особенности программирования микроконтроллеров: ограниченность вычислительных ресурсов, непосредственная работа со встроенной периферией, активное применение аппаратных прерываний и ограничения по энергопотреблению — требуют модификации существующих методик и языков программирования для их эффективного использования.

Цель исследования состояла в выборе наиболее перспективного подхода и его адаптации к разработке встраиваемых систем на микроконтроллерах.

1. Постановка задачи. Эффективная разработка встраиваемых систем требует использования методик, одновременно учитывающих особенности управляющих алгоритмов и специфику программирования микроконтроллеров.

1.1. *Особенности алгоритмов управления.* Системы управления в отличие от классических вычислительных систем обладают свойством открытости — наличием внешней среды и необходимостью постоянного взаимодействия с ней. В роли внешней среды выступают объект управления и оператор системы. Система управления открыта, т. е. микроконтроллер формирует управляющие воздействия по событиям, возникающим во внешней среде. Взаимодействие с внешней средой осуществляется посредством датчиков и исполнительных устройств. В силу независимости событий, образовавшихся во внешней физической среде, в микроконтроллере требуется их параллельная обработка. Поскольку в микроконтроллерах физически отсутствуют достаточные средства для параллельной обработки событий, данное требование обеспечивается средствами логического параллелизма, реализуемого программно за счёт разделения процессорного времени. Это, в свою очередь, предполагает наличие специальных механизмов для разрешения противоречий, возникающих при работе с разделяемыми ресурсами. Дополнительно при формировании управляющих воздействий необходимо учитывать динамические характеристики внешней среды, что обусловливает наличие в микроконтроллерах программно-реализуемых средств работы с временны́ми интервалами.

Поскольку алгоритмы управления влияют на физические процессы во внешней среде, от их работы зависит безопасность обслуживающего персонала и сохранность дорогостоящего оборудования. В связи с этим к системам управления предъявляются требования надёжности и устойчивости: система должна сохранять корректное поведение на протяжении длительного времени работы в нормальных условиях и предусматривать реакцию на возникновение нештатных ситуаций.

В задачах автоматизации часто появляется необходимость корректировки поведения управляющей системы или расширения её функциональности в процессе эксплуатации. Расширяемость системы и порог вхождения использованных при разработке методов и языковых средств могут существенно влиять на стоимость доработки. Эти показатели отвечают за расширяемость системы и определяются адекватностью методик и языков решаемой задачи, а также размером области их специализации. Специализированные методы и языки программирования позволяют уменьшить размер исходного кода программ и повысить его сопровождаемость.

Таким образом, алгоритмы управления в задачах автоматизации должны обладать следующими свойствами: открытостью, событийностью, параллелизмом, синхронизмом, надёжностью, устойчивостью и сопровождаемостью [2]. Данные свойства определяют требования к методикам и языкам программирования, используемым для описания алгоритмов управления, а также справедливы при разработке встраиваемых систем на базе микроконтроллеров.

1.2. *Специфика программирования микроконтроллеров.* Основное отличие микроконтроллеров от промышленных ПК и программируемых логических контроллеров (ПЛК) — существенная ограниченность вычислительных ресурсов. В связи с ограничениями по энергопотреблению и габаритам микроконтроллерные платформы не имеют активного охлаждения и работают на относительно низких тактовых частотах (десятки МГц). Объёмы встроенной в процессор памяти также относительно малы — десятки Кбайт оперативного запоминающего устройства и сотни Кбайт постоянного запоминающего устройства. Использование внешней памяти повышает стоимость устройства, увеличивает его габариты и энергопотребление.

Появление на рынке дешёвых 32-битных микроконтроллеров с ARM-архитектурой расширило возможности реализации сложных алгоритмов: типичный пример — микроконтроллеры серии STM32F103 с тактовыми частотами до 72 МГц, объёмами флэш-памяти до 1 Мбайта и ОЗУ до 96 Кбайт. Распространению этого класса микроконтроллеров препятствуют сравнительно высокий порог вхождения (относительно PIC или AVR) и отсутствие моделей, поддерживающих работу с периферийными устройствами, рассчитанными на логические уровни от 0 до 5 В. Преимущество ARM-микроконтроллеров по объёмам памяти обесценивается тем, что 32-битная RISC-архитектура предполагает значительное увеличение размеров исполняемого кода программ в сравнении с 8-битными реализациями.

Недостаток вычислительных мощностей компенсируется наличием большого количества встроенных в микроконтроллер периферийных устройств: таймеров/счётчиков, генераторов широтно-импульсной модуляции (ШИМ), АЦП и аппаратной поддержкой низкоуровневых протоколов связи, таких как UART, SPI, I2C и других. Взаимодействие программы со встроенной периферией и внешними устройствами осуществляется преимущественно с помощью аппаратных прерываний. При этом в отличие от ПЛК, где для работы с прерываниями используются абстрагирующие программные прослойки, при программировании микроконтроллеров разработчик описывает процедуры обработки прерываний непосредственно.

Таким образом, к основным особенностям программирования микроконтроллеров относятся: ограниченность вычислительных ресурсов, непосредственное взаимодействие со встроенной периферией без операционной системы и активное применение аппаратных прерываний.

1.3. *Методики программирования микроконтроллеров.* Степень соответствия понятийного аппарата и языковых средств особенностям управляющих алгоритмов и специфике программирования микроконтроллеров — основной критерий эффективности использования языков и методов программирования микроконтроллеров.

1.3.1. *Объектно-ориентированное программирование.* Наиболее распространённые языки программирования микроконтроллеров на данный момент — языки C и C++. Разработка ведётся на базе как процедурного, так и объектно-ориентированного программирования [3].

Принципы, лежащие в основе объектно-ориентированного программирования: абстракция, инкапсуляция, композиция, полиморфизм типов — явно ориентированы на описание структуры представления данных в виде иерархии классов и объектов, что затрудняет описание систем со сложным поведением.

Попытки реализации управляющих систем на языках C/C++ приводят к образованию так называемого спагетти-кода. По мере развития и расширения системы сопровождаемость кода быстро снижается, что влечёт за собой труднодиагностируемые и трудноисправляемые ошибки в программе.

Причина этих проблем — несоответствие объектно-ориентированных принципов особенностям управляющих алгоритмов. Языки C/C++ ориентированы на разработку программ с относительно простым поведением, и их использование в области встраиваемых систем неоправданно трудоёмко.

1.3.2. *Языки стандарта МЭК 61131-3.* Наиболее распространённая платформа промышленной автоматизации — ПЛК в сочетании с языками стандарта МЭК 61131-3. Известны попытки адаптации этих языков для программирования микроконтроллерных платформ. Стандарт включает набор языков, описывающих управляющее ПО на различных уровнях с использованием разных понятийных аппаратов.

Языки стандарта имеют общие элементы — типы данных, переменные и функции (процедуры), облегчающие их совместную реализацию в одной системе. Понятийный аппарат языка ST — императивное процедурное программирование в применении к систе-

мам управления имеет те же недостатки, что и аппарат языков C/C++. Язык IL описывает программу на уровне элементарных инструкций и не имеет механизмов обеспечения событийности и параллелизма. Язык LD предназначен для описания систем, ранее реализованных на релейной логике. Язык FBD основан на концепции потоков данных, предполагает логический параллелизм и допускает использование средств синхронизации потоков управления. В нем отсутствуют средства обеспечения событийности алгоритма управления. Наибольший интерес представляет язык SFC. Понятийный аппарат языка сформирован на сетях Петри и предоставляет возможность организации событийности, параллелизма и синхронизации. Слабая структурированность программ снижает сопровождаемость и делает язык неудобным для описания сложных алгоритмов с множеством параллельных процессов [4].

Применение языков МЭК 61131-3 для создания встраиваемых систем дополнительно осложнено отсутствием поддержки работы с аппаратными прерываниями и ограниченным набором поддерживаемых микроконтроллеров.

1.3.3. *MATLAB/Simulink.* При разработке киберфизических систем, таких как в [5, 6], используется пакет MATLAB/Simulink. Управляющая система при этом описывается одновременно с моделью объекта управления на графическом языке программирования потоков данных (dataflow). Simulink предоставляет возможность генерации исполняемого кода, в том числе для микроконтроллера, непосредственно из модели. Подход эффективен при разработке систем непрерывного управления, однако dataflow-парадигма, как и в случае с языком FBD из состава МЭК 61131-3, сильно затрудняет описание поведенческих алгоритмов.

1.3.4. *Операционные системы реального времени.* Real-Time Operating Systems (RTOS) предоставляют механизмы для обеспечения параллелизма и синхронизации. Такие системы ориентированы на облегчение структурирования программы, планировку разделения времени, контроль доступа к памяти и организацию контекста исполнения процессов [7]. Значительный вклад в сокращение времени разработки на основе RTOS вносит наличие готовых библиотек «драйверов» для наиболее часто используемых периферийных устройств [8].

В основе большинства RTOS лежит планировщик, обеспечивающий вытесняющую многозадачность [9] и равномерное распределение времени между несколькими независимыми вычислительными задачами. Эта стратегия показывает хорошие результаты при постоянной загрузке процессора. Однако большинство встраиваемых систем характеризуется краткими промежутками активности с длительными периодами бездействия. В случаях автономного питания от аккумулятора и при наличии требования энергосбережения разработчики вынуждены отказываться от использования RTOS из-за отсутствия механизмов управления режимом сна [10].

Применение RTOS также значительно увеличивает риск возникновения гонок, повышает требования к объёмам используемой памяти и затраты на синхронизацию по сравнению с кооперативным вариантом организации многозадачности [11, 12].

По этим причинам поиск альтернативных подходов к организации программ на микроконтроллерах постоянно привлекает внимание исследователей. Основное направление активности — разработка специализированных языков программирования с возможностью статического анализа кода и с обнаружением семантических ошибок на этапе компиляции программ.

1.3.5. *Комбинированный подход.* В системе TinyOS [13] предпринята попытка устранить недостатки RTOS-подхода введением специализированного языка nesC [14]. Язык предоставляет высокоуровневые конструкции для обеспечения вытесняющей и частично кооперативной многозадачности. Анализ кода, производимый компилятором nesC, позволяет диагностировать места вероятного возникновения гонок. Синтаксис языка ориенти-

рован на разработку так называемой «умной пыли» (smart dust) — распределённых систем, состоящих из большого количества функционально простых микроконтроллерных модулей, и слабо подходит для спецификации встраиваемых систем широкого класса. По результатам практического использования языка разработчики обнаружили большое количество участков кода, организованных в виде конечных автоматов. Отсутствие встроенных средств организации программы в виде конечных автоматов было отмечено как недостаток подхода [15].

1.3.6. *Событийное программирование на основе конечных автоматов.* Сложное поведение алгоритмов управления наиболее успешно описывается моделями на основе конечных автоматов. Конечный автомат предполагает наличие внешней среды, представленной входом и выходом автомата. В каждом состоянии автомат сопоставляет входным значениям набор действий — реакций. Таким образом, модель автомата обладает свойством событийности. Автомат может не иметь конечного состояния и работать циклически неопределённое время. Благодаря этим свойствам конечные автоматы широко применяются для моделирования программных и аппаратных дискретных систем. Большинство алгоритмов автоматической верификации требует, чтобы система была представлена в виде конечного автомата.

Основной недостаток этой модели — отсутствие поддержки параллелизма. Попытка описать систему с множеством параллельных процессов приводит к комбинаторному взрыву сложности автомата. Даже с небольшим количеством параллельных процессов модель становится сложной для восприятия, поддержания и расширения. Для обеспечения поддержки параллелизма было предложено несколько модификаций автомата. Пример такого расширения — платформа Quantum Platform (QP), имеющая реализацию для Arduino. Платформа QP представляет систему в виде множества активных объектов (актёров). Поведение актёров описывается иерархическим конечным автоматом Hierarchical State Machine (HSM) [16]. Разработка ПО в QP ведётся на графическом языке с последующей кодогенерацией из диаграмм состояний (statecharts).

Формализм иерархических конечных автоматов, используемый в QP, разрабатывался для борьбы с комбинаторным ростом числа состояний путём уменьшения избыточности автомата за счёт переиспользования кода состояний. Модель HSM затрудняет описание параллелизма, поскольку алгоритм управления задаётся одним автоматом.

1.3.7. *Процесс-ориентированное программирование.* Другой способ решения проблем комбинаторного взрыва сложности и облегчения восприятия системы — представление её в виде набора отдельных параллельно функционирующих автоматов [17]. В методике процесс-ориентированного программирования [2] эта идея используется для описания алгоритмов управления.

В основе процесс-ориентированного программирования лежит понятие гиперпроцесса — набора взаимодействующих процессов. Каждый процесс реализуется конечным автоматом. Модель автомата расширена операциями работы с данными и межпроцессного взаимодействия.

Методика процесс-ориентированного программирования реализована в специализированном языке Reflex [18, 19], ориентированном на создание систем промышленной автоматизации на базе ПЛК. Этот язык показал высокую эффективность при разработке алгоритмов управления в промышленных задачах [20–23].

Модель гиперпроцесса обеспечивает событийность, логический параллелизм и синхронизм алгоритмов управления, удовлетворяя всем требованиям задач автоматизации. Специфика программирования микроконтроллеров состоит в необходимости дополнительной поддержки работы с аппаратными прерываниями. Существующие реализации языка Reflex и модели гиперпроцесса предполагают абстракцию обработки прерываний в отдельном программном слое. В таком случае средствами Reflex используется только высокоуров-

невая часть алгоритма управления, а обработка прерываний обеспечивается средствами языка C. Создание и поддержание такой системы предполагают работу с двумя языками программирования, что приводит к повышению требований к квалификации разработчика. Эффективная методика программирования микроконтроллеров должна допускать реализацию всей программы единым набором средств.

Таким образом, в исследовании поставлена задача разработки понятийного аппарата, одновременно позволяющего эффективно описывать алгоритм управления и учитывающего активное использование аппаратных прерываний микроконтроллера.

2. Предлагаемое решение. Для решения поставленной задачи понятийный аппарат языка Reflex предложено расширить следующим образом. Программа представляется множеством гиперпроцессов H, а гиперпроцесс — множеством процессов P, начальным процессом p_1 и источником активации a:

$$h \equiv \langle a, P, p_1 \rangle.$$

В качестве источника активации может выступать основной цикл программы или аппаратное прерывание. Таким образом, множество гиперпроцессов делится на циклически активируемый фоновый гиперпроцесс h_{bkg} и гиперпроцессы $h_{\mathrm{Int}\,1}, \ldots, h_{\mathrm{Int}\,N}$, активируемые прерываниями

$$H \equiv \langle h_{bkg}, h_{\mathrm{Int}\,1}, \ldots, h_{\mathrm{Int}\,N} \rangle.$$

Процесс p_i представляется машиной состояний с множеством функций-состояний F_i, выделенным начальным состоянием f_i^1, текущим состоянием f_i^{cur} и состоянием останова f_i^{stop}, а также временем нахождения в текущем состоянии t_i^p:

$$p_i \equiv \langle F_i, f_i^1, f_i^{cur}, f_i^{stop}, t_i^p \rangle.$$

Функция-состояние f_i^j описывается множеством событий E_i^j, множеством реакций R_i^j, отображением g_i^j, задающим соответствие реакций событиям, и временем тайм-аута T_i^j:

$$f_i^j \equiv \langle E_i^j, R_i^j, g_i^j, T_i^j \rangle, \qquad g_i^j : E_i^j \to R_i^j.$$

Событие e_i^{jk} — произвольная суперпозиция фактов, задаваемая через логические операции над значениями входных сигналов, значениями текущих функций-состояний процессов (проверка активности процесса `active(p)`) и значениями t_p (проверка тайм-аутов процессов `timeout(p)`). В качестве реакций рассматриваются вычислительные операции, операции запуска и останова процессов `start(p)`, `stop(p)`, операция сброса тайм-аута `reset(p)` и операция перехода в другое состояние `set_state(p,f)`. Выделяются безусловные реакции процесса, соответствующие тождественно-истинному событию e_i^{j0}.

2.1. *Динамическая семантика модели алгоритма управления.* Вышеописанная структура алгоритма управления предусматривает наличие аппаратных прерываний. Для демонстрации динамических свойств этой модели определим её основные составляющие с точки зрения порядка исполнения. Активация гиперпроцесса заключается в последовательном исполнении текущих функций-состояний входящих в него процессов:

$$h \equiv (f_1^{cur}; \ldots; f_N^{cur}).$$

При начальной активации гиперпроцесса выделенный начальный процесс p_1 находится в начальной функции-состоянии f_1^1, остальные процессы — в состоянии останова f_i^{stop}.

Таблица 1

Динамическая семантика служебных событий

Событие	Семантика
$\texttt{timeout(p)} \equiv (t_p \geqslant T_{f_p^{cur}})$	Проверка тайм-аута процесса
$\texttt{active(p)} \equiv \neg(f_p^{cur} = f_p^{stop})$	Проверка активности процесса

Таблица 2

Динамическая семантика служебных операций

Операция	Семантика
$\texttt{start(p)} \equiv (f_p^{cur} := f_p^1;\ t_p := 0)$	Запуск процесса
$\texttt{stop(p)} \equiv (f_p^{cur} := f_p^{stop})$	Остановка процесса
$\texttt{set\_state(p,f)} \equiv (f_p^{cur} := f;\ t_p := 0),\ f \in F_p$	Смена состояния процесса
$\texttt{reset(p)} \equiv (t_p := 0)$	Сброс тайм-аута процесса

Исполнение алгоритма управления представляется как циклическая активация фонового гиперпроцесса и асинхронная активация остальных гиперпроцессов по возникновению соответствующих прерываний:

$$H \equiv h_{bkg}*, \qquad \mathrm{Int}_1 \to h_{\mathrm{Int}\,1},\ \ldots,\ \mathrm{Int}_M \to h_{\mathrm{Int}\,M}.$$

Динамическая семантика служебных событий и операций дана в табл. 1 и 2.

Пример спецификации функции-состояния f_1^1 процесса p_1:

$$\begin{aligned}
&f_1^1\text{:}\\
&e^0 \to x := x + 1\\
&\neg\ \texttt{active}(p_2) \to (\texttt{set_state}(p_1, f_1^3);\ y := x)\\
&(w = 0 \wedge z \geqslant 5) \to (z := 0)\\
&\texttt{timeout}(p_1) \to (\texttt{set_state}(p_1, f_1^2);\ \texttt{stop}(p_2)).
\end{aligned}$$

Тождественно-истинному событию e^0 соответствует безусловная реакция $x := x + 1$ изменения значения сигнала x, при обнаружении останова процесса p_2 осуществляются переход в состояние f_1^3 и изменение выходного значения сигнала y. По событию $(w = 0 \wedge z \geqslant 5)$ значений сигналов w и z происходит изменение выходного значения сигнала z. При возникновении тайм-аута процесса происходят переход в состояние f_1^2 и останов процесса p_2.

Заключение. В представленном исследовании проведён обзор подходов к разработке программного обеспечения встраиваемых систем на микроконтроллерах. Предложена математическая модель алгоритма управления, предусматривающая механизм описания прерываний микроконтроллера в виде гиперпроцессов, описана динамическая семантика служебных событий и операций. Данная модель и её динамическая семантика предоставляют понятийный аппарат для создания специализированных языков процесс-ориентированного программирования встраиваемых систем.

Финансирование. Работа выполнена в рамках государственного задания (государственная регистрация № АААА-А17-117060610006-6).

СПИСОК ЛИТЕРАТУРЫ

1. **Якубайлик О. Э., Кадочников А. А., Токарев А. В.** Геоинформационная веб-система и приборно-измерительное обеспечение оперативной оценки загрязнения атмосферы // Автометрия. 2018. **54**, № 3. С. 39–46.

2. **Зюбин В. Е.** Язык Рефлекс. Математическая модель алгоритмов управления // Датчики и системы. 2006. № 5. С. 24–30.
3. **Kormanyos C.** Real-Time C++. Berlin — Heidelberg: Springer, 2018. P. 61–84.
4. **Зюбин В. Е.** К пятилетию стандарта IEC 1131-3. Итоги и прогнозы // Приборы и системы. Управление, контроль, диагностика. 1999. № 1. С. 64–71.
5. **Белоконь С. А., Золотухин Ю. Н., Филиппов М. Н.** Архитектура комплекса полунатурного моделирования систем управления летательными аппаратами // Автометрия. 2017. **53**, № 4. С. 44–50.
6. **Белоконь С. А., Золотухин Ю. Н., Нестеров А. А.** Планирование маршрутов движения летательного аппарата с использованием гладких траекторий // Автометрия. 2017. **53**, № 4. С. 44–50.
7. **Wroldsen T., Tveitane T.** A real time operating system for embedded platforms: Masters thesis. Norway: Agder University College, 2004. 87 p.
8. **Bichu T., Kaingade S., Walambe A., Gupta N.** RTOS based software architecture for intelligent unmanned systems // Proc. of the Intern. Conf. on Intelligent Unmanned Systems. Jaipur, India, 25–27 Sept., 2013. P. 1–8.
9. **Rahman H. M., Senthil A.** Preemptive multitasking on Atmel AVR microcontroller // Proc. of the 9th Intern. Conf. on Computer Engineering and Applications (CEA 2015). Dubai, Unighted Arab Emirates, 22–24 Febr., 2015. P. 196–205.
10. **Simonovic M., Saranovac L.** Power management implementation in FreeRTOS on LM3S3748 // Serbian Journ. Electrical Eng. 2013. **10**, N 1. P. 199–208.
11. **Short M., Pont M. J., Fang J.** Exploring the impact of task preemption on dependability in time-triggered embedded systems: A pilot study // Proc. of the EuroMicro Conf. on Real-Time Systems. Prague, Czech Republic, 2–4 July, 2008. P. 83–91.
12. **Robert D., Merriam N., Tracey N.** How embedded applications using an RTOS can stay within on-chip memory limits // Proc. of the 12th EuroMicro Conf. on Real-Time Systems. Stockholm, Sweden, 19–21 June, 2000. P. 71–77.
13. **Levis Ph., Madden S., Polastre J. et al.** Ambient intelligence. Berlin — Heidelberg: Springer, 2005. P. 115–148.
14. **Gay D., Levis Ph., Culler D., Brewer E.** NesC 1.1 language reference manual, 2003. 28 p. URL: http://nescc.sourceforge.net/papers/nesc-ref.pdf (дата обращения: 19.10.2018).
15. **Gay D., Levis Ph., von Behren R. et al.** The nesC language: A holistic approach to networked embedded systemsn // Proc. of the Conf. on Programming language design and implementation (PLDI 2003). San Diego, USA, 9–11 June, 2003. P. 1–11.
16. **Harel D.** Statecharts: A visual formalism for complex systems // Sci. Computer Program. 1987. **8**, N 3. P. 231–274.
17. **Wagner F., Schmuki R., Wagner T., Wolstenholme P.** Modeling software with finite state machines: A practical approach. Boca Raton: Auerbach Publications, 2006. 369 p.
18. **Zyubin V. E.** Hyper-automaton: A model of control algorithms // Proc. of the IEEE Intern. Siberian Conf. on Control and Communications (SIBCON-2007). Tomsk, Russia, 20–21 April, 2007. P. 51–57.
19. **Liakh T. V., Rozov A. S., Zyubin V. E.** Reflex language: A practical notation for cyber-physical systems // System Informatics. 2018. N 12. P. 85–104.
20. **Зюбин В. Е.** Процесс Чохральского: создание системы управления на основе пакета LabVIEW // Матер. VIII Междунар. конф. по актуальным проблемам физики, материаловедения, технологии и диагностики кремния, наноразмерных структур и приборов на его основе «Кремний-2011». М.: Изд. дом «МИСиС». С. 96–97.

21. **Степанова Т. Н., Зюбин В. Е.** Автоматизация исследований роста монокристаллов методом Чохральского на физическом имитаторе // Матер. VIII Междунар. конф. по актуальным проблемам физики, материаловедения, технологии и диагностики кремния, наноразмерных структур и приборов на его основе «Кремний-2011». М.: Изд. дом «МИСиС». С. 119–120.
22. **Ковадло П. Г., Лубков А. А., Бевзов А. Н. и др.** Система автоматизации Большого солнечного вакуумного телескопа // Автометрия. 2016. **52**, № 2. С. 97–106.
23. **Лях Т. В., Зюбин В. Е., Сизов М. М.** Опыт применения языка Reflex при автоматизации Большого солнечного вакуумного телескопа // Промышленные АСУ и контроллеры. 2016. № 7. С. 37–43.

Поступила в редакцию 19.10.2018
После доработки 09.01.2019
Принята к публикации 24.01.2019