УДК 621.316.9

А.В.ПРИЛИПКО, студент НТУ "ХПИ"
В.С. ТРОЙНИКОВ, канд. техн. наук, доц. НТУ "ХПИ"

ИССЛЕДОВАНИЕ И ОПИСАНИЕ ПЛАТФОРМЫ QUANTUM LEAPS

У статті розглянута платформа для написання програм для умонтовуваних систем, Quantum Leaps, яка є альтернативою традиційним операційним системам реального часу.

The platform for a writing of programs for embedded systems, Quantum Leaps which is alternative to traditional operating systems of real time in this article.

Постановка проблемы. В связи с повышающимися требованиями заказчиков к функциональности приборов и ответного роста производительности всех вычислительных систем, становится всё сложнее писать новые программы и сопровождать старые. Такие платформы ,как и операционные системы, призваны помочь разработчику справится со множеством проблем связанными с необходимостью написания самому ту же операционную систему и сопровождать её в дальнейшем за самим прибором. Теряется много времени на разработку внутреннего механизма, прикладная часть проекта приобретает второстепенный приоритет, хотя именно о ней должен думать разработчик, когда получит заказ.

Анализ публикаций. В обзоре литературных источников были рассмотрены возможности применения операционных систем для микроконтроллеров с относительно малым объемом памяти небольшими временными ресурсами. Как альтернативный вариант обычной многозадачной ОС, была выбрана платформа Quantum Leaps(QL) исходные коды которой для обучения и экспериментов распространяются свободно и бесплатно.

Целью настоящей статьи является исследование и описание платформы для написания программ для встраиваемых систем.

Материалы и результаты исследования. Почти все компьютерные системы вообще, и встроенные системы в частности являются событийно управляемыми, что означает, что они непрерывно ждут возникновения некоторого внешнего или внутреннего события, такого как тик таймера, прибытие пакета данных, нажатия кнопки, или щелчка мыши. После прихода события такие системы реагируют, выполняя соответствующее вычисление, которое может включать управление аппаратными средствами или выполнение "софт" событий, которые вызывают другие внутренние компоненты программного обеспечения. (Именно поэтому управляемые событием системы

так же называют событийными системами.). Как только обработка события завершена, программное обеспечение возвращается к ожиданию следующего.

Большинство разработчиков приучены к простому последовательному контролю, в котором программа ждет событий в различных местах пути ее выполнения или активно опрашивает события или пассивно блокирует на семафоре или другом каком-то механизме операционной системы. Хотя этот подход к программированию управляемых событиями систем функционален во многих ситуациях, это не слишком хорошо работает, когда есть многократные возможные источники событий, время прибытия которых невозможно предсказать и где важно обрабатывать события во время их поступления. Проблема состоит в том, что, в то время как последовательная программа ждет одного вида события, она не делает никакой другой работы и не чувствительна к другим событиям.

Необходима структура программы, которая может ответить на множество возможных событий, любое из которых может прийти в непредсказуемое время и в непредсказуемой последовательности. Хотя эта проблема очень распространена во встраиваемых системах, таких как домашние приборы, сотовые телефоны, индустриальные диспетчеры, медицинские устройства и многие другие, она также очень распространена в современных настольных компьютерах. К примеру, возьмем Web-браузер, словарный процессор, или крупноформатной таблицы. У большинства этих программ есть современный графический пользовательский интерфейс, который естественно способен к обработке с многократных событий. Все разработчики современных систем, приняли общую структуру программы, которая изящно решает проблему контакта со многими асинхронными событиями в реальном времени. Эту структуру программы вообще называют событийно управляемым программированием.

Инверсия контроля. Событийно управляемое программирование требует совсем иного мышления, не такого как при обычном последовательном программировании, в таком как "супер петля" или задача в традиционной RTOS. Большинство современных событийно управляемых систем структурировано согласно Голливудскому принципу, что означает, "не вызывайте нас, мы назовем вас.” Таким образом событийно управляемая программа не контролирует приход события; фактически, она даже не активна. Подпрограмма вызывается, только, при возникновении события, чтобы обработать его, и затем она снова быстро отдаёт контроль. Эта договоренность позволяет управляемой событием системе ждать многих событий параллельно, таким образом, система остается восприимчивой ко всем событиям, на которые она должна реагировать.

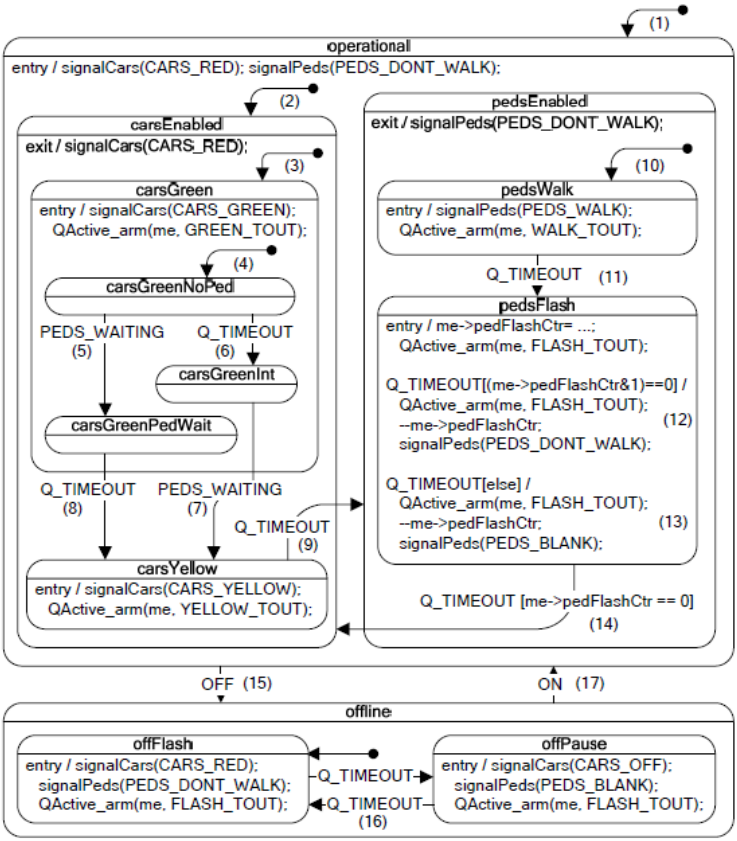

Рис. 1. Диаграмма состояний светофора

У этой схемы есть три важных последствия. Во-первых, это подразумевает, что событийно управляемая система естественно разделена на прикладную часть, которое фактически обращается с событиями, и контролирующую, управляющую инфраструктурой, которая ждет событий и посылает их заявлению. Во-вторых, контроль осуществляется в событийно управляемой инфраструктуре, таким образом, с прикладной точки зрения контроль инвертирован по сравнению с традиционной последовательной программой. И в-третьих, событийно управляемое приложение должно возвратить контроль после обработки каждого события, таким образом, контекст выполнения не может быть сохранен в основанных на стеке переменных и счётчике программы, поскольку он находится в последовательной программе. Вместо этого событийно управляемое приложение становится машиной состояний, или

фактически рядом сотрудничающих машин состояний, которые сохраняют контекст от одного случая до следующего в статических переменных.

На рисунке 1 представлена диаграмма состояний дорожного светофора, которая даётся разработчиком в качестве примера для изучения механизмов работы этой платформы. На диаграмме можно видеть состояния как закруглённые прямоугольники и стрелки которые обозначают переходы из состояния в состояния по сигналам. Выполняемые действия связаны сигналами. Сигналы генерируются как самим пользователем в ходе программы, так и автоматически по программному таймеру при входе в состояние при выходе из него и в других ситуациях.

QL платформа(QP) представляет собой реализацию на языке C/C++ машины состояний. Основные термины которыми оперируют разработчики на этой платформе относятся к диаграммам состояний UML. Программист описывает «Активный Объект» как совокупность его «Состояний», переходов между этими состояниями по «Сигналам». Специальная версия QP для «маленьких» контроллеров – QP-nano может занимать всего 2 Кбайта памяти программы и всего 100 байт оперативной памяти контроллера, что делает её интересным вариантом при выборе платформы проектирования. Для примера типичные операционные системы занимают на порядок, а то и на два больше ресурсов

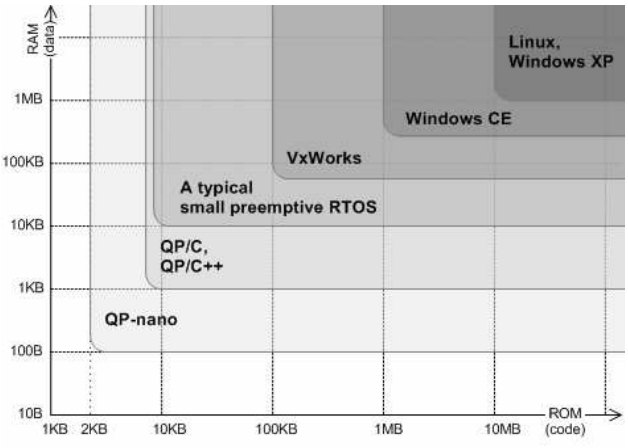

Рис. 2. Сравнение потребляемых ресурсов памяти

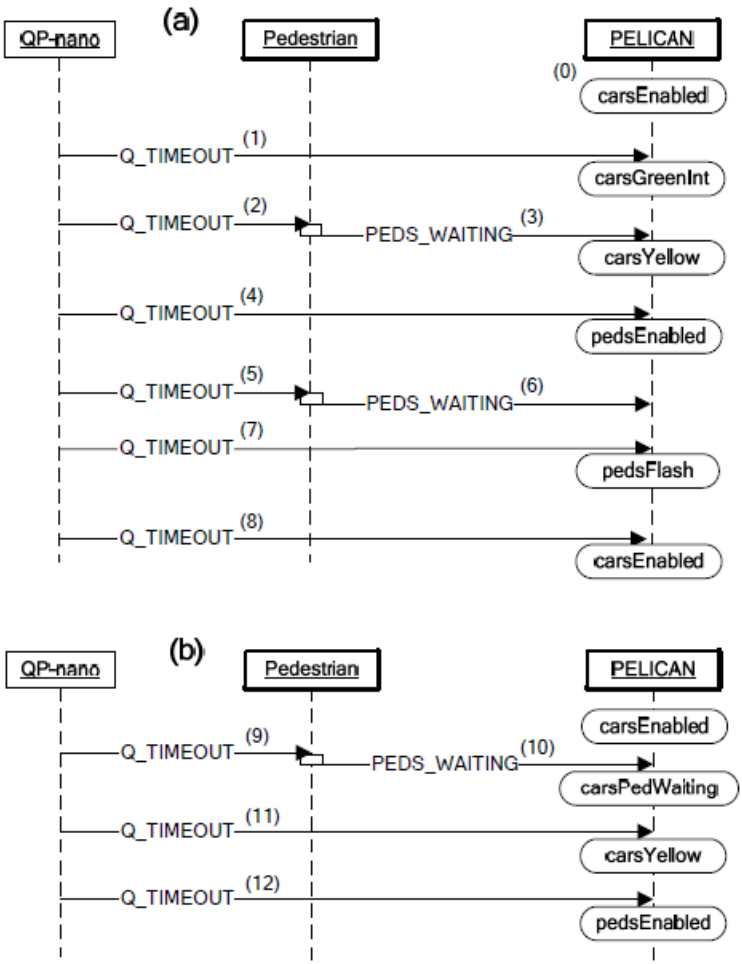

Рис. 3. Пример передачи сигнала от пешехода к светофору

QP не является операционной системой реального времени, в классическом понимании RTOS. Она накладывает определённые ограничение на программиста который будет писать программу, но тем самым давая ему возможность самому достаточно точно определить критические места программы, приоритеты. Для создания эффективной программы необходимо правильно определить что нужно выделить как отдельный активный объект, какие состояния он имеет на какие сигналы должен реагировать, какие сигналы генерировать для других активных объектов. Современные микроэлектрон-

ные устройства имеют точно определённую модульную структуру, и при использовании QP можно без труда определить каждый модуль как отдельный активный объект. Описать активный объект и его состояния как того требует реально существующий модуль по логике его работы и наладить взаимодействие между активными объектами по средствам сигналов.

В примере QP-nano (рис.3) два таких активных объекта представлены в виде светофора и пешехода а сигналами между ними нажатия кнопок пешеходом на светофоре.

Ограничения, которые ложатся на программиста, не столь гнетущи, нужно стараться делать подпрограммы обработки отдельных сигналов как можно меньшими по времени, возлагая управление временем на плечи QP.

Выводы. 1. Исследована платформа написания программ для встраиваемых систем. Описаны основные особенности её устройства и работы. 2. Приведены обоснования её использования. Рассмотрен пример использования системы на обучающем примере.

Материалы статьи будут использованы при разработке экспертной системы для проектирования устройств на базе микроконтроллера. В дальнейших исследованиях предполагается расширить рассматриваемую область применения различных платформ и надстроек для встраиваемых систем.

Список литературы: 1. Материалы с сайта http://www.state-machine.com/ **2**. Практическое руководство по программированию машин состояний на C/C++ Practical UML Statecharts in C/C++, 2nd Edition **3**. Справочная информация с сайта http://atmel.com /dyn/ products/ datasheets_v2.asp ?family_id = 607 **4.** Application Note PEdestrian LIght CONtrolled Crossing Example

Статья представлена д.т.н. проф. НТУ «ХПИ» Дербуновичем Л.В.

Поступила в редакцию 09.05.2010