В настоящее время в основе работы таких систем лежит алгоритм анализа изменения среднего уровня сигнала. Однако в источнике [1] была рассмотрена возможность использования вейвлет-функции Френеля (формула 1) для повышения показателей качества детектирования нарушителя.

$$w(t) = e^{-\frac{t^2}{c^2}} \cos(g \cdot t^2). \quad (1)$$

Параметры g и c рассчитываются согласно формулам 2 и 3.

$$g = \frac{4\pi V_n^2}{\lambda R_m (1 - \frac{(2x_n)^2}{R_m^2})}, \quad (2)$$

где V_n – скорость перемещения нарушителя (м/с); x_n – расстояние от нарушителя до центра охраняемой зоны (м); R_m – ширина охраняемой зоны (м).

$$c = \frac{\sqrt{\frac{4\pi(N-1)}{g}}}{3{,}03}, \quad (3)$$

где N – количество осцилляций, используемых при анализе.

В работе представлен сравнительный анализ показателей надежности двух выше представленных алгоритмов для объектов с низкой скоростью перемещения (не более 3 м/с).

Согласно источнику [1] для обеспечения точности анализа достаточно 5 осцилляций. Для моделирования работы системы необходимо определить ее основные характеристики. Расстояние приемник-передатчик в двупозиционных радиолучевых системах охраны составляет от 100 до 500 метров. Для исследований будет использоваться среднее значение в 300 метров. Длина несущей волны обычно составляет 0,1-0,2 метра. Для исследований будем считать, что длина волны - 0,1 метра. Моделирование будет производиться в критической точке, где ширина главного лепестка функции Френеля минимальна. Этой точкой является максимально близкое расстояние от нарушителя до приемника или передатчика – граница «мертвой зоны» охранной системы. В РЛ ТСО ширина мертвой зоны обычно составляет 0,5 метра.

Подставляя значения в формулы 1, 2 и 3 получаем вейвлет-функцию, представленную на рисунке.

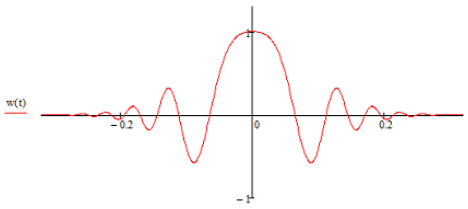


Рис. 1. График вейвлет-функции Френеля

Для моделирования представим информационный сигнал S(t) в виде суммы вейвлет-функции Френеля, характеризующего перемещения нарушителя через охраняемую зону, и "белого шума", моделирующего влияние окружающей среды на информационный сигнал (формула 5).

$$S(t) = S(w(t) + S_{\text{шум}}(t) + 1), \quad (5)$$

где $w(t)$ – вейвлет-функция Френеля, $S_{\text{шум}}(t)$ – «белый шум», S – постоянный уровень сигнала.

Для проверки системы на ложные срабатывания сформируем модель сигнала, без воздействия нарушителя, представляющую собой сумму постоянного уровня сигнала и «белого шума» (формула 6).

$$S(t) = S \cdot (1 + S_{\text{шум}}(t)). \quad (6)$$

При анализе на основе среднего уровня среднее значение в идеальном случае равен постоянной составляющей сигнала. Для моделируемой системы примем это значение равное 5.

Для вейвлет-анализа результат преобразования при отсутствие нарушителя должен быть равен квадрату постоянной составляющей сигнала. Для данной модели - 25.

Моделирование производилось в среде программирования QtCreator 3.2.1 на языке C++ с использованием стандартных библиотек.

Результаты моделирования системы представлены в табл. 1 и 2.

Таблица 1

Результаты моделирования системы на основе анализа среднего значения

Номер эксперимента	Эталонное значение	Значение среднего уровня	Относительно отклонение, %
При наличии нарушителя			
1	5	6,73	34,6
2		6,04	20,8
3		5,85	17
4		6,43	28,6
5		5,83	16,6
При отсутствии нарушителя			
6	5	5,14	2,8
7		4,82	3,6
8		4,73	5,4
9		4,96	8
10		4,99	2

Таблица 2

Результаты моделирования системы на основе вейвлет-преобразования

Номер эксперимента	Эталонное значение	Значение вейвлет-преобразования	Относительное отклонение, %
При наличии нарушителя			
1	25	30,38	21,52
2		30,41	21,64
3		29,87	19,48
4		30,75	23
5		29,82	19,23
При отсутствии нарушителя			
6	25	23,21	7,1
7		24,67	1,32
8		22,78	8,88
9		24,22	3,12
10		26,48	5,92

При отсутствии нарушителя отклонение расчетного значения от эталона не превышает 10% показателя.

При прохождении нарушителя значение на выходе системы имеет более высокое значение, вследствие возникновения интерференционных максимумов у сигнала. Отклонения от эталона составляют от 15 до 30%, что позволяет однозначно детектировать нарушителя с почти 100% вероятностью.

Однако при грубом анализе [3] (отклонение более 50% от эталонного значения) может возникнуть не срабатывание системы. Тогда для повышения вероятности срабатывания системы требуется сузить временной интервал наблюдения.

Таким образом, оба алгоритма обладают достаточно высоким показателями надежности для обработки сигналов двупозиционных РЛ ТСО. Но в то же время алгоритм анализа среднего значения более предпочтителен для решения таких задач, в связи с простотой реализации и меньших временных затрат на выполнение.

Список литературы

1. Сальников И.И. Растровые пространственно-временные сигналы в системах анализа изображений. М.:Физматлит, 2009.

ОСНОВНЫЕ ТЕНДЕНЦИИ ПРОЕКТИРОВАНИЯ ОПЕРАЦИОННЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

ПашкинА.В., Шмокин М.Н.

Пензенский государственный технологический университет
Пенза, Россия, e-mail: los@pgta.ru

Введение

Обычно операционные системы реального времени *(RTOS)* реализованы как программное обеспе-

чение, но все больше и больше встраиваемых систем разрабатываются на *FPGA* платформе.*FPGA* может быть использована при ускоренном параллельном выполнении различных задач.

Основными недостатками стандартного программного обеспечения на основе RTOS является то, что они страдают от нагрузки на вычислительные мощности и часто требуют большой объем памяти.

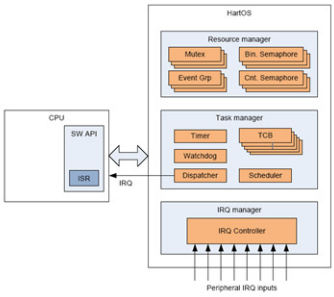
Требования высокой производительности (вычислительной мощности)RTOS обусловлены в основном блоком управления прерываниями, который снижает производительность с ростом количества задач и высоких частотах, но и планирования задач, ресурсов распределение и освобождение, обнаружение тупиков и различные другие функции ОС / API, потребуется время выполнения задач, выполняемых на CPU.

Внешняя обработка асинхронных прерываний также является источником индетерминизма, так как это будет вызывать неопределенность уровня приоритета, чтобы обеспечить завершение задачи в течение их времени выполнения.

Повышенные требования предъявляются к памяти программного обеспечения *RTOS*, отчасти это связано с необходимостью создания стека для каждой задачи, но и требования в объеме пространства как ОЗУ и ПЗУ, для хранения структурных данных *RTOS*, так как программный код может занимать достаточно большой объем.Разработанным решением является HartOS, аппаратное обеспечение реализовано в операционной системе реального времени, поддерживающей все основные функции, ожидаемые от полной RTOS без ущерба для гибкости.

Упрощенная блок-схема системы

HartOS состоит из двух главных частей: программное обеспечение *API* и твердое - ядро изделия. Рисунок показывает первоначальный проект блок-схемы системы.



Блок-схема системы

Программное обеспечение *API* будет содержать все функции интерфейса, необходимые для связи с ядром, а также *CPU* определенного функционала для обработки контекстных переключений, инициализирующих стековые фреймы и т.д.

Аппаратное ядро будет содержать всю функциональность *RTOS*. Его структура состоит из трёх главных частей: Диспетчер задач, Менеджер по IRQ/Прерыванию и Менеджер ресурсов.

Диспетчер задач будет решать все функции, необходимые для реализации базируемого ядра чистой задачи. Главные функции этого: Таймер, Сторожевой таймер, Диспетчер и модули Планировщика. Массив *TCB's* (Блок управления задачей) будет содержать необходимые данные, чтобы управлять и восстановить контекст задач в системе. Менеджер по *IRQ* реализует контроллер прерываний и логику, необходимую для взаимодействия через интерфейс с остальной частью ядра. Менеджер ресурсов должен управлять ресурсами ядра.

Особенности общейархитектуры

Как иллюстрировано в рисунке 1 *HartOS* будет состоять из трех главных (ядро) модулей: Диспетчер задач, диспетчер прерываний и менеджер ресурсов. Главная цель такой архитектуры состоит в том, чтобы сделать каждый из этих модулей максимально независимым, таким образом, модуль может быть легко удален, должна также быть, возможность обновить/изменить внутреннюю функциональность модуля, не затрагивая остальную часть системы.

Заключение

Исходя из рассмотренных публикаций, можно сказать, что основные недостатки программного обеспечения на основе *RTOS* могут быть ликвидированы путем реализации всего ядра операционной системы в режиме реального времени на аппаратном уровне. Продуманный дизайн и использование новейших технологий *FPGA*, позволяет реализовать полнофункциональные и гибкие аппаратные*RTOS (HartOS*).

Список литературы

1. J. Lee, I. Mooney, V.J., A. Daleby, K. Ingstrom, T. Klevin, and L. Lindh, “Acomparison of the rtu hardware rtos with a hardware/software rtos,” pp. 683 – 688, jan. 2003.

2. S. Nordstrom, L. Lindh, L. Johansson, and T. Skoglund, “Application specific real-time microkernel in hardware,” p. 4 pp., jun. 2005.

3. S. Nordstrom and L. Asplund, “Configurable hardware/software support for single processor real-time kernels,” pp. 1 –4, nov. 2007.

АНАЛИЗ СПЕЦИАЛИЗИРОВАННЫХ СИСТЕМ ОБРАБОТКИ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ

Печаткин С.В., Грачева Е.В.

Пензенский государственный технологический университет, Пенза, Россия, e-mail: los@pgta.ru

Введение

Концепция параллельной обработки давно привлекала внимание специалистов своими потенциальными возможностями повышения производительности и надежности вычислительных систем. В последнее время эстафета параллельных вычислений перешла к массовому рынку, связанному с трёхмерными приложениями. Универсальные устройства с многоядерными процессорами для параллельных векторных вычислений, используемых в 3D-графике, достигают высокой пиковой производительности, которая универсальным процессорам не достижима.

Сравнение CPU и GPU в параллельных расчётах

Для 3D видеоускорителей ещё несколько лет назад появились первые технологии неграфических расчётов общего назначения GPGPU (General-Purpose computation on GPUs). Современные видеочипы содержат сотни математических исполнительных блоков, и эта мощь может использоваться для значительного ускорения множества интенсивных приложений. Современные GPU обладают гибкой архитектурой, что вместе с высокоуровневыми языками программирования и программно-аппаратными архитектурами, раскрывает эти возможности и делает их значительно более доступными.

На создание GPCPU разработчиков побудило появление достаточно быстрых и гибких шейдерных программ, которые способны исполнять современные видеочипы. Разработчики задумали сделать так, чтобы GPU рассчитывали не только изображение в 3D приложениях, но и применялись в других параллельных расчётах. В GPGPU для этого использовались графические API: OpenGL и Direct3D, когда данные к видеочипу передавались в виде текстур, а расчётные программы загружались в виде шейдеров. Недостатками такого метода является:

- сравнительно высокая сложность программирования;

- низкая скорость обмена данными между CPU и GPU и другие ограничения.