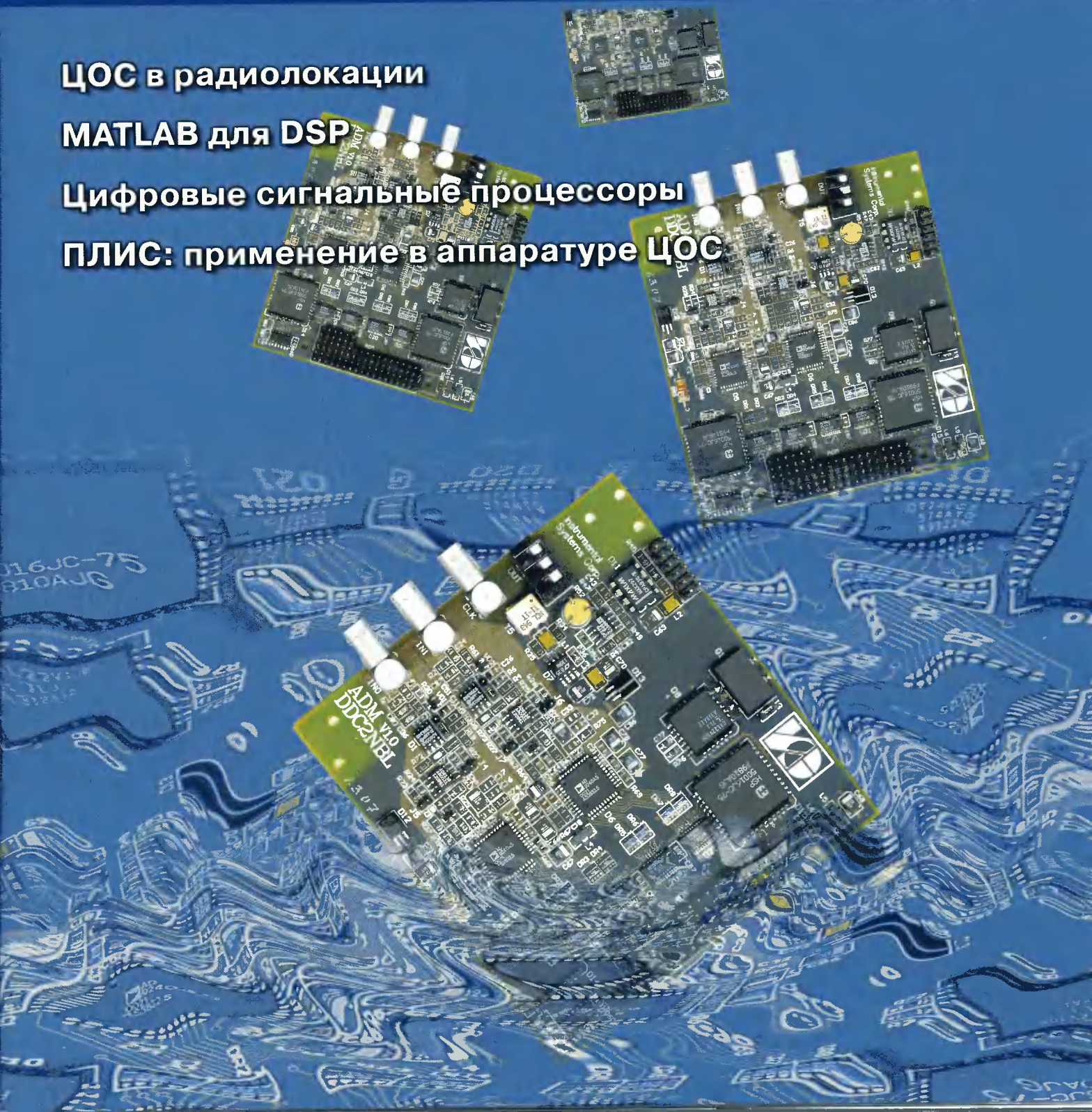
ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ

2/2000

НАУЧНО - ТЕХНИЧЕСКИЙ ЖУРНАЛ



ЦОС в радиолокации

MATLAB для DSP

Цифровые сигнальные процессоры

ПЛИС: применение в аппаратуре ЦОС

ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ

№ 2/2000

Адрес редакции:
103064 Москва, ул. Казакова, 16
Тел./факс: (095) 267-0662
E-mail: niircom@ccs.ru

Учредители:
НИИ радио, АО "Инструментальные системы" при участии Отделения информатики, вычислительной техники и автоматизации РАН

Цифровая обработка сигналов
№ 2/2000, с. 1 – 48

Редактор В.Н. Вяльцев
Дизайн обложки И.В. Валентик
С.Г.Тюканова
Верстка С.Г. Тюканова
Корректура Е.В. Карасева
Н.Н. Сергеева

Издание зарегистрировано в Министерстве Российской Федерации по делам печати, телерадиовещания и средств массовых коммуникаций
Свидетельство о регистрации
ПИ № 77-1488 от 14.01.00

В НОМЕРЕ:

А.Г. Соловьев

Тракт цифровой обработки сигналов когерентной импульсно-доплеровской РЛС

В данной работе описан тракт цифровой обработки сигналов (ЦОС) радиолокационного комплекса (РЛК) управления воздушным движением. Рассматривается тракт цифровой обработки сигналов когерентной доплеровской радиолокационной станции. Тракт выполняет функции цифрового радиоприема, согласованной фильтрации сложного сигнала, подавления пассивных помех и обнаружения. Приведена структурная схема алгоритмов обработки, рассмотрено и обосновано распределение функций алгоритма по устройствам цифровой обработки. Отмечены особенности реализации алгоритмов на сигнальном процессоре TMS320C6201. Приведены результаты экспериментального исследования характеристик тракта.

Комплекс РЛК представляет собой двухкоординатный (азимут, наклонная дальность) импульсный обзорный радиолокатор L-диапазона с внутренней когерентностью и твердотельными передатчиком и приемником. Зондирование осуществляется двумя импульсами: коротким (1,5 мкс) без внутриимпульсной модуляции и длинным (120 мкс) линейно-частотно-модулированным (ЛЧМ) сигналом с полосой 1,08 МГц. Основными особенностями системы обработки данного РЛК являются переход к цифровым устройствам обработки и формирования сигнала на промежуточной частоте (ПЧ) и применение цифрового фазового детектора (ЦФД).

Поскольку одной из основных задач тракта обработки является селекция движущихся целей (СДЦ), важно обеспечить высокий коэффициент подавления пассивных помех. Применение ЦФД позволило устранить многие недостатки аналоговых фазовых детекторов, которые ограничивали степень подавления. ЦФД не имеет нестабильной во времени постоянной составляющей, сигналы на выходах его квадратурных каналов строго ортогональны. Данные недостатки преодолимы и при применении аналоговых фазовых детекторов, но это требует значительных аппаратных затрат и не гарантирует полного их устранения. Кроме того, при применении ЦФД за счет дискретизации сигнала на ПЧ и последующей цифровой фильтрации повышается динамический диапазон тракта обработки при той же разрядности АЦП. Применение твердотельного передатчика и цифрового формирователя зондирующего сигнала на ПЧ позволило обеспечить высокую степень когерентности в РЛК.

Тракт ЦОС выполняет следующие функции:

1. Синхронизация между устройствами обработки и формирование сигнала по фазе и времени.
2. Дискретизация и квантование сигнала ПЧ 30 МГц 12-разрядным АЦП с частотой выборок 24 МГц. При этом приемник обеспечивает подав-

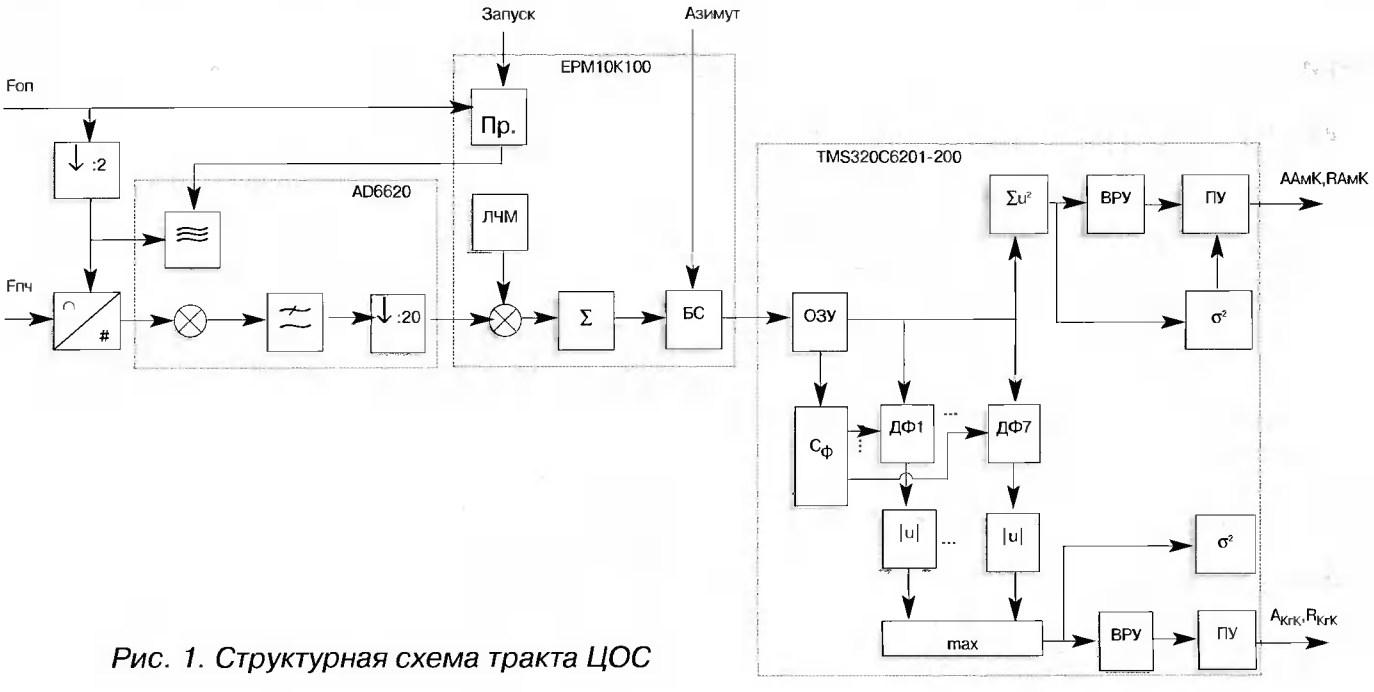


Рис. 1. Структурная схема тракта ЦОС

ление помех и шумов на входе АЦП вне полосы 24 ... 36 МГц.
3. Формирование квадратурных каналов с помощью ЦФД.
4. Цифровая фильтрация для формирования АЧХ-тракта необходимой формы и понижение частоты дискретизации до 1,2 МГц при односторонней полосе сигнала 0,54 МГц.
5. Корреляционная обработка ЛЧМ-сигнала.
6. Объединение дистанций короткого и длинного импульсов после сжатия в одну.
7. Селекция движущихся целей и формирование когерентного канала.
8. Некогерентное накопление сигнала в амплитудном канале.
9. Объединение сигналов разных частотных каналов РЛК до пороговой обработки.
10. Стабилизация уровня ложных тревог (нормировка), пороговая обработка и формирование кодограмм отметок для передачи их в систему первичной обработки информации (ПОИ).

Получены следующие характеристики тракта ЦОС:

1. Отношение сигнал-шум на входе ЦОС при вероятности обнаружения 0.9 и вероятности ложной тревоги 10^{-6} — 13,5 дБ.
2. Суммарные энергетические потери при обработке 2,3 дБ.
3. Динамический диапазон по выходу ЦФД 72 дБ, по выходу коррелятора 85 дБ, по выходу накопления 90 дБ.
4. Уровень боковых лепестков на выходе коррелятора –38 дБ.
5. Коэффициент подавления пассивных помех более 60 дБ.

Тракт ЦОС одного частотного канала реализован на одной плате типа ADP67PCI с установленным субмодулем типа ADMDDC2WB фирмы «Инструментальные системы». Функции 2, 3, 4 выполняет субмодуль, функции 1, 5, 6 — установленная на плате ПЛИС EPM10K100 фирмы Altera, функции 7 — 10 — цифровой процессор обработки сигналов (ЦПОС) TMS320C6201 фирмы Texas Instruments с тактовой частотой 200 МГц. Структурная схема тракта ЦОС показана на *рис. 1*.

Дискретизированные с частотой 24 МГц отсчеты сигнала ПЧ поступают на цифровой фазовый детектор на микросхеме AD6620. Подробную структурную схему данного ЦФД можно найти в [1]. Особенностью применения здесь является сброс фазы опорного генератора ЦФД, настроенного на 6 МГц. Сброс фазы (сброс регистра-накопителя фазы) осуществляется по привязанному к опорной частоте импульсу запуска (импульсу начала зондирования). Необходимость в этом возникает из-за применения цифрового способа формирования зондирующего сигнала на промежуточной частоте. Микросхема AD6620 имеет необходимые входные сигналы для реализации сброса фазы, изначально предназначенные для синхронизации нескольких ЦФД в системе. Здесь же решается задача взвешивания принятого импульса в частотной области для снижения уровня боковых лепестков сжатого импульса. Взвешивание в частотной области позволило освободить ресурсы в ПЛИС коррелятора. Кроме этого, из-за ограничений по длине импульсной характеристики фильтра ЦФД, накладываемых AD6620, АЧХ, соответствующую взвешивающей функции, удается реализовать существенно более точно и с большим подавлением вне полосы пропускания, чем прямоугольную АЧХ. При максимальной степени приближения АЧХ к прямоугольной и высоком внеполосном подавлении оказываются либо слишком высокая неравномерность в полосе прозрачности частотной характеристики при малой ширине ската характеристики, либо протяженный скат характеристики при приемлемой неравномерности. Последний случай фактически сводится к взвешиванию в частотной области.

Комплексный сигнал с ЦФД поступает на коррелятор, работающий во временной области. Известно, что свертка с применением быстрого преобразования Фурье требует меньше вычислительных затрат. Однако быструю свертку всей рабочей дистанции не удалось реализовать из-за ограничений по объему памяти. При применении секционированной быстрой свертки [2] требуется меньший объем памяти, но усложняются алгоритм самой свертки и алгоритм передачи данных в ЦПОС. Кроме того, для реализации БПФ при заданном динамическом диапазоне входных сигналов требуется большее число разрядов арифметических устройств, чем для реализации свертки во временной области. Коррелятор рассчитан на ЛЧМ-сигнал с той же скоростью изменения частоты, что и принятый, но с двусторонней полосой, точно равной частоте дискретизации сигнала. При этом за счет симметрии фазовой структуры сигнала удается снизить число умножителей в четыре раза по сравнению с прямой реализацией, не учитывающей структуру сигнала. Возникающие при этом энергетические потери пропорциональны отношению частоты дискретизации и двусторонней полосы сигнала и составляют 0,46 дБ.

ПО ЦПОС написано на языке «С» с использованием языковых расширений компилятора фирмы Texas Instruments и с учетом архитектурных особенностей процессора TMS320C6201. Для получения максимальной производительности все вычислительные операции производятся в памяти на кристалле процессора. Внешняя по отношению к процессору память используется для буферизации входных и выходных данных. Перенос данных внутрь кристалла осуществляется по каналам прямого доступа к памяти по возможности крупными блоками данных.

Прием 16-разрядных комплексных отсчетов с коррелятора осуществляется следующим образом. Коррелятор по очередному импульсу запуска очищает свое выходное FIFO и генерирует прерывание процессору. Процессор по данному прерыванию программирует канал ПДП на прием кодов номера импульса, азимута и данных очередной

рабочей дистанции, запускает передачу по ПДП и разрешает выдачу данных коррелятору. Таким образом осуществляется привязка данных к нулю дистанции. По концу приема рабочей дистанции контроллер ПДП вырабатывает прерывание, по которому проверяется правильность кодов номера импульса и азимута и переставляется указатель записи входного FIFO программы. FIFO имеет большой (1000 рабочих дистанций, около $^1/_3$ обзора) объем, что позволяет имитировать или документировать выходные данные коррелятора в большом секторе. Обработка ведется в скачущем через три периода повторения окне, т.е. последние восемь периодов повторения обрабатываются каждый третий период повторения.

Блок формирования когерентного канала представляет собой гребенку комплексных нерекурсивных 8-точечных режекторных фильтров. Каждый фильтр обеспечивает заданное подавление эхо-сигналов от местных предметов и максимум коэффициента передачи на своей частоте Доплера. Для достижения наилучших характеристик по подавлению коэффициенты фильтров выбираются в соответствии с текущим периодом повторения, т.е. в соответствии с текущим положением скачущего окна обработки относительно вобулированной последовательности зондирующих импульсов. Количество фильтров выбрано исходя из минимума неравномерности скоростной характеристики с одной стороны и имеющегося ресурса по производительности процессора с другой стороны и составляет семь. На выходах фильтров вычисляются модули комплексных отсчетов. Выходы фильтров объединяются устройством выбора максимума.

Таким образом, фильтр когерентного канала реализуется в соответствии с выражением

$$\dot{u}^{(i)}_{k,m} = \sum_{j=0}^{7} \dot{c}^{(i)}_{j,m} \cdot \dot{u}^{вх}_{k,m-j}, \qquad (1)$$

где k — дискрет дальности; m — текущий период повторения; i — номер фильтра; $\dot{u}^{(вх)}$ — входные комплексные отсчеты; $\dot{c}$ — комплексные коэффициенты фильтров. Коэффициенты фильтров рассчитаны для заданной расстановки периодов повторения по методике, изложенной в [3]. Используются умножения 16 на 16 разрядов с 32-разрядным результатом и 32-разрядное суммирование. Модуль комплексных отсчетов с выходов фильтров вычисляется по приближенной формуле

$$U \approx \max \begin{vmatrix} \frac{7}{8}|\mathrm{Re}(\dot{u})| + \frac{1}{2}|\mathrm{Im}(\dot{u})| \\ \frac{1}{2}|\mathrm{Re}(\dot{u})| + \frac{7}{8}|\mathrm{Im}(\dot{u})| \\ |\mathrm{Re}(\dot{u})| \\ |\mathrm{Im}(\dot{u})| \end{vmatrix}. \qquad (2)$$

Ошибка вычислений по данной формуле не превышает 4% во всем диапазоне аргументов комплексного отсчета. Точный расчет модуля не удается реализовать из-за ограничений, накладываемых производительностью.

Блок формирования амплитудного канала выполняет функции вычисления мощностей отсчетов эхо-сигналов и суммирования мощностей в восьми периодах повторения, т.е. в соответствии с выражением

$$U^{a\ 2}_{k,m} = \sum_{j=0}^{7} |\dot{u}^{вх}_{k,m-j}|^2. \qquad (3)$$

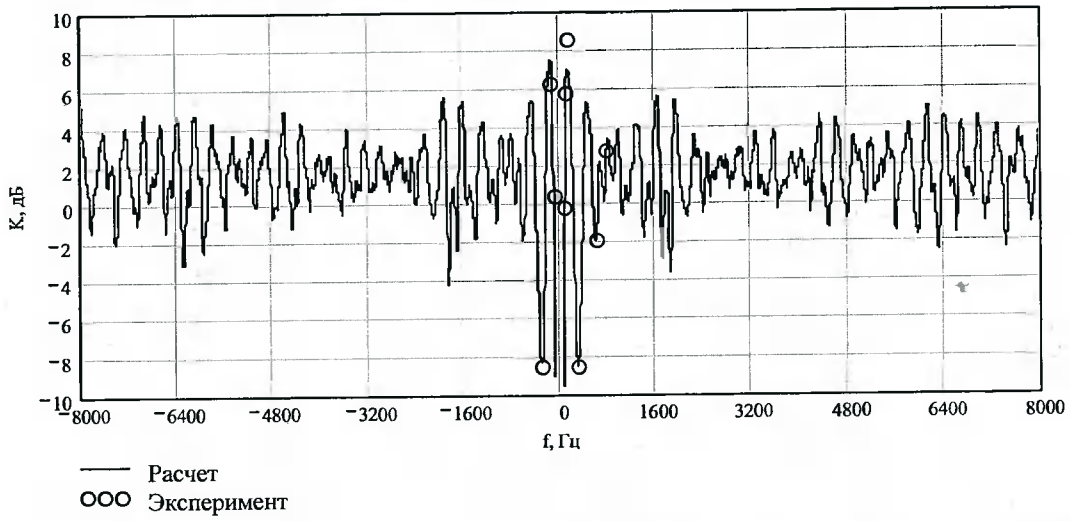
Используются умножения 16 на 16 разрядов с 32-разрядным результатом и насыщением и 32-разрядное суммирование с насыщением.

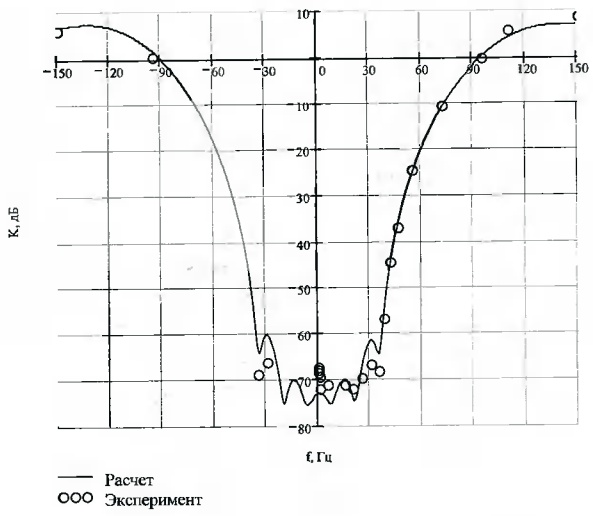
С выходов амплитудного и когерентного каналов данные после масштабирования поступают на сумматоры. На эти же сумматоры поступают данные амплитудного и когерентного каналов соседнего частотного канала по последовательному порту процессора СДЦ. Ячейки обработки частотных каналов располагаются рядом, поэтому длина кабелей последовательного канала связи не превышает нескольких сантиметров. Из-за ограничений, накладываемых пропускной способностью канала связи и размером памяти на кристалле процессора СДЦ, возможно суммирование либо только амплитудного, либо только когерентного каналов. После суммирования с соседним частотным каналом данные поступают на блоки стабилизации уровня ложной тревоги (СУЛТ) и пороговой обработки.

Тревога СУЛТ амплитудного канала представляет собой так называемый медианный фильтр, который свободен от многих недостатков классической СУЛТ со скользящим средним (подавление обнаружения целей вблизи интенсивных отражений от местных предметов) и позволяет более точно построить карту местных предметов [4]. Вместе с тем такой фильтр требует значительных вычислительных затрат. Медианный фильтр представляет собой скользящее по дальности окно, в котором производятся ранжирование входных отсчетов по амплитуде и выбор отсчета с рангом, равным половине длины окна, т.е. из середины ранжированного ряда.

Тревога СУЛТ когерентного канала реализована как два скользящих по дальности окна слева и справа от текущего отсчета дальности с последующим выбором максимальной из сумм отсчетов в левом и правом окнах. Два окна используются для исключения из вычисления суммы самой цели, максимальная из сумм выбирается для уменьшения количества ложных тревог в областях с резким изменением интенсивности остатков пассивных помех. Отказ от медианной СУЛТ в когерентном канале вызван ограниченным ресурсом процессора СДЦ по производительности с одной стороны и тем, что пассивные помехи в когерентном канале в

Рис. 2. Скоростные характеристики когерентного канала





значительной степени подавлены режекторными фильтрами СДЦ с другой стороны. Сумма отсчетов из скользящего окна СУЛТ когерентного канала подается на блок пороговой обработки.

Перед подачей на пороговые устройства сигналы амплитудного и когерентного каналов проходят блоки временной регулировки усиления (ВРУ), масштабирующих сигналы в зависимости от дальности. Это необходимо для уменьшения влияния высокого по сравнению с динамическим диапазоном тракта обработки уровня боковых лепестков диаграммы направленности антенны (ДНА). В отсутствие ВРУ на малых дальностях объекты с большой ЭПР будут образовывать пачки обнаруженных эхо-сигналов как по основному лепестку ДНА, так и по боковым.

На вход ПОИ поступают амплитуды превысивших пороги отсчетов из амплитудного и когерентного каналов с указанием номера дискрета дальности и азимута. На вход ПОИ также выдаются сами пороги амплитудного и когерентного каналов (с 8-кратным прореживанием по дальности) для построения карты пассивных помех и выделения малоскоростных целей.

На *рис. 2* представлены расчетные и экспериментальные скоростные характеристики. При экспериментальном снятии характеристики с помощью устройства формирования зондирующих импульсов формировался сигнал с доплеровским смещением частоты. Цифровой способ формирования импульса и доплеровского смещения позволил задавать последнее с дискретностью в 0,01 Гц. Затем через регулируемый аттенюатор сигнал подавался на вход устройства обработки. В зависимости от заданного доплеровского смещения затухание аттенюатора устанавливалась так, чтобы вероятность обнаружения на выходе когерентного канала составляла 0,9 при вероятности ложной тревоги 10^{-6}. Оценка вероятностей производилась в реальном масштабе времени за время около одной минуты. При построении графиков из величины затухания вычиталась величина затухания при отключенной гребенке фильтров когерентного канала.

Из анализа графиков можно сделать вывод, что выбранная элементная база и технические решения позволили с высокой точностью реализовать заданные параметры тракта обработки.

ЛИТЕРАТУРА

1. *Шлеев С.Е.* Элементная база и архитектура цифровых радиоприемных устройств. // Цифровая обработка сигналов, 1999. – №1. С.36 — 47.
2. *Цифровая обработка сигналов.* Справочник / Гольденберг Л.М., Матюшкин Б.Д., Поляк М.Н. М.: Радио и связь, 1985. – 312 с.
3. *Бакулев П.А., Степин В.М.* Методы и устройства селекции движущихся целей. – М.: Радио и связь, 1986. – 288 с.
4. *Линкевичюс С.П.* Обнаружение нестационарных участков помехового фона // Радиотехника, 1999. – №9. – С.24 — 28.

И.В. Андреев, А.А. Ланнэ

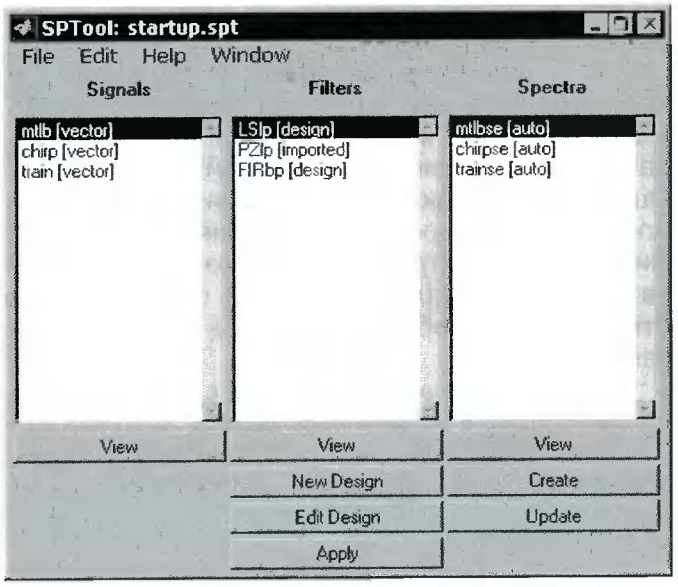
MATLAB для DSP

SPTool — инструмент для расчета цифровых фильтров и спектрального анализа сигналов

Для решения задач проектирования цифровых фильтров (ЦФ) важно получение передаточной функции и ее представление в нужной форме. SPTool в прямом виде не обеспечивает решение этой задачи. Однако использование средств MATLAB позволяет элементарно расширить возможности GUI и, таким образом, сделать SPTool в известном смысле замкнутой системой, позволяющей решать задачи синтеза ЦФ и спектрального анализа сигналов в их классической постановке от начала и до конца. Сказанное в полной мере относится к задаче получения сигналов для обработки. Они могут быть созданы средствами MATLAB либо введены извне. Упомянутые вопросы рассмотрены в статье.

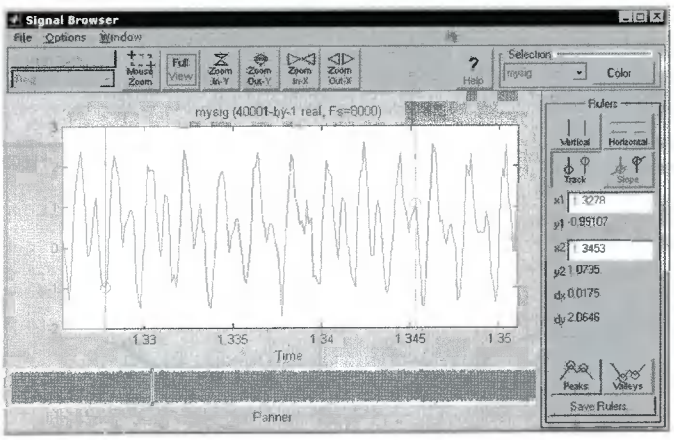
Графический интерфейс пользователя (GUI) SPTool представляет собой совокупность интерактивных инструментов, обеспечивающих удобный, наглядный и быстрый доступ к ряду важных функций цифровой обработки сигналов (ЦОС), содержащихся в Signal processing toolbox — наборе инструментов (коротко тулбокс), встроенных в вычислительную среду MATLAB и поддерживающий широчайший спектр операций обработки сигналов. SPTool поддерживает самые популярные операции ЦОС — фильтрацию и спектральный анализ. При этом пользователю предоставляются наиболее известные методы обработки, в деталях описанные в учебниках и общеизвестных монографиях.

Рис. 1. Главное окно SPTool



Дополнительно SPTool обеспечивает множество других возможностей: экспорт и импорт сигналов и спектров, измерение ряда их характеристик и параметров, возможности наблюдения и прослушивания, построение графиков и т.д.

Рис. 2. Компонента Signal Browser



1. Описание SPTool

1.1. Подключение SPTool

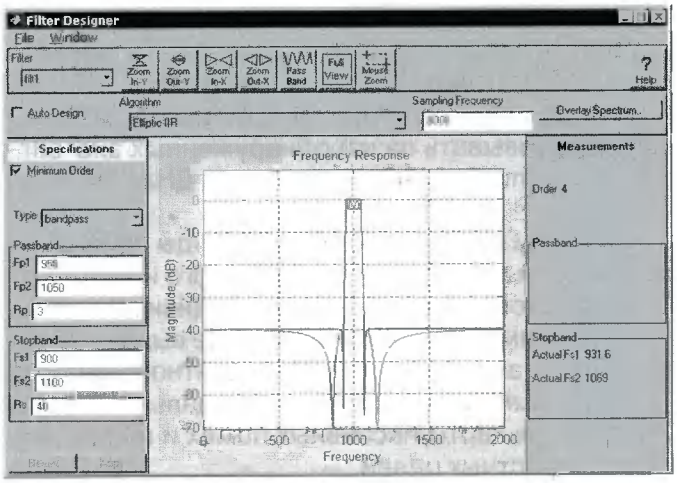
Интерфейс SPTool запускается из командного окна MATLAB командой 'sptool'.

Главное окно SPTool (*рис. 1*) содержит три окна (компоненты): Signals, Filters и Spectra, а также командное меню. Каждое окно служит контейнером для объектов соответствующего типа. Например, в окне Signals можно видеть импортированные в SPTool сигналы, в Filters — импортированные или синтезированные фильтры и соответственно в контейнере Spectra содержатся рассчитанные оценки спектра сигналов.

1.2. Структура SPTool

SPTool состоит из четырех графических компонент, которые составляют интегрированную среду для просмотра, расчета, анализа и измерений. Эти четыре компоненты включают:

Рис. 3. Компонента Filter Designer



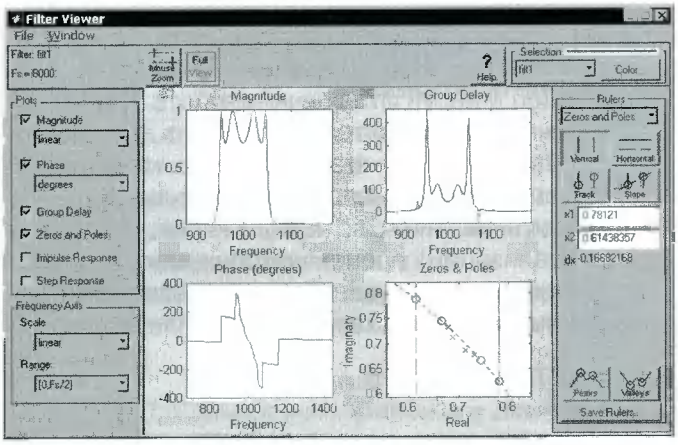
1. **Signal Browser** (*рис.2*) — предоставляет возможности просмотра, измерения и анализа выбранных в SPTool цифровых сигналов
2. **Filter Designer** (*рис.3*) — позволяет различными методами рассчитывать и редактировать (модифицировать требования, изменять методы расчета и т.д.) фильтры нижних и высоких частот, полосовые и режекторные БИХ- и КИХ-фильтры.
3. **Filter Viewer** (*рис.4*) — дает возможность полностью проанализировать все характеристики рассчитанного фильтра, включая амплитудно-частотные (АЧХ) и фазо-частотные характеристики (ФЧХ), групповое время прохождения (ГВП), расположение нулей и полюсов на комплексной плоскости, импульсную (ИХ) и переходную характеристики (ПХ).
4. **Spectrum Viewer** (*рис.5*) — позволяет исследовать выбранные сигналы в частотной области с использованием различных методов вычисления спектра.

1.3. Меню SPTool

File Menu:

❑ **Open Session** — с помощью этой команды можно загрузить сохраненный ранее сеанс работы с SPTool;

Рис. 4. Компонента Filter Viewer



❑ **Import** — импорт в SPTool различных структур данных (сигналов, фильтров, спектров) из рабочего пространства[1] MATLAB или из MAT-файлов;

❑ **Export** — экспорт сигналов, фильтров и спектров из SPTool (в виде специальных переменных-структур) в рабочее пространство MATLAB или в MAT-файл;

❑ **Save Session и Save Session As...** — позволяет сохранить текущий сеанс работы SPTool в файл на диске;

❑ **Preferences** — конфигурация SPTool;

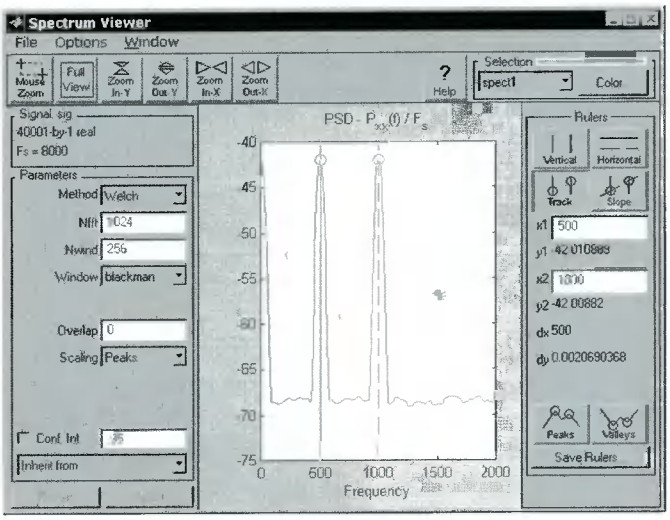
❑ **Close** — выход из SPTool.

Edit Menu — позволяет копировать, удалять и переименовывать сигналы, фильтры и спектры.

Help Menu — предоставляет общую и контекстно-зависимую справку по работе с SPTool.

[1] *Рабочее пространство MATLAB – совокупность всех переменных, определенных в MATLAB в текущий момент времени; рабочее пространство может быть сохранено в файле (обычное расширение – MAT).*

Рис. 5. Компонента Spectrum Viewer



1.4. Импорт данных в SPTool

После выбора пункта меню 'File->Import' SPTool предоставляет возможность переноса данных из рабочего пространства MATLAB или из MAT-файла в рабочее пространство SPTool. Например, нужно перенести данные из рабочего пространства MATLAB. Для этого нужно отметить пункт **From Workspace**. Как видно из *рис.6*, список переменных, определенных в текущий момент в рабочем пространстве MATLAB, отображается в окне **Workspace Contents**. Переменные из этого списка могут быть импортированы в SPTool в виде сигнала, фильтра, спектра или как значение частоты дискретизации и других параметров. Импорт переменных из MAT-файла производится аналогично импорту из рабочего пространства MATLAB.

К сожалению, возможности импорта SPTool ограничены и, для того чтобы получить сигнал не из MAT-файла и не из рабочего пространства, а из произвольного файла на диске, необходимо проделать дополнительные манипуляции.

Предположим, файл с сигналом находится на диске по следующему пути: c:\work\samples\fspeech. Формат сигнала, дискретизированного с частотой 8000 Гц, простой — целочисленный в 16-битном дополнительном коде (по два байта на отсчет, и файл не содержит никакого заголовка). Для того чтобы импортировать этот сигнал в пространство SPTool, необходимо предварительно считать его в рабочее

Рис. 6. Окно импорта данных

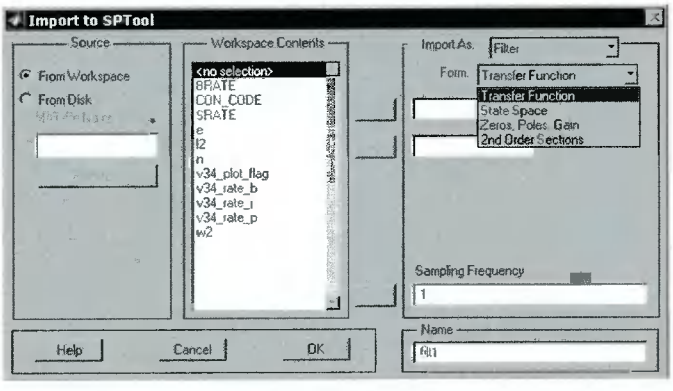
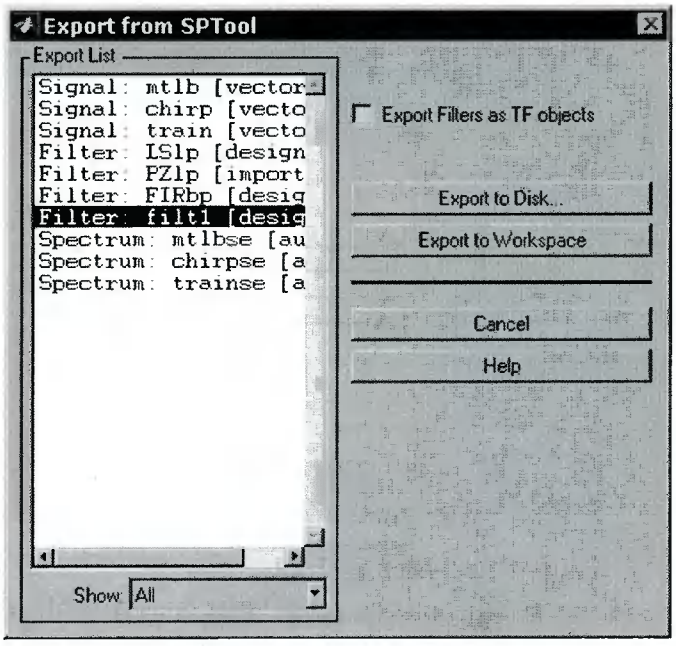


Рис. 7. Окно экспорта данных



пространство MATLAB следующей последовательностью команд:

```
file = fopen( 'c:\work\samples\fspeech', 'rb' );
sig = fread( file, inf, 'int16' );
```

Командой fopen ( ) открывается указанный файл:

- первый параметр — путь к файлу;
- второй параметр означает, что файл открывается для чтения бинарных данных.

Если файл будет открыт успешно, в переменную file будет занесено значение, используя которое в будущем будут производиться все обращения к этому файлу, в случае ошибки переменная file будет равна –1.

Командой fread ( ) производится чтение данных из файла в рабочее пространство MATLAB:

- первый параметр — идентификатор открытого файла;
- второй параметр означает, что нужно считать весь файл до конца;
- и третий параметр указывает команде считать данные в файле 16-разрядными целыми числами

После успешного выполнения этих команд в рабочем пространстве MATLAB появится переменная sig — вектор, содержащий значения отсчетов указанного сигнала. Далее сигнал можно перенести в SPTool выбором в окне импорта переменной sig, нажатием кнопки '->' напротив поля 'Data' и указанием в поле 'Sampling frequency' соответствующей частоты дискретизации — 8000 Гц.

1.5. Экспорт данных из SPTool

Выбрав пункт меню 'File->Export' можно переносить объекты, находящиеся в контейнерах SPTool в рабочее пространство MATLAB или в MAT-файл. На *рис.7.* показано окно экспорта. Слева находится список имен объектов, определенных в SPTool, с указанием их типов — Signal, Filter или Spectrum.

После экспорта какого-либо объекта в рабочем пространстве MATLAB появляется переменная-структура, содержащая разный набор полей в зависимости от типа объекта (*табл. 1*).

2. Синтез фильтров с использованием SPTool

2.1. Общие положения

Задача синтеза фильтров состоит из двух основных этапов. На первом этапе (аппроксимации) конструируется передаточная функция фильтра. Этот этап может выполняться с помощью GUI. На втором этапе (реализации) происходит декомпозиция передаточной функции, что в известной степени эквивалентно построению схемы фильтра или алгоритма

Таблица 1. Экспортируемые из SPTool структуры

Экспортированный объект	Поля соответствующей структуры	
Сигнал	.data[]	массив, содержащий значения отсчетов сигнала
	.Fs	частота дискретизации сигнала, Гц
Фильтр	.tf	структура, содержащая коэффициенты передаточной функции
	.Fs	частота дискретизации, для которой рассчитан фильтр, Гц
	.specs	структура, содержащая информацию о расчете фильтра: указывают метод расчета и в зависимости от метода параметры расчета
Фильтр (экспортированный как класс 'tf')	в этом случае появляется переменная класса 'tf', которую далее можно использовать в Control System Toolbox	
Спектр	.P	массив, содержащий значения плотности спектра мощности в линейной шкале, каждое такое значение соответствует значению частоты из поля .f
	.f	массив, содержащий значения частот, соответствующих значениям плотности из массива .P; значения частоты, Гц, в диапазоне от –Fs/2 до Fs
	.Fs	частота дискретизации сигнала, для которого была рассчитана оценка спектра
	.specs	информация о методе и параметрах оценки спектра

Таблица 2. Список спецификаций расчета фильтров различными методами с ручным определением степени передаточной функции

Спецификации расчета фильтров (нет галочки напротив Minimun Order)				
	НЧ	**ВЧ**	**Полосовой**	**Режекторный**
Equiripple FIR	Fp,Fs,Wp,Ws	Fp,Fs,Wp,Ws	Fp1,Fp2,Fs1,Fs2,Wp,Ws	Fp1,Fp2,Fs1,Fs2,Wp,Ws
Least Squares FIR	Fp,Fs,Wp,Ws	Fp,Fs,Wp,Ws	Fp1,Fp2,Fs1,Fs2,Wp,Ws	Fp1,Fp2,Fs1,Fs2,Wp,Ws
Kaiser Window FIR	Fc,Beta	Fc,Beta	Fc1,Fc2,Beta	Fc1,Fc2,Beta
Butterworth IIR	F3db	F3db	F3db1,F3db2	F3db1,F3db2
Chebyshev Type I IIR	Fp,Rp	Fp,Rp	Fp1,Fp2,Rp	Fp1,Fp2,Rp
Chebyshev Type II IIR	Fs,Rs	Fs,Rs	Fs1,Fs2,Rs	Fs1,Fs2,Rs
Elliptic IIR	Fp,Rp,Rs	Fp,Rp,Rs	Fp1,Fp2,Rp,Rs	Fp1,Fp2,Rp,Rs

где Wp, Ws — веса, позволяющие регулировать точность аппроксимации АЧХ в полосе пропускания и полосе задерживания соответственно;
Fc — частота среза, Гц;
Beta — параметр *b* для оконной функции Кайзера;
F3db — граница полосы пропускания, на которой затухание составляет 3 дБ, Гц.

преобразования входного сигнала в выходной. Второй этап исполняется инструментами тулбокса.

Нажатием кнопок 'New Design' или 'Edit Design' в главном окне SPTool активизируется компонента синтеза фильтров — Filter Designer.

Окно расчета фильтров содержит следующие элементы управления и отображения (*рис.2*):

- выпадающее меню выбора способа аппроксимации;
- область отображения АЧХ рассчитываемого фильтра с возможностью вручную подстраивать некоторые параметры синтеза;
- область отображения карты нулей и полюсов для прямого задания передаточной функции фильтра;
- панель спецификаций фильтра (Specifications), содержащую поля для отображения и ввода параметров синтеза фильтра (тип фильтра, частоту среза, уровень затухания и др.);
- панель измерений (Measurements), предназначенную для вывода характеристик рассчитанного фильтра;
- панель управления графическим отображением.

Процедуру синтеза можно осуществлять как расчетом передаточной функции по одному из предлагаемых ниже алгоритмов (первый способ), так и путем ручной расстановки нулей и полюсов на комплексной плоскости — эвристическим конструированием (второй способ).

2.2. Конструирование передаточной функции (задача аппроксимации)

Для синтеза по первому способу предварительно необходимо задать тип фильтра — нижних (lowpass), высоких (highpass), полосовой (bandpass) или режекторный (bandstop) и выбрать способ аппроксимации. SPTool предоставляет три варианта построения КИХ-фильтров и четыре — БИХ-фильтров.

Передаточные функции КИХ-фильтров:

1. Equiripple FIR — расчет передаточной функции КИХ-фильтра по алгоритму Ремеза (синтезируется фильтр с минимально возможной максимальной ошибкой аппроксимации АЧХ при заданной длине

Таблица 3. Список измеряемых в окне Measurements характеристик рассчитываемых фильтров

Minimum Order — галочка есть				
	НЧ	**ВЧ**	**Полосовой**	**Режекторный**
Equiripple FIR	Порядок,Rp,Rs,Wp,Ws	Порядок,Rp,Rs,Wp,Ws	Порядок,Rp,Rs,Wp,Ws	Порядок,Rp,Rs,Wp,Ws
Least Squares FIR	Для этого метода режим автоматического расчета порядка не поддерживается			
Kaiser Window FIR	Порядок,Fc,	Порядок,Fc,	Порядок,Fc1,Fc2,	Порядок,Fc1,Fc2,
	Beta,Rp,Rs	Beta,Rp,Rs	Beta,Rp,Rs	Beta,Rp,Rs
Butterworth IIR	Порядок,Rp,	Порядок,Rp,	Порядок,Rp,	Порядок,Rp,
F3db	F3db	F3db1,F3db2	F3db1,F3db2	
Chebyshev Type I IIR	Порядок,Fs	Порядок,Fs	Порядок,Fs1,Fs2	Порядок,Fs1,Fs2
Chebyshev Type II IIR	Порядок,Fs	Порядок,Fs	Порядок,Fs1,Fs2	Порядок,Fs1,Fs2
Elliptic IIR	Порядок,Fs	Порядок,Fs	Порядок,Fs1,Fs2	Порядок,Fs1,Fs2
Minimum Order — галочки нет				
	НЧ	**ВЧ**	**Полосовой**	**Режекторный**
Equiripple FIR	Rp,Rs	Rp,Rs	Rp,Rs	Rp,Rs
Least Squares FIR	Rp,Rs	Rp,Rs	Rp,Rs	Rp,Rs
Kaiser Window FIR	Fp,Fs,Rp,Rs	Fp,Fs,Rp,Rs	Fp1,Fs1,Fp2,Fs2,Rp,Rs	Fp1,Fs1,Fp2,Fs2,Rp,Rs
Butterworth IIR	Fp,Fs,Rp,Rs	Fp,Fs,Rp,Rs	Fp1,Fs1,Fp2,Fs2,Rp,Rs	Fp1,Fs1,Fp2,Fs2,Rp,Rs
Chebyshev Type I IIR	Fp,Fs,Rs	Fp,Fs,Rs	Fp1,Fs1,Fp2,Fs2,Rs	Fp1,Fs1,Fp2,Fs2,Rs
Chebyshev Type II IIR	Fp,Fs,Rp	Fp,Fs,Rp	Fp1,Fs1,Fp2,Fs2,Rp	Fp1,Fs1,Fp2,Fs2,Rp
Elliptic IIR	Fs	Fs	Fs1,Fs2	Fs1,Fs2

фильтра (порядке передаточной функции) — минимаксный фильтр, или фильтр Чебышева);

2. Least Squares FIR — расчет по методу наименьших квадратов (синтезируется фильтр с минимальной среднеквадратичной ошибкой аппроксимации АЧХ при заданной длине фильтра);
3. Kaizer Window FIR — расчет КИХ-фильтра методом взвешивания (методом окон) с использованием окна Кайзера (наиболее быстрый и простой метод расчета).

Первые два типа вида аппроксимации приводят к построению оптимальных передаточных функций. Расчет методом взвешивания прост, пригоден для фильтров практически любой сложности, но приводит к передаточным функциям излишне высокого порядка.

Передаточные функции БИХ-фильтров (на основе преобразования аналогового прототипа):

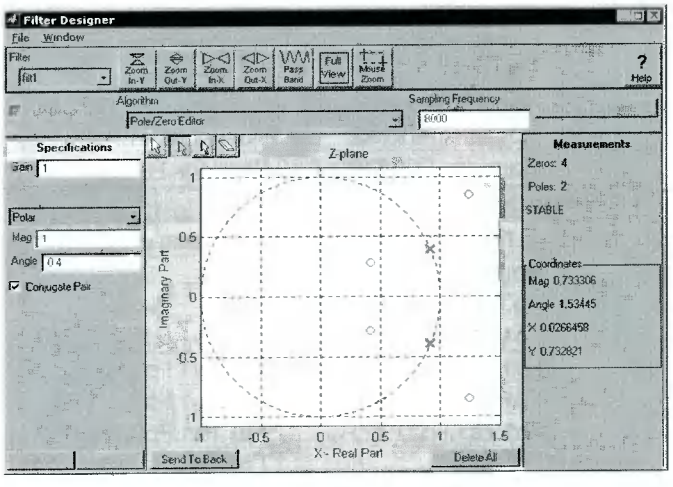
1. Butterworth IIR — передаточные функции фильтров Баттерворта (с монотонной АЧХ в полосах пропускания и задерживания);
2. Chebyshev Type I IIR — передаточные функции фильтров Чебышева, тип I (с равноволновой АЧХ в полосе пропускания и монотонной в полосах задерживания);
3. Chebyshev Type II IIR — передаточные функции фильтров Чебышева, тип II (фильтр с монотонной АЧХ в полосе пропускания и равноволновой в полосах задерживания);
4. Elliptic IIR — передаточные функции эллиптических фильтров (фильтры Золотарева-Кауэра — равноволновая АЧХ в полосах пропускания и задерживания).

Перечисленные типы передаточных функций оптимальны, но каждый в своем смысле [2,3].

После задания типа фильтра и выбора вида аппроксимации необходимо задать параметры расчета — требования к АЧХ и частоту дискретизации.

Процесс синтеза (этап аппроксимации) фильтра можно автоматизировать (кроме метода наименьших квадратов), поставив галочку напротив надписи Minimum Order. В этом случае Filter Designer будет автоматически определять передаточную функцию минимального порядка, для которой выполняются заданные требования, которые включают следующие параметры:

Рис. 8. Окно Filter Designer



- для фильтра высоких и нижних частот — граничные частоты полосы пропускания Fp (Гц) и задерживания Fs (Гц), максимально допустимое отклонение характеристики затухания от номинального уровня в полосе пропускания Rp (дБ), и минимально допустимое затухание в полосе задерживания Rs (дБ);
- для полосового или режекторного фильтра — граничные частоты полосы пропускания Fp1 и Fp2 (Гц), граничные частоты полосы задерживания Fs1 и Fs2 и параметры Rp (дБ) и Rs (дБ).

В случае, когда галочка напротив Mininum Order не стоит, каждый тип фильтра и вид аппроксимации имеют свой набор полей для ввода спецификаций. Набор параметров для каждого из этих случаев указан в *табл. 2*, а порядок передаточной функции определяется разработчиком и устанавливается в поле Order.

Аналогично на панели измерений (Measurements) в зависимости от состояния Minimum Order отображаются различные наборы характеристик (*табл.3*).

Заметим, что характеристики расчета грубо можно задавать и с помощью указателя мыши, двигая соответсвующие линейки-ограничители на графике АЧХ. А чтобы наблюдать остальные характеристики рассчитываемого фильтра, необходимо открыть его в окне Filter Viewer. Тогда кроме графика АЧХ возможно увидеть ФЧХ, карту нулей и полюсов, импульсную характеристику и т.д.

2.3. Эвристическое конструирование передаточной функции

Средства SPTool позволяют наблюдать, как изменения расположения нулей и полюсов влияют на АЧХ. Зная общие закономерности такого влияния, можно «вручную» конструировать передаточные функции с желаемыми АЧХ. Это особенно удобно, когда не требуется высокой точности воспроизведения характеристик, например при синтезе некоторых типов амплитудных корректоров. Для решения этой задачи используется режим Pole/Zero Editor компоненты Filter Designer.

Чтобы перейти в этот режим, нужно выбрать пункт 'Pole/Zero Editor' из выпадающего меню Algorithm. После этого окно Filter Designer будет выглядеть, как показано на *рис.8*. Вместо графика АЧХ отображается комплексная плоскость, на которой можно добавлять, перемещать или удалять нули и полюсы передаточной функции.

Окно спецификаций содержит поле для ввода коэффициента усиления фильтра (Gain), а также поля для ввода и отображения координат выбранного полюса или нуля в полярной либо в прямоугольной системе. Галочка напротив 'Conjugate pair' означает, что выбранный нуль или полюс имеет комплексно-сопряженную пару. Если выбран отдельный нуль или полюс, то при установке галочки к нему автоматически будет добавлен комплексно-сопряженный. Окно измерений отображает число нулей и полюсов, а также автоматически определяется стабильность фильтра.

Если в момент редактирования положения нулей и полюсов фильтр открыт в компоненте Filter Viewer, то можно отслеживать, как изменение положения

нулей и полюсов тут же отражается на какой-либо характеристике фильтра (например, на АЧХ).

2.4. Анализ характеристик полученного фильтра

Важным аспектом при расчете фильтров является подробный анализ характеристик фильтров во временной и частотной областях.

Компонента Filter Viewer позволяет интерактивно провести такой анализ. С помощью Filter Viewer можно графически просмотреть следующие характеристики фильтров (*рис. 4*): АЧХ, ФЧХ, ГВП, карту нулей и полюсов, ИХ и ПХ.

2.5. Задача реализации

Как уже говорилось, на втором этапе (этапе реализации) происходит декомпозиция передаточной функции, или построение алгоритма преобразования входного сигнала в выходной.

При экспорте рассчитанного фильтра из SPTool в общем случае в рабочее пространство MATLAB помещается структура, содержащая коэффициенты передаточной функции фильтра. Для определенности положим, что вектор значений числителя перенесен из структуры в переменную-вектор num, а знаменатель — в вектор den. Тогда для того, чтобы преобразовать передаточную функцию в одну из эквивалентных форм представления, необходимо воспользоваться соответствующими функциями Signal Processing Toolbox. Вот самые важные из них:

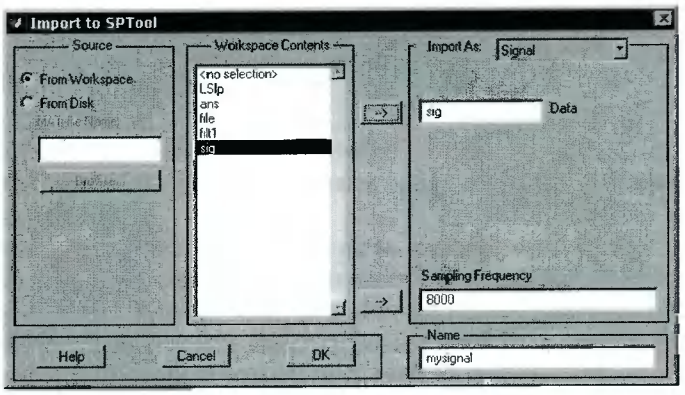
Передаточная функция → последовательное (каскадное) соединение звеньев второго порядка
[sos, g] = tf2sos(num, den)
Передаточная функция → пространство состояний
[A, B, C, D] = tf2ss(num, den)
Передаточная функция → нули и полюса
[z, p, k] = tf2zp(num, den)
Передаточная функция → решетчатый фильтр
[k, v] = tf2latc(num, den)
Пространство состояний → соединение звеньев второго порядка
[sos, g] = ss2sos(A, B, C, D)
Передаточная функция → параллельное соединение звеньев
[r, p, k] = residuez(num, den)

3. Исследование спектрального состава сигнала

Задача спектрального анализа — выяснение спектрального состава сигнала. Если реализация сигнала — детерминированная функция, то спектр оценивается с помощью преобразования Фурье. Для стационарных и квазистационарных случайных процессов оцениваются спектральные плотности мощности.

MATLAB предоставляет набор современных методов спектрального оценивания. Эти методы обеспечивают повышенную разрешающую способность и отсутствие ложных боковых лепестков при короткой выборке.

Рис. 9. Окно Import SPTool



SPTool позволяет интерактивно оценить спектральную мощность выбранного сигнала с помощью компоненты Spectrum Viewer (*рис. 5*).

Методы оценки [4], реализованные в Spectrum Viewer:

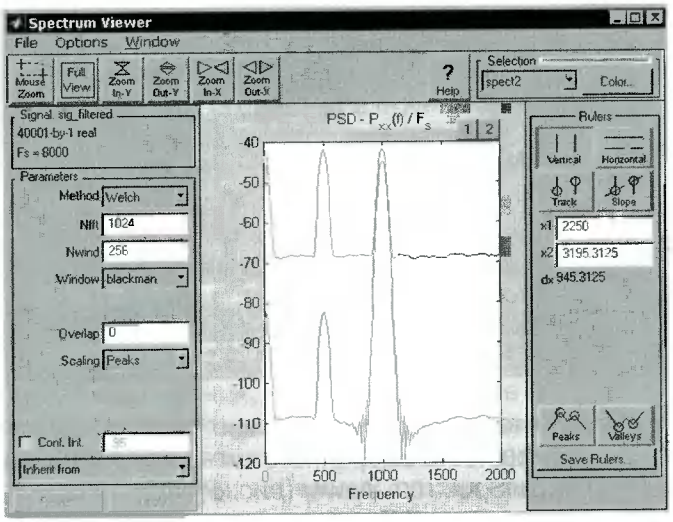
- Burg — метод Берга, один из самых первых и наиболее известных алгоритмов авторегрессионного спектрального оценивания, также называется алгоритмом максимальной энтропии;
- Covariance — ковариационный метод, основанный на АР-модели и минимизации ошибки предсказания вперед;
- FFT — асимптотически несмещенная оценка на основе быстрого преобразования Фурье — периодограмма;
- Modified covariance — модифицированный ковариационный метод — фактически полная аналогия ковариационного метода, но минимизируется ошибка предсказания как вперед, так и назад;
- MTM — Multitaper Method — для оценки спектра используются ортогональные окна (т.н. tapers);
- MUSIC — Multiple signal classification — алгоритм «классификации множественных сигналов»;
- Welch — метод Уэлша, основанный на временном усреднении коротких модифицированных периодограмм;
- Yule AR — метод Юла-Уолкера, основанный на модели авторегрессии-скользящего среднего (АРСС).

4. Пример

Рассматривается фильтрация сложного сигнала с последующей оценкой спектра и реализацией фильтра. План примера следующий: вначале генерируется сложный сигнал, далее проектируется фильтр (конструируется его передаточная функция), выполняется фильтрация сигнала, осуществляется оценка спектров исходного и фильтрованного сигналов. Передаточная функция фильтра экспортируется в рабочее пространство MATLAB, где осуществляется ее преобразование к виду, удобному для реализации фильтра с последовательным соединением звеньев второго порядка.

Сгенерируем в MATLAB сигнал, представляющий собой две синусоиды единичной амплитуды с частотой 500 и 1000 Гц, и добавим к этому сигналу шум. Для этого средствами MATLAB образуем исходный (входной) сигнал **sig**.

Рис. 10

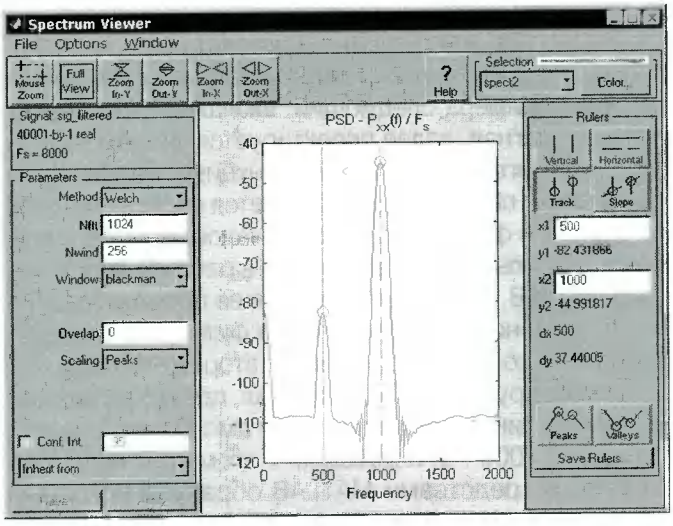



```
% time — вектор значений времени, длиной 5 с с интервалом 125 мкс (8 кГц)
time = ( 0: 1/8000: 5 );
% y500 — вектор отсчетов синусоиды с частотой 500 Гц
y500 = sin( 2 * pi * 500 * time );
% y1000 — вектор отсчетов синусоиды с частотой 1000 Гц
y1000 = sin( 2 * pi * 1000 * time );
% noise — вектор случайных значений, равномерно распределенных от 0 до 1 % (шум с постоянной составляющей)
noise = rand( 1, length( time ) );
% sig — результирующий сигнал
sig = y500 + y1000 + noise.
```

Запустим SPTool, набрав в командной строке MATLAB команду 'sptool'.

Далее нужно импортировать сгенерированный нами сигнал в SPTool. Для этого выберем в меню File пункт Import. Из появившегося списка переменных рабочего пространства MATLAB выберем переменную sig и поместим ее в поле Data, введем частоту дискретизации — 8000 Гц — в поле 'Sampling frequency' (*рис.9*). После нажатия кнопки 'OK' импортированный сигнал появится в контейнере Signals. Теперь, например, можно просмотреть или прослушать этот сигнал, выбрав его и нажав кнопку 'View' под окном Signals. Можно вычислить оценку плотности спектра мощности, нажав кнопку 'Create' под окном Spectra.

Рис. 11



Выберем в появившемся окне 'Spectrum Viewer' метод и параметры расчета спектра, например, как на *рис.5*, и нажмем кнопку 'Apply'. На графике полученного спектра отчетливо видны три спектральных пика — на частотах 500, 1000 и 0 Гц, обусловленные тем, что исследуемый сигнал состоит из двух синусоид (500 и 1000 Гц) и добавленного к ним шума, равномерно распределенного в интервале от 0 до 1 с постоянной составляющей, равной 0,5.

Попробуем уменьшить шумовую компоненту в сигнале и избавиться (заметно ослабить) синусоидальную компоненту на частоте 500 Гц. Для этого создадим полосовой фильтр и пропустим через него наш сигнал. Чтобы запустить компоненту Filter Designer, нужно нажать кнопку 'New Design' под контейнером Filters.

Выберем метод и параметры расчета такими, как на *рис.3*, и после нажатия кнопки 'Apply' фильтр будет рассчитан.

Отфильтруем сигнал. Для этого выберем и сигнал и фильтр (он по умолчанию будет называться filt1), нажмем кнопку 'Apply' под контейнером Filters. Будет предложено выбрать один из трех алгоритмов фильтрации (Direct Form II Transposed, Zero phase IIR, FFT based FIR). Первый из них — наиболее распространенный — связан с вычислением выходного сигнала по разностному уравнению вход-выход, которое непосредственно получается из передаточной функции. Второй удобен при анализе КИХ-фильтров с линейной фазой, устраняет задержку выходного сигнала по отношению ко входному. В результате применения этого алгоритма ФЧХ фильтра оказывается равной нулю, а АЧХ — квадрату АЧХ рассчитанного фильтра. Третий алгоритм использует принцип фильтрации в частотной области с применением быстрого преобразования Фурье (БПФ).

Выберем алгоритм по умолчанию — Direct Form II Transposed. Введем в поле Output signal имя фильтрованного сигнала — 'sig_filtered'. После нажатия кнопки 'OK' в контейнере Signals появится отфильтрованный сигнал sig_filtered. Этот сигнал можно просмотреть или прослушать, выбрав его и нажав кнопку View под окном Signals.

Рассчитаем оценку спектра сигнала на выходе фильтра по тому же методу и с теми же параметрами, что и для исходного (*рис.10*). Для наглядности эти оценки удобно наложить, как на *рис.11*. Видно, что теперь спектральные компоненты на частотах 0 и 500 Гц стали по уровню гораздо ниже, чем компонента на частоте 1000 Гц — это результат полосовой фильтрации. Вне полосы пропускания фильтра заметно уменьшился спектр шума.

Предположим, что нам необходимо реализовать рассчитанный полосовой фильтр. Для этого мы должны прежде всего получить его передаточную функцию.

Напрямую в SPTool этого сделать нельзя. Однако можно экспортировать фильтр в рабочее пространство MATLAB, выбрав из меню 'File' пункт 'Export'. В появившемся окне нужно выбрать фильтр для экспорта — в нашем случае он будет называться filt1 — и нажать кнопку 'Export to Workspace' (галочка напротив пункта 'Export Filters as TF objects' должна отсутствовать!!!)

После экспорта в рабочем пространстве MATLAB создается переменная-структура с соответствующим именем — filt1.

Эта структура будет содержать некоторое количество полей, но информацию о рассчитанном фильтре будут нести только поля filt1.tf и filt1.Fs (остальные поля используются внутри SPTool и могут меняться в разных версиях MATLAB).

Поле .tf, в свою очередь, тоже является структурой и в своих полях содержит описание передаточной функции фильтра:

- tf.num — вектор, содержащий коэффициенты числителя передаточной функции;
- tf.den — вектор, содержащий коэффициенты знаменателя передаточной функции (для КИХ-фильтра этот вектор будет отсутствовать или содержать один единичный элемент).

$$H(z)=\frac{num(1)+num(2)\cdot z^{-1}+num(3)\cdot z^{-2}+\ldots+num(n)\cdot z^{-(n-1)}}{den(1)+den(2)\cdot z^{-1}+den(3)\cdot z^{-2}+\ldots+den(m)\cdot z^{-(m-1)}},$$

где n, m — размерности векторов tf.num и tf.den соответственно.

В нашем примере значения коэффициентов передаточной функции рассчитанного фильтра следующие:

Числитель	Знаменатель
» filt1.tf.num'	» filt1.tf.den'
ans =	ans =
0,0099	1,0000
-0,0557	-5,6246
0,1567	15,8151
-0,2760	-27,7960
0,3314	33,2231
-0,2760	-27,4847
0,1567	15,4629
-0,0557	-5,4377
0,0099	0,9560

Реализуем фильтр в виде каскадного (последовательного) соединения звеньев второго порядка (2.4):

```
» [ sos, g ] = tf2sos( filt1.tf.num, filt1.tf.den )
sos =
1,0000   -1,2294   1,0000   1,0000   -1,3762   0,9820
1,0000   -1,5624   1,0000   1,0000   -1,4285   0,9826
1,0000   -1,3320   1,0000   1,0000   -1,3563   0,9952
1,0000   -1,4898   1,0000   1,0000   -1,4636   0,9955

  g =
   0.0099
```

Каждая строка матрицы sos содержит коэффициенты одного звена: первые три элемента строки — коэффициенты числителя, остальные три — знаменателя, т.е. соответствующая передаточная функция выглядит следующим образом:

$$H(z)=g\prod_{i=1}^{4}\frac{sos(i,1)+sos(i,2)\cdot z^{-1}+sos(i,3)\cdot z^{-2}}{1+sos(i,5)\cdot z^{-1}+sos(i,6)\cdot z^{-2}}.$$

***

Материал, изложенный в данной статье, свидетельствует, что SPTool является исключительно удобным и полезным средством решения разнообразных задач, связанных с двумя основными видами обработки сигналов: фильтрацией и спектральным анализом. Следует, однако, заметить, что успех применения мощных методов, заложенных в SPTool, все же зависит от квалификации исследователя, от того, насколько выбранные методы соответствуют содержанию решаемой задачи и целям проекта. Заметим также, что SPTool использует только часть методов фильтрации и спектрального анализа, содержащихся в MATLAB и, в частности, в Signal Processing Toolbox.

Несмотря на то что SPTool позволяет конструировать передаточные функции цифровых фильтров и использовать эффективные методы спектрального анализа, его применение для задач проектирования требует выхода за пределы возможностей GUI. Действительно, в какой-то мере это было показано в данной статье при обсуждении задачи реализации фильтров. Для разложения передаточной функции в формы, удобные для построения вычислительных алгоритмов, необходимо использовать общие средства MATLAB. Более того, для моделирования алгоритмов, например в арифметике с фиксированной точкой, следует применять Simulink, Realtime Workshop и другие соответствующие тулбоксы.

Авторы считают своим долгом выразить благодарность фирме SoftLine (http://www.softline.ru) за предоставленное программное обеспечение.

ЛИТЕРАТУРА

1. *Дьяконов В.П., Абраменкова И.В.* Matlab 5.0/5.3. Система символьной математики. — М.: Нолидж, 1999. — 634 с.
2. *Ланнэ А.А.* Оптимальный синтез линейных электронных схем. — М.: Связь, 1978. — 335 с.
3. *Рабинер Л., Гоулд Б.* Теория и применение цифровой обработки сигналов. — М.: Мир, 1978. — 848 с.
4. *Марпл-мл. С.Л.* Цифровой спектральный анализ и его приложения. — М.: Мир, 1990. — 584 с.

Е.И. Кренгель, К.А. Мешковский

m-подобные последовательности над GF(2^m) и их применение в широкополосных системах связи

На основе сдвинутых копий двоичных m-последовательностей длины 2^M-1, где M=mk, m≥2, k≥2, строится новый класс последовательностей над GF(2^m) длины 2^M-1. Доказывается, что этот класс обладает в точности такими же статистическими свойствами, что и класс m-последовательностей над GF(2^m) длины 2^M-1. Рассматриваются различные аспекты применения этих последовательностей в широкополосных системах связи и их схемной реализации.

Среди применяющихся в широкополосных системах связи псевдослучайных последовательностях (ПСП) наибольшую известность и распространение получили двоичные последовательности максимальной длины, или m-последовательности [1]. На их популярность среди разработчиков не отразился даже заметно выросший в последние годы интерес к нелинейным последовательностям, обладающим высокой структурной скрытностью. Такая необыкновенная живучесть m-последовательностей в основном обусловлена следующими факторами:

1) хорошими статистическими свойствами;
2) сравнительно большим объемом ансамбля, особенно при больших значностях;
3) линейностью и в силу этого простотой аппаратной реализацией;
4) m-последовательности служат основой для формирования других многочисленных ансамблей линейных и нелинейных ПСП (последовательностей Голда, Касами, бент-последовательностей, а также последовательностей Гордона, Милза, Велча, известных за рубежом как последовательности GMW [2]).

На сегодняшний день существуют достаточно хорошо развитая теория и практика как двоичных, так и q-ичных m-последовательностей. Бесспорно, двоичные m-последовательности являются наиболее простыми с точки зрения генерации, так как в отличие от q-ичных используют обычную булевую логику. Тем не менее в достаточно большом числе приложений, связанных с кодированием, модуляцией по закону прыгающей частоты и другого, применяются q-ичные m-последовательности. Известны исследования, в которых генерация q-ичных m-последовательностей сводится к генерации двоичных [3,4]. Из них наибольший интерес представляет работа [4], касающаяся выявления связей между m-последовательностями над GF(q^m) и GF(q). Согласно [4] любой элемент m-последовательности над GF(q^m) длины q^{nm}-1 может быть представлен в виде линейной комбинации базисных элементов GF(q^m) над GF(q) с коэффициентами из элементов m-последовательностей над GF(q). Недостатком полученных в [4] результатов является то, что, доказывая существование такого представления, они не содержат явного указания на то, как следует выбирать эти m-последовательности над GF(q). Вместо этого в [4] сначала строится m-последовательность над GF(q^m), а уже потом по ней находят, какими должны быть m-последовательности над GF(q). В настоящей работе предпринята попытка в какой-то мере обойти эту трудность за счет построения нового класса q-ичных последовательностей длины 2^M-1, где $q=2^m$, M=mk, m≥2, k≥2, обладающих статистическими свойствами класса m-последовательностей над GF(2^m) и формирующихся на основе сдвинутых копий двоичных m-последовательностей той же длины.

1. Описание конструкции нового класса

Пусть L есть линейный функционал из GF(2^M) в GF(2) такой, что L(1)=1. Соответственно пусть L_0 есть сужение L до подполя GF(2^m), а L_2 есть линейный функционал из GF(2^M) в GF(2^m) такой, что для $\forall x \in GF(2^M)$ справедливо $L_0(L_2(x)y)=L(xy)$ для $\forall y \in GF(2^m)$. Тогда согласно [5] можно показать, что последовательность $\{b_n\}$ с элементами вида

$$b_n = L_2(\alpha^n), \qquad (1)$$

где α – примитивный элемент GF(2^M); $0 \le n < 2^M-1$, – m-последовательность над GF(2^m) длины 2^M-1. Последовательность $\{c_n\}$ с элементами

$$c_n = L(\alpha^n), \qquad (2)$$

есть двоичная m-последовательность длины 2^M-1.

Пусть $\beta=a^{\varepsilon}$, где $\varepsilon=2^M-1/2^m-1$. Тогда β – примитивный элемент поля GF(2^m) порядка $w=2^m-1$. Пусть $\{d_n\}$ последовательность над GF(2^m) с элементами вида

$$d_n=L(a^n)+\beta L(a^{n+\varepsilon})+\beta^2 L(a^{n+2\varepsilon})+...+\beta^{m-1}L(a^{n+\varepsilon(m-1)}). \quad (3)$$

Покажем, что последовательность {dn} имеет период 2^M-1 и не совпадает с классом m-последовательностей GF(2^m). Первое утверждение тривиально и следует из того, что период последовательностей $\{L(a^n)\},\ldots,\{L(a^{n+(m-1)\varepsilon})\}$ есть 2^M-1. Для доказательства второго утверждения представим последовательности $\{b_n\}$, $\{c_n\}$, $\{d_n\}$ в виде двумерных таблиц T_b, T_c, T_d размерности εxw, в которых каждый n-й элемент последовательности стоит на пересечении i строки и j столбца, где $0 \le i < \varepsilon$, $0 \le j < w$, $n=i+j\varepsilon$. Со-

гласно [5] таблица T_b содержит ε-2^{M-m} нулевых строк, т.е. строк из w $0\in GF(2^m)$, тогда как остальные строки являются циклическими сдвигами строки с номером i=0 вида 1, β, β^2,...,β^{w-1}. Таблица T_c также содержит ε-2^{M-m} нулевых строк, однако уже из GF(2). Соответственно остальные являются циклическими сдвигами m-последовательности над GF(2) длины w вида $\{L(a^{j\varepsilon})\}$, $0\le j<w$. Далее замечаем, что в силу построения последовательности $\{d_n\}$ нулевые строки таблицы T_d оказываются расположенными на тех же местах , что и в таблицах T_b и T_c. Кроме того, в силу свойства «окна» m-последовательности любые ненулевые строки таблицы T_d будут состоять из w различных ненулевых элементов $GF(2^m)$ и являться сдвигами друг друга. Предположим, что для некоторого i-го элемента последовательности $\{d_n\}$ $d_i=1$. Тогда $L(a^i)=1$, $L(a^{i+\varepsilon})=L(a^{i+2\varepsilon})=...L(a^{i+\varepsilon(m-1)})=0$. Рассмотрим строку таблицы T_d, содержащую элемент d_i. В соответствие с (3) следующим элементом в этой строке будет

$$d_{i+\varepsilon}=L(a^{i+\varepsilon})+\beta L(a^{i+2\varepsilon})+\beta^2 L(a^{i+3\varepsilon})+\ldots+\beta^{m-1}L(a^{i+\varepsilon m})=\beta^{m-1}L(a^{i+\varepsilon m}).$$

Данная строка не содержит нулевых элементов, поэтому $L(a^{i+\varepsilon m})=1$ и $d_{i+\varepsilon}=\beta^{m-1}$. Таким образом установлено, что в ненулевой строке T_d за элементом 1 всегда следует элемент β^{m-1}. Отсюда следует, что последовательности $\{b_n\}$ и $\{d_n\}$ суть различные последовательности, не являющиеся сдвигами друг друга. Рассуждая аналогично, можно показать, что последовательность $\{d_n\}$ не совпадает также ни с одной другой q-ичной m-последовательностью той же длины. Данное утверждение справедливо применительно к любой другой последовательности $\{d_n'\}$,элементы которой определяются выражением

$$d_n'=L(\gamma^n)+\lambda L(\gamma^{n+\varepsilon})+\lambda^2 L(\gamma^{n+2\varepsilon})+\ldots+\lambda^{m-1}L(\gamma^{n+\varepsilon(m-1)}),$$

где $\gamma=a^t$, t – целое,взаимно простое с 2^M-1, $\lambda=a^{t\varepsilon}=\beta^t$.

Число различных последовательностей $\{d_n\}$ совпадает с числом m-последовательностей степени M над GF(2) , равным $\varphi(2^M-1)/M$, что меньше общего числа m-последовательностей степени M над $GF(2^m)$ в k раз.

2. Статистические свойства последовательностей $\{d_n\}$

В [1] показано, что псевдослучайный характер последовательностей целиком определяется ее статистическими свойствами. При исследовании статистических свойств последовательностей $\{d_n\}$ будем опираться на известные статистические свойства m-последовательностей $\{b_n\}$ в том виде, как они даны в [2].

Свойство 1 (балансное). Число N_b появления ненулевого символа b на периоде m-последовательности $\{b_n\}$ на 1 превышает число N_0 появления символа **0** на этом периоде, т.е. $N_b=N_0+1$.

Свойство 2. Число N_a позиций, внутри периода на которых встречается J-строка $\mathbf{a}=a_1a_2...a_J$, определяется выражением

$$N_a=\begin{cases}2^{M-mJ}, \text{ для } \mathbf{a}\neq 0,\ 1\le J\le k\\ 2^{M-mJ}, \text{ для } \mathbf{a}=0,\ 1\le J\le k\\ 0 \text{ или } 1,\ k<J.\end{cases}$$

Свойство 3 (аддитивно-циклическое). Разность между m-последовательностью $\{b_n\}$ и ее τ-сдвигом $\{b_{n+\tau}\}$ есть другой ν-сдвиг $\{b_{n+\nu}\}$ той же самой m-последовательности. При этом выполняется $b_{n+\nu}=b_{n+\tau}-b_n$ для всех n.

Свойство 4. Пусть $\{i_n\}$ есть последовательность целых чисел таких, что

$$i_n=\sum_{J=0}^{J-1} b_{n+j}2^{mj},$$

где $(b_n, b_{n+1},..., b_{n+J-1})$ – J-строка m-последовательности $\{b_n\}$,а $1\le J\le k$.

Пусть $\mathbf{i_k}$ есть последовательность длины 2^{M-1}, образованная из элементов $\{i_n\}$, начиная с k-го. Тогда для $\tau\neq 0 \bmod(2^M-1)$ расстояние по Хэммингу между $\mathbf{i_1}$ и $\mathbf{i_{1+\tau}}$ вычисляется по формуле

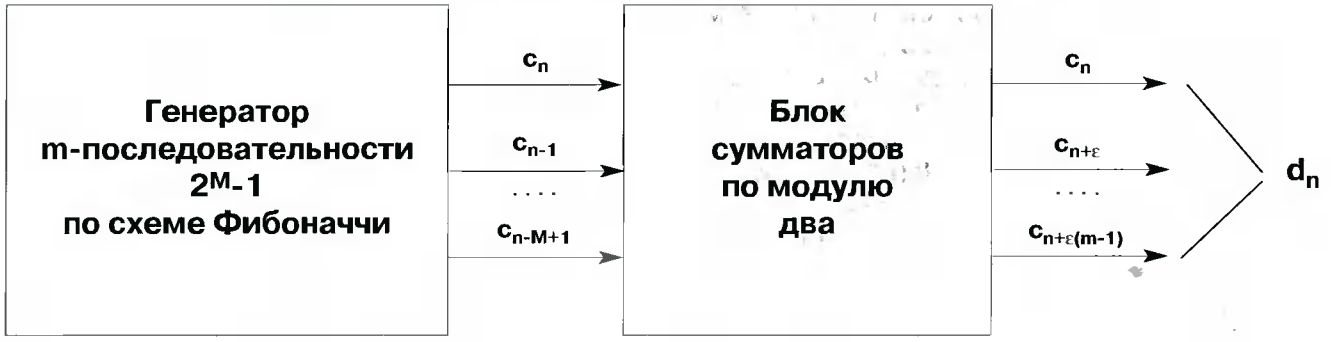
$$H(\mathbf{i_1},\mathbf{i_{1+\tau}})=2^M(1-2^{-mJ}).$$

Здесь $H(x,y)=\sum_{i=1}^{N} h(x_i,y_i)$ – метрика Хэмминга, где $h(x_i,y_i)$ есть 0, если $x_i=y_i$ и 1 в противном случае.

Покажем, что свойства 1 – 4 имеют также построенные выше последовательности $\{d_n\}$. Действительно, выполнение свойств 1 – 2 для $\{d_n\}$ следует из взаимной однозначности элементов строк таблиц T_d и T_b. Справедливость свойства 3 следует из справедливости этого свойства для компонент $L(a^n)\},...,\{L(a^{n+(m-1)\varepsilon})$ элемента d_n . Доказательство свойства 4 для последовательности $\{d_n\}$ вытекает из наличия у ней свойств 2 и 3. Заметим, что этим одновременно доказывается и псевдослучайность последовательности $\{d_n\}$.

Таким образом, обладая всеми свойствами m-последовательностей, последовательности $\{d_n\}$, по существу, являются m-подобными последовательностями меньшей мощности. Поэтому в дальнейшем последовательности $\{d_n\}$ будем называть m-подобными. Вообще говоря, существуют и другие классы m-подобных последовательностей, получаемых, например, перестановкой компонент в символах $\{d_n\}$ или $\{b_n\}$. Очевидно, такие последовательности являются производными последовательностей $\{d_n\}$ и $\{b_n\}$ и обладают свойствами 1 – 4.

Рис. 1. Генератор m-подобной последовательности над GF(2^m)



3. Применение m-подобных последовательностей в широкополосной связи

В соответствие с (3) генератор m-подобных последовательностей может быть реализован в виде устройства, формирующего m сдвинутых друг относительно друга на ε разрядов копий двоичной m-последовательности. Способы формирования сдвинутых копий достаточно подробно исследованы во многих работах, в частности в [6]. Большинство этих способов основываются на аддитивно-циклическом свойстве m-последовательности и могут быть реализованы различным образом. На *рис.1* представлена блок-схема генератора m-подобных последовательностей над GF(2^m), выполненного на основе последовательно соединенных генератора двоичной m-последовательности по схеме Фибоначчи и блока сумматоров по модулю два. Можно указать по меньшей мере две возможные области применения m-подобных последовательностей. Это широкополосные системы многостанционного доступа с модуляцией прямыми последовательностями (DSMA) и широкополосные системы многостанционного доступа с модуляцией прыгающей частотой (FHMA). В настоящее время среди используемых в широкополосных системах связи ПСП большой интерес вызывают нелинейные двоичные ПСП GMW, обладающие такими же идеальными периодическими автокорреляционными функциями, что и m-последовательности, но имеющие при этом во много раз большую линейную сложность. В [2] подробно исследованы некоторые классы ПСП GMW и предложен способ их генерации на основе 2^m-ичных m-последовательностей длины 2^{M-1}. Однако построение 2^m-ичного генератора не является простой задачей, так как помимо знания полинома с коэффициентами из GF(2^m) требует выполнения операций сложения и умножения над GF(2^m). В [7] предложен новый, более простой метод генерации ПСП GMW на основе формирования m-подобных последовательностей. Данный метод теоретически позволяет получать все существующие ПСП GMW независимо от того, на какие бы два сомножителя m и k не раскладывалось число M. В соответствие с [7] элементы последовательности GMW могут быть представлены в следующем виде

$$gmw_n=g(L(a^n)+\beta L(a^{n+\varepsilon})+\beta^2 L(a^{n+2\varepsilon})+...+\beta^{m-1}L(a^{n+\varepsilon(m-1)})) \quad (4),$$

где g: GF(2^m) → GF(2) есть функционал, определяемый условиями

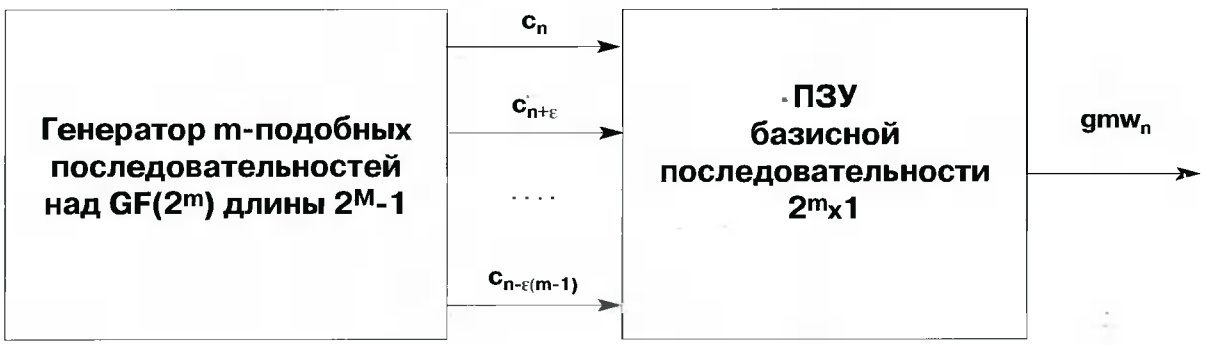
$$g(L(\alpha^{\varepsilon j})+L(\alpha^{\varepsilon(j+1)})\beta+...+L(\alpha^{\varepsilon(j+m-1)})\beta^{m-1})=C_j,$$

$$0\le j<2^m-1 \text{ и } g(0)=0\,, \quad (5)$$

а $\{C_j\}$, $0\le j<2^m-1$ – некоторая базисная последовательность с двухуровневой функцией автокорреляции, образованная на основе разностного множества с параметрами

$$\upsilon^*=2^m-1,\ \kappa^*=2^{m-1},\ \lambda^*=2^{m-2}\,.$$

Рис.2. Генератор последовательности GMW



Блок-схема генератора последовательности GMW по методу [7] представлена на *рис.2*. Кроме того, генератор m-подобных последовательностей может быть использован для получения других семейств нелинейных последовательностей, строящихся на основе m-последовательностей над $GF(2^m)$ и разностных множеств [8].

m-подобные последовательности могут найти применение и в системах с кодовым разделением и модуляцией с прыгающей частотой, закон изменения которой определяется m-последовательностью над $GF(2^m)$ [2]. В этом случае генераторы m-последовательностей также могут быть безболезненно заменены более простыми генераторами m-подобных последовательностей. Однако, как справедливо отмечено в [2], недостатком этих последовательностей является их малая линейная сложность, численно равная k. Поэтому, когда требуется большая линейная сложность, предлагается использовать другие, более сложные в реализации последовательности, например обобщенные бент-функции и бент-последовательности [9]. Этой же цели могут служить сконструированные нами на основе m-подобных последовательностей 2m-ичные последовательности, описание которых приводится ниже.

Рассмотрим последовательность $\{z_n\}$ длины 2^M-1 с элементами вида

$$z_n=[L_2(a^n)]^t, \qquad (6)$$

где $1<t<w$ – целое число и $(t, 2^M-1)=1$.

Пусть T_z – двумерная (ε,w)-таблица последовательности $\{z_n\}$. Тогда в силу того, что отображение $\beta^j\rightarrow\beta^{jt}$, где $0\le j<w$, есть изоморфизм поля Галуа $GF(2^m)$, заключаем, что между элементами соответствующих строк таблиц T_b и T_z имеется взаимно однозначное соответствие. С учетом этого нетрудно убедиться, что для последовательностей $\{z_n\}$ справедливы статистические свойства 1, 2, 4 и не справедливо 3. В соответствие с [2] находим, что линейная сложность L_z последовательности $\{z_n\}$ определяется следующим выражением:

$$L_z=k^u, \qquad (7)$$

где u – число единиц в двоичном представлении числа t.

При этом максимальное значение $L_z=k^{m-1}$ достигается при $t=(2^{m-1}-1)2^s \bmod (2^M-1)$, где $0\le s<m$. Аналогично предыдущему в выражении (6) заменим m-последовательность $L_2(a^n)$ соответствующей m-подобной последовательностью $\{d_n\}$. В результате генератор последовательности $\{z_n\}$ будет состоять из последовательно соединенных генератора m-подобных последовательностей и ПЗУ объемом 2^mxm, по соответствующим адресам которого хранятся двоичные представления всех элементов поля $GF(2^m)$. В целях упрощения генерации последовательностей с максимальной линейной сложностью выберем t=-1 и рассмотрим отображение L3: $GF(2^m)\rightarrow GF(2^m)$, удовлетворяющее следующим условиям

$$(\beta^j)^{-1}\rightarrow L(a^{\varepsilon(2^m-2-j)})+\beta^{-1}L(a^{\varepsilon(2^m-3-j)})+\ldots+\beta^{-(m-1)}L(a^{\varepsilon(2^m-m-1-j)})$$

$$\text{и } 0\rightarrow 0 \quad (8),$$

где $0\le j<2^m-1$.

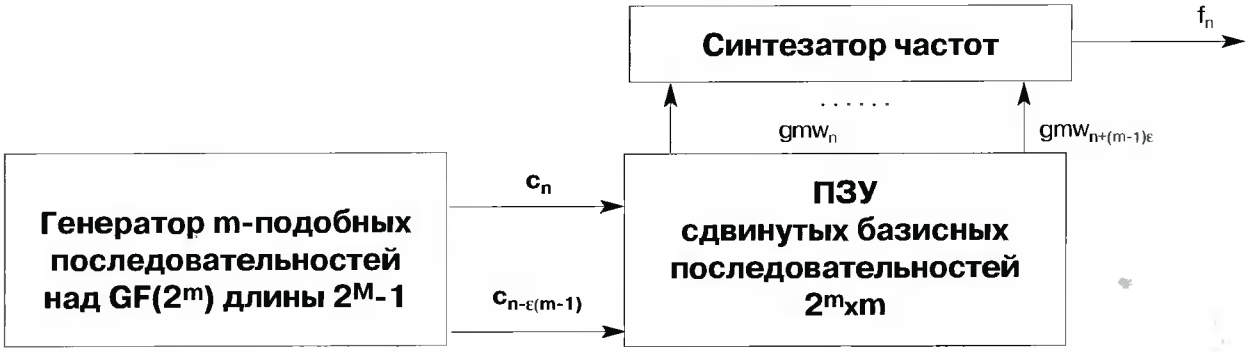
Это отображение взаимно однозначное, так как $\{L(a^{\varepsilon(2^m-2-j)})\}$, где $0\le j<2^m-1$, есть m-последовательность длины 2^m-1 и, следовательно, любой ненулевой набор из m последовательных ее символов встречается на ее периоде ровно один раз, причем число различных таких наборов равно 2^m-1. Если затем в ПЗУ по тем же самым адресам вместо элементов $(\beta^j)^{-1}$ разместить соответствующие им согласно отображению (8) элементы, то в результате на выходе такого генератора образуется последовательность с элементами $z_n'=L_3([L_2(a^n)]^{-1})$, которая имеет такие же статистические свойства и линейную сложность, что и последовательность $\{[L_2(a^n)]^{-1}\}$. Нетрудно заметить, что элементы последовательности $\{z_n'\}$ обладают следующим замечательным свойством. Согласно (4), (5) двоичные координаты ее элементов в рассмотренном базисе $GF(2^m)$ представляют собой сдвинутые на ε разрядов копии последовательности GMW с базисной m-последовательностью $\{L(a^{-\varepsilon j})\}$, т.е. $z_n'=(gmw_n,gmw_{n+\varepsilon},\ldots,gmw_{n+\varepsilon(m-1)})$. Основное преимущество данного генератора перед генератором последовательности $\{[L_2(a^n)]^{-1}\}$ состоит в том, что для построения его ПЗУ необходимо лишь знать обратную к $\{L(a^{\varepsilon j})\}$, где $0\le j<2^m-1$, последовательность $\{L(a^{-\varepsilon j})\}$, для получения которой достаточно использовать арифметику над GF(2).

Для построения псевдослучайного генератора для FHMA поступим следующим образом. Пусть $\mathbf{z^n'}$ J-строка последовательных элементов из $\{z_n'\}$. Кроме того, пусть $s(\mathbf{x})$ произвольное взаимно однозначное отображение $\mathbf{z_n'}$ в множество 2^{Jm} различных частот. Рассмотрим последовательность $f_n=s(\mathbf{z_n'})$. Тогда для всех $\tau\ne0$ будет выполняться равенство $H(f_n,f_{n+\tau})=2^M-2^{M-Jm}$. Это следует из свойства (4) и взаимооднозначности отображения s. На *рис.3* изображена блок-схема FHMA псевдослучайного генератора на основе $\mathbf{z_n'}$ при J=1. В качестве примера рассмотрим генерацию 2^3-ичной последовательности $\{L_3([L_2(a^n)]^{-1})\}$ длины 4095 на основе генератора m-подобной последовательности с параметрами M=12, m=3 и J=1. Пусть примитивный элемент α поля $GF(2^{12})$ является корнем примитивного полинома

$$X^{12}+X^{11}+X^8+X^6+1. \qquad (9)$$

Тогда в соответствие с [7] генератор m-подобной последовательности должен формировать m=3 сдвинутых последовательностей $\{c_n\}$, $\{c_{n+585}\}$, $\{c_{n+1170}\}$, где $c_n=L(\alpha^n)$. Очевидно, по-

Рис.3. FHMA псевдослучайный генератор на основе z_n' при J=1



следовательности c_{n+585} и c_{n+1170} являются задержанными соответственно на 3510 и 2925 чипов копиями m-последовательности c_n. Формирование задержанных копий m-последовательности наиболее просто осуществить суммированием по модулю два выходов соответствующих разрядов регистра сдвига генератора m-последовательности по схеме Фибоначчи. Пусть $\mathbf{c_n, c_{n-1}, \ldots, c_{n-(M-1)}}$ последовательности, образующиеся на выходах разрядов регистра сдвига этого генератора. Тогда для $c_{n+585}=c_{n-3510}$ и $c_{n+1170}=c_{n-2925}$ с помощью компьютера находим, что

$$c_{n+585}=c_n+c_{n-1}+c_{n-2}+c_{n-3}+c_{n-5}+c_{n-7}+c_{n-10} \text{ и}$$

$$c_{n+1170}=c_{n-1}+c_{n-2}+c_{n-8}+c_{n-9}+c_{n-10}\,.$$

Образуем 7-элементную ненулевую последовательность $\{e_j\}$, где $e_j=L(a^{585j})$. Это всегда можно сделать путем соответствующего выбора начального состояния генератора последовательности $\mathbf{c_n}$. При L(1)=1 эта последовательность имеет вид 1101001. Тогда обратная к ней последовательность есть 1001011. Для построения ПЗУ образуем *табл. 1*, в которой 3-разрядным наборам следующим подряд символов последовательности $\{e_j\}$, рассматриваемым согласно (3) как двоичные адреса, ставятся в соответствие 3-разрядные элементы $L_3(\beta^{-j})$. Далее учитывая, что по нулевому адресу всегда находится символ 0, после упорядочения таблицы по возрастанию адресного параметра получаем *табл. 2*, полностью определяющую структуру ПЗУ генератора, функциональная схема которого приведена на *рис.4*. Формируемая последовательность имеет следующие параметры: L=16 и H=3584. Однако задержав z_n' всего на один разряд и переходя к случаю J=2, можно существенно улучшить параметр H до 4032. При этом число различных частот составит 64. Заметим, что такой же результат можно получить посредством генератора с параметрами M=12, m=6, k=2 и J=1.

К сожалению, генератор на основе последовательности $\{z_n'\}$ позволяет получать только сдвинутые копии последовательности $\{f_n\}$. В то же время описанный в [2] псевдослучайный генератор FH на основе 2^m-ичной m-последовательности обеспечивает генерацию целого семейства последовательностей с элементами $f_n^{\nu}=s(\mathbf{x}+\nu)$, где ν – произвольная J-строка, с хорошими Хэмминговыми расстояниями. Но их линейная сложность мала. Альтернативой этому являются FH-последовательности, формируемые на основе последовательности (6) с помощью преобразования $s(\mathbf{x}+\nu)$. При этом их Хэмминговые расстояния будут такими же, а линейная сложность много больше.

Таблица 1

АДРЕС			$L_3(\beta^{-j})$
1	1	0	100
1	0	1	001
0	1	0	010
1	0	0	101
0	0	1	011
0	1	1	111
1	1	1	110

Таблица 2

АДРЕС			СИМВОЛ GF(2^3)
0	0	0	000
0	0	1	011
0	1	0	010
0	1	1	111
1	0	0	101
1	0	1	001
1	1	0	100
1	1	1	110

***

Получен новый класс m-подобных последовательностей над GF(2^m) длины 2^M-1, обладающий в точности такими же статистическими свойствами, что класс m-последовательностей над GF(2^m) той же длины, но строящийся более простым способом на основе m равномерно сдвинутых друг

относительно друга на $\varepsilon = 2^M-1/2^m-1$ разрядов копий двоичной m-последовательности 2^M-1.

Рассмотрены некоторые варианты применение этих последовательностей в широкополосных системах многостанционного доступа с модуляцией прямыми последовательностями и широкополосных системах многостанционного доступа с модуляцией прыгающей частотой. Показано, что m-подобные последовательности успешно могут применяться как при генерации двоичных последовательностей GMW, так и при генерации 2^m-ичных последовательностей с большей линейной сложностью.

ЛИТЕРАТУРА

1. *Цифровые методы в космической связи./* Под ред. Голомба С. М.: Связь, 1969.

2. *Simon M.K., Omura J.K., Scholtz R.A., Levit B.K.* Spread Spectrum communications Handbook. – McGraw-Hill,Inc., 1994.

3. *Krone S.M., Sarwate D.V.* Quadriphase sequences for Spread-Spectrum Multiple-Access Communication // IEEE Trans. Inform. Theory. – Vol. 30. – № 5. – 1984.

4. *Park W.J., Komo J.J.*. Relationships between m-sequences over GF(q) and GF(q^m). // IEEE Trans. Inform. Theory. – Vol. 35. – №1. – 1989.

5. *Baumert L.D.* Cyclic difference sets. – Berlin-Heidelberg: Springer-Verlag. 1971.

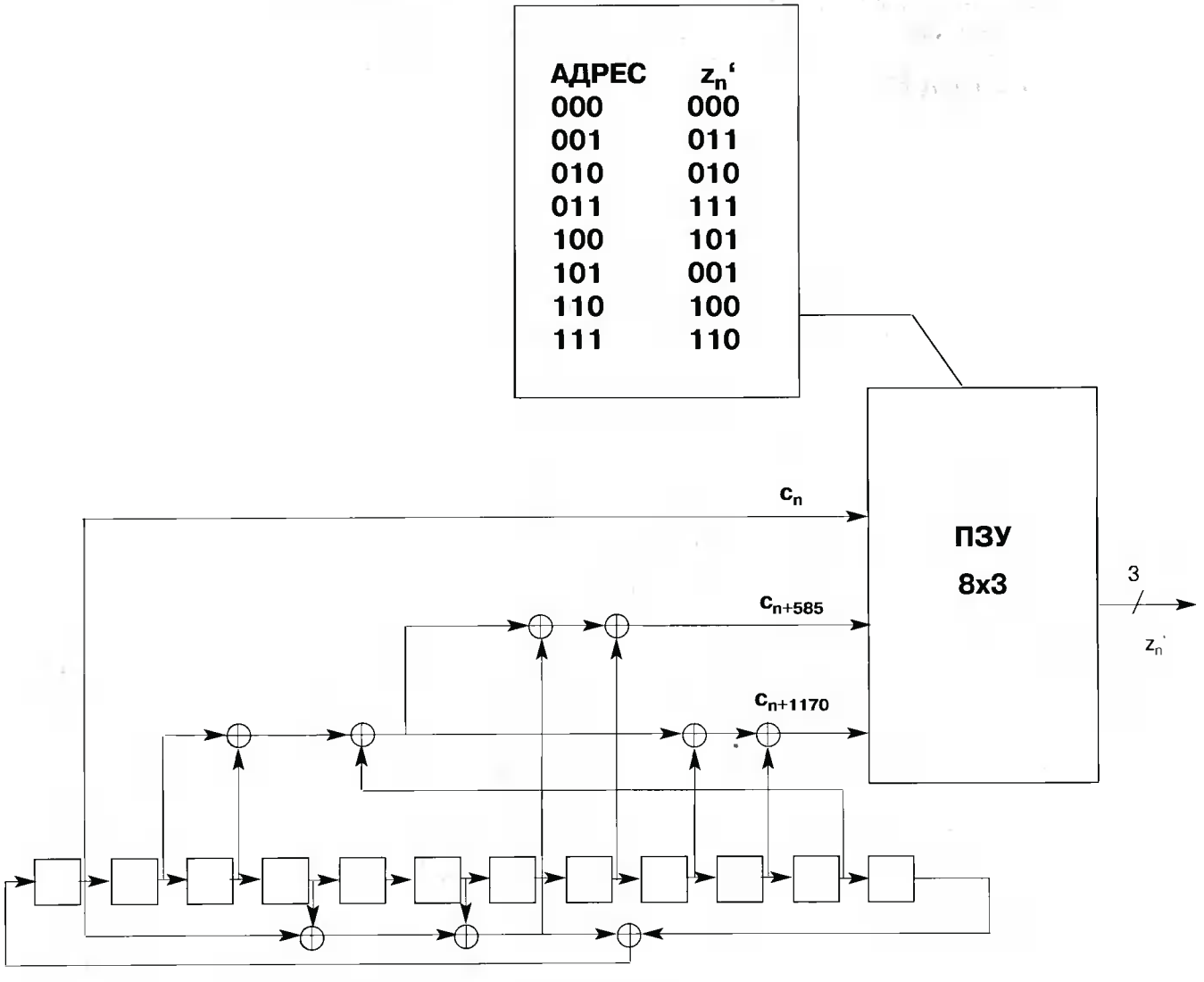
6. *Бессарабова А.П., Журавлев В.И.* Псевдослучайные последовательности сигналов и их применение в технике связи. М.: ВИНИТИ, 1991, №7.

7. *Мешковский К.А. Кренгель Е.И.* Генерация псевдослучайных последовательностей Гордона, Милза, Велча. // Радиотехника, 1998, №5.

8. *Ипатов В.П.* Периодические дискретные сигналы с оптимальными корреляционными свойствами. М.: Радио и связь, 1992.

9. *Kumar P.V.* Frequency-hopping code sequence designs having large linear span. // Trans. Inform. Theory. – Vol.34. – №1. – 1988.

Рис.4. Генератор последовательности $z_n'=(gmw_n, gmw_{n+585}, gmw_{n+1170})$ над $GF(2^3)$ длины 4095 с параметрами M=12, m=3 и ε=585



С.А. Грибачев, В.Ф. Козаченко Ю.В. Гончаров

Цифровые сигнальные процессоры. Концепция трех платформ компании Texas Instruments. Платформа 'C2000

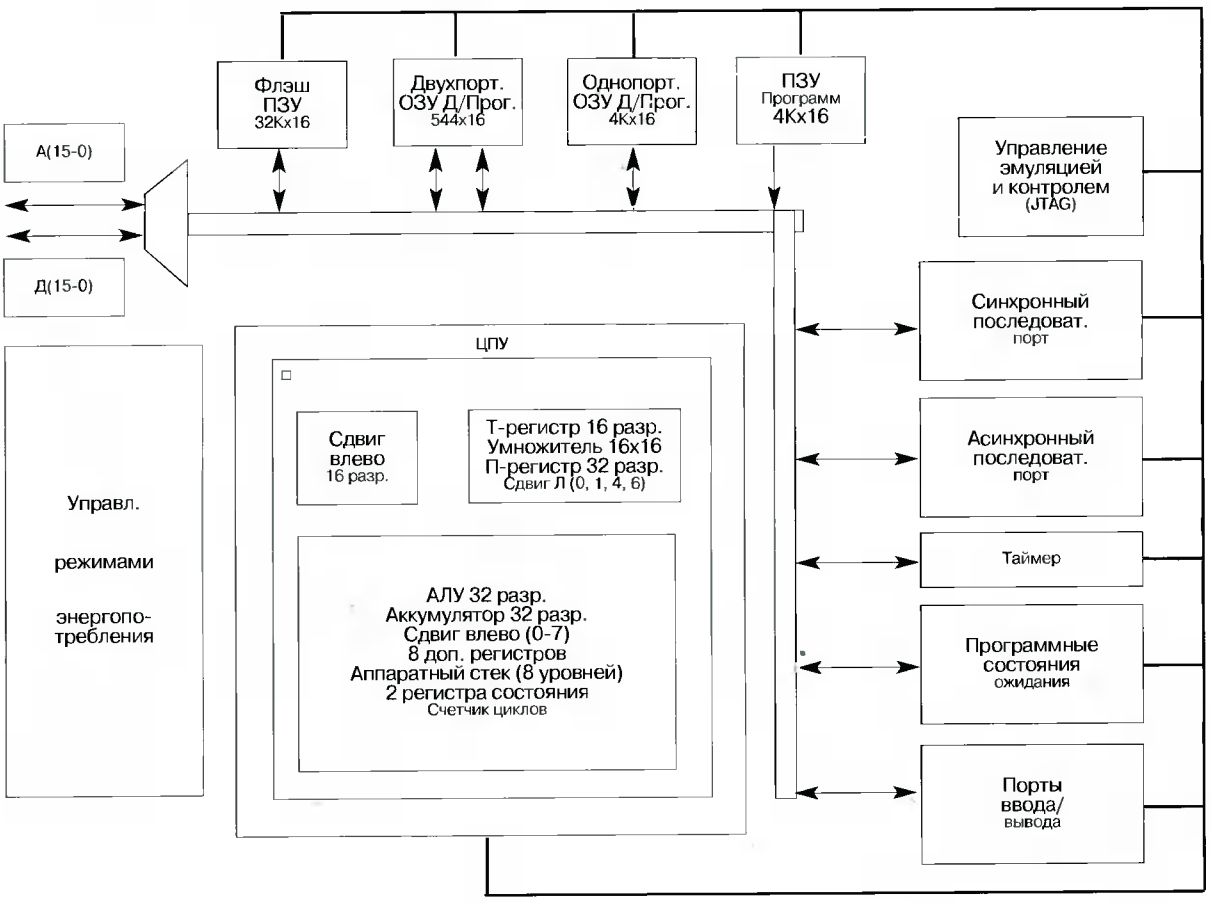
В первых двух статьях данной серии, посвященной цифровым сигнальным процессорам компании Texas Instruments [1, 2], были рассмотрены платформы 'C6000 и 'C5000. ЦСП первой платформы ориентированы на достижение предельного быстродействия, областью эффективного применения второй являются портативные высокопроизводительные системы. Платформа TMS320C2000 создавалась для рынка встроенных систем цифрового управления. При ее разработке ставилась задача существенно потеснить доминирующие на этом рынке универсальные микроконтроллеры. Семейства ЦСП платформы 'C2000 будут рассмотрены в данной статье.

Платформа 'C2000 состоит из двух семейств ЦСП: семейства универсальных ЦСП для микроконтроллерных приложений TMS320C20x и семейства ЦСП для цифрового управления электродвигателями TMS320C24x.

Семейство TMS320C20x исторически появилось на свет первым. Будучи полностью программно совместимыми с популярным семейством TMS320C25, данные ЦСП были реализованы на новой технологической базе, что позволило существенно увеличить производительность (до 40 MIPS), снизить потребляемую мощность (до 1,1 mA/MIPS), увеличить объем встроенной памяти (до 32K слов). ЦСП TMS320F206 стал первым на мировом рынке сигнальным процессором со встроенной FLASH-памятью. На *рис. 1* приведена структурная схема ЦСП TMS320C20x.

Процессоры данного семейства имеют модифицированную Гарвардскую архитектуру с независимым доступом к памяти команд и памяти данных. Ядро ЦСП включает 32-разрядное АЛУ, 32-разрядный аккумулятор, однотактный умножитель 16x16, набор сдвигателей и регистры состояния и управления. 8 дополнительных 16-разрядных адресных регистров дают возможность организовать эффективную адреса-

Рис. 1



цию памяти. Шинная архитектура ЦСП позволяет выполнять одновременную выборку команд и операндов. В сочетании с 4-уровневым конвейером это повышает эффективность работы процессора, поскольку сразу несколько команд могут выполняться одновременно. Наличие встроенного двухпортового ОЗУ команд/данных позволяет производить одновременную выборку двух операндов в одном такте либо выборку операнда/команды и запись результата. Наличие встроенного **FLASH-ПЗУ емкостью 32K** слов дает возможность не только хранить всю программу во внутренней памяти ЦСП, но и при необходимости легко модифицировать программный код. Некоторые модели данного семейства имеют записанный в ПЗУ начальный загрузчик, который позволяет загружать программу из внешнего 8-разрядного ПЗУ при включении питания.

Включенный в процессор набор периферийных устройств облегчает ЦСП общение с «внешним миром». Так, **программируемый генератор задержек** служит для увеличения длительности цикла обмена по шине до семи машинных тактов при обмене с медленными внешними устройствами. Он работает без использования дополнительного внешнего оборудования. Количество тактов задержки задается через регистры конфигурации.

Расширенный **синхронный последовательный порт** с максимальной частотой синхронизации, равной половине тактовой частоты процессора, позволяет реализовать обмен со скоростями до 20 Мбит/с. Он может использоваться для подключения ЦАП, АЦП, кодеков и других периферийных устройств с последовательным интерфейсом, а также для межпроцессорных обменов в многопроцессорных системах. Как в приемном, так и в передающем каналах порта имеется память FIFO глубиной четыре слова. Порт имеет гибкую программируемую структуру генерации тактовых импульсов и синхронизации. Возможна работа как с 8-, так и с 16-разрядными данными, подключение устройств по протоколу SPI, многоканальный режим для прямого подключения к большинству типов кодеков и плат речевого интерфейса.

Асинхронный последовательный порт работает в дуплексном режиме с двойной буферизацией и поддерживает скорости передачи до 2,5 Мбит/с. Порт работает с 8-разрядными данными и может использоваться для реализации интерфейса RS-232. Имеет средства определения входной скорости обмена данными; 16-разрядный программируемый делитель тактовой частоты позволяет задавать стандартные скорости обмена без необходимости специального подбора частоты тактового генератора.

Важным свойством семейства ЦСП 'C20x является наличие встроенного интерфейса для отладки и диагностики по стандарту **JTAG (IEEE 1191.1)**. Возможность использования внутрисхемного эмулятора XDS-510 в комплекте со средой отладки Code Composer существенно повышает эффективность отладки, сокращает сроки и позволяет оптимизировать программный код.

В *табл. 1* приведены основные характеристики ЦСП семейства TMS320C20x. Как уже было отмечено, данные ЦСП предназначены для рынка микроконтроллерных приложений. Обладая относительно невысокой стоимостью (от 6 долларов для TMS320C203), это семейство благодаря высокой производительности ядра ЦСП позволяет реализовать качественно новый набор функций в устройствах, обычно создаваемых на основе микроконтроллеров. Например, при реализации системы охраны объекта один ЦСП способен не только контролировать сигналы пассивных датчиков, но может наделить систему интеллектуальными способностями. Так, анализируя спектр сигналов микрофонного датчика, ЦСП в состоянии отличить шум проезжающего автомобиля или низко летящего самолета от звука разбиваемого окна либо скрежета стеклореза по стеклу. В результате существенно сокращается число ложных срабатываний системы. Кроме того, при обнаружении факта вторжения скрытая видеокамера может передать сжатую цифровую «картинку» злоумыш-

Таблица 1

Устройство	U пит., В	Частота, МГц	Скорость, MIPS	Длительность такта, нс	ОЗУ, слов	ПЗУ, слов	ПДП	Посл. порт	Нач. загр
TMS320C203	5	40	20	50	544		Внеш.	2	Да
TMS320C203-57	5	57	28.5	35	544		– II –	2	– II –
TMS320C203-80	5	80	40	25	544		– II –	2	– II –
TMS320LC203	3.3	40	20	50	544			2	– II –
TMS320F206	5	40	20	50	4,5 K	32K Flash	Внеш.	2	Нет
TMS320C206	5	80	40	25	4,5 K	32K mask	– II –	2	Да
TMS320LC206	3.3	80	40	25	4,5 K	32K mask	– II –	2	Да
TMS320C209	5	57	28.5	35	4,5 K	4K mask			Нет

ленника по модему на внешний пульт охраны. И все эти функции (плюс ряд дополнительных) могут быть реализованы одним ЦСП семейства 'C20x.

По неофициальной информации от представителей Texas Instruments не планируется дальнейшего развития данного семейства. Тем не менее выпуск рассмотренных ЦСП активно продолжается, и на их основе производится большое количество модемов, устройств сжатия речи, цифровых АТС, медицинских приборов и другого оборудования.

Семейство **ЦСП TMS320C24x** разрабатывалось для решения задач цифрового управления различным оборудованием (в т.ч. электродвигателями). Данное семейство появилось на рынке чуть позже семейства 'C20x, однако именно оно развивается фирмой наиболее активно во многом благодаря огромной емкости рынка, для которого данные ЦСП предназначены. Сегодня это семейство включает серийно выпускаемые ЦСП TMS320C24x и доступные в виде образцов ЦСП TMS320C240x. Процессорное ядро данных семейств аналогично ядру 'C20x, но периферия значительно расширена. Важным отличием семейств 'C24x является наличие встроенных модулей АЦП и блока менеджера событий.

TMS320C24x. В состав каждого ЦСП данного семейства входят по два 10-разрядных АЦП со встроенным устройством выборки/хранения. Минимальное время преобразования составляет 0,85 мкс; 16 аналоговых входов подаются на входы АЦП через два 8-входовых мультиплексора. Каждый из двух АЦП работает независимо и может запускаться как от внешнего сигнала, так и программно. Менеджер событий представляет собой оптимизированный блок управления, включающий три таймера и до девяти компараторов. Позволяет организовать до 12 ШИМ-выходов. Поддерживает широкий спектр режимов ШИМ. **Сторожевой таймер** служит для контроля работы аппаратуры программного обеспечения и генерирует сигнал сброса, если в течение заданного времени к нему не было обращения со стороны процессора. **Модуль прерываний реального времени** служит для периодической генерации прерываний через установленный интервал с программируемой частотой от 1 до 4096 прерываний в секунду.

Процессоры 'F240 имеют 16 Кслов встроенной флэш-памяти программ, полный набор периферийных устройств, адаптированных для целей управления двигателями, последовательный коммуникационный интерфейс SCI, последовательный периферийный интерфейс SPI, а также интерфейс с внешней памятью и внешними периферийными устройствами EMIF. Наличие последнего интерфейса позволяет проектировать сложные системы управления, обеспечивающие дружественный интерфейс с оператором и дополнительным тех-

Таблица 2

Характеристика	F240	C240	F241	C241	C242	F243
'C2xx DSP-ядро	Да	Да	Да	Да	Да	Да
Цикл выполнения команды, нс	50	50	50	50	50	50
Производительность (при 20 МГц), MIPS	20	20	20	20	20	20
ОЗУ двойного доступа DARAM (16-разр.)	544	544	544	544	544	544
Секторная флэш-память (16-разр.)	16 K	–	8 K	–	–	8 K
Встроенное ПЗУ (16-разрядных слов)	–	16 K	–	8 K	4 K	–
Интерфейс внешней памяти EMIF	Есть	Есть	–	–	–	Есть
Менеджер событий:	Есть	Есть	Есть	Есть	Есть	Есть
◆ число таймеров общего назначения	3	3	2	2	2	2
◆ число каналов сравнения/ШИМ	9/12	9/12	5/8	5/8	5/8	5/8
◆ число каналов захвата/ввода «квадратурных» сигналов	4/2	4/2	3/2	3/2	3/2	3/2
Сторожевой таймер	Есть	Есть	Есть	Есть	Есть	Есть
АЦП (10-разрядный):	Есть	Есть	Есть	Есть	Есть	Есть
◆ число каналов	16	16	8	8	8	8
◆ время преобразования (минимальное)	6,6 мкс	6,6 мкс	850 нс	850 нс	850 нс	850 нс
Последовательный периферийный интерфейс SPI	Есть	Есть	Есть	Есть	–	Есть
Последовательный коммуникац. интерфейс SCI	Есть	Есть	Есть	Есть	Есть	Есть
CAN-интерфейс локальной сети	–	–	Есть	Есть	–	Есть
Число линий дискретного ввода/вывода	28	28	26	26	26	32
Число внешних прерываний	6	6	6	6	6	6
Напряжение питания, В	5	5	5	5	5	5
Тип корпуса	132 PQFP	132 PQFP	68 PLCC, 64 PQFP	68 PLCC, 64 PQFP	68 PLCC, 64 PQFP	144 TQFP

Таблица 3

Характеристика	'LF2407	'LF2406	'LF2402	'LC2406	'LC2404	'LC2402
'C2xx DSP-ядро	Да	Да	Да	Да	Да	Да
Цикл выполнения команды, нс	33	33	33	33	33	33
Производительность (при 30 МГц), MIPS	30	30	30	30	30	30
ОЗУ (16-разрядных слов) DARAM	544	544	544	544	544	544
SARAM	2 K	2 K	–	2 K	1 K	–
Секторная флэш-память (16-разр. слов)	32 K	32 K	8 K	–	–	–
Встроенное ПЗУ (16-разрядных слов)	–	–	–	32 K	16 K	4 K
Загрузочное ПЗУ (16-разрядных слов)	256	256	256	–	–	–
Интерфейс внешней памяти	Есть	–	–	–	–	–
Менеджеры событий A и B (EVA, EVB):	EVA, EVB	EVA, EVB	EVA	EVA, EVB	EVA, EVB	EVA
◆ число таймеров общего назначения	4	4	2	4	4	2
◆ число каналов сравнения/ШИМ	10/16	10/16	5/8	10/16	10/16	5/8
◆ число каналов захвата/ ввода «квадратурных» сигналов	6/4	6/4	3/2	6/4	6/4	3/2
Сторожевой таймер	Есть	Есть	Есть	Есть	Есть	Есть
АЦП (10-разрядный):	Есть	Есть	Есть	Есть	Есть	Есть
◆ число каналов	16	16	8	16	16	8
◆ время преобразования (минимальное), нс	500	500	500	500	500	500
Последоват периферийный интерфейс SPI	Есть	Есть	–	Есть	Есть	–
Последоват. коммуникац. интерфейс SCI	Есть	Есть	Есть	Есть	Есть	Есть
CAN-интерфейс локальной сети	Есть	Есть	–	Есть	–	–
Число линий дискретного ввода/вывода	41	41	21	41	41	21
Число внешних прерываний	5	5	3	5	5	3
Напряжение питания, В	3.3	3.3	3.3	3.3	3.3	3.3
Тип корпуса	144 TQFP	100 TQFP	64 TQFP	100 TQFP	100 TQFP	64 TQFP

нологическим оборудованием (возможность подключения пультов оперативного управления любой сложности, карт ввода/вывода и т.д.).

Процессоры '241 и '243 имеют встроенный контроллер CAN-интерфейса, что позволяет объединять несколько систем управления в локальную промышленную сеть, которая представляет собой последовательную двухпроводную шину с произвольным числом ведущих контроллеров (контроллеров-мастеров), каждый из которых может обмениваться данными с ведомыми контроллерами (узлами сети). CAN-протокол реализован в '241, '243 аппаратно, что существенно уменьшает затраты времени на разработку драйверов обмена и ускоряет процесс проектирования распределенных систем управления реального времени.

В *табл. 2* приведены основные технические характеристики семейства ЦСП TMS320C24x.

Применение ЦСП семейств '241, '243 позволяет простыми средствами создавать распределенные системы управления оборудованием на базе локальных промышленных сетей, а также строить системы автоматизации производства в станкостроении, робототехнике, автомобилестроении. При этом можно решать стоящие задачи с помощью распределенных систем позиционного и контурного управления на базе микроконтроллеров с CAN-интерфейсами. Преимущество такого подхода состоит в удешевлении системы управления и снятии ограничений на число одновременно работающих осей привода.

В конце 1999 г. было объявлено о планах дальнейшего развития семейства ЦСП TMS320C24x. Новое **семейство TMS320C240x** реализовано с использованием следующего уровня развития технологии и имеет существенно лучшие характеристики, приведенные в *табл. 3*. Важной особенностью является использование для всех ЦСП семейства напряжения питания 3,3 В, что позволило снизить энергопотребление. При этом удалось повысить производительность версий со встроенной флэш-памятью с 20 до 30 MIPS.

Первые опытные образцы изделий уже появились в России, а с середины 2000 г. ожидается начало их массового производства. Сигнальные процессоры имеют различный объем встроенной памяти и различный набор периферийных устройств, что дает возможность разрабатывать системы управления, оптимальные по критерию производительность/цена/функциональные возможности.

Ядро ЦСП семейства '240x также имеет модифицированную многошинную Гарвардскую архитектуру с раздельным доступом к памяти

программ и памяти данных, аналогичную архитектуре семейства 'C20x. Команды выполняются параллельно на 4-уровневом конвейере. Поддерживается обмен данными между различными областями памяти, размещение коэффициентов цифровых регуляторов и цифровых фильтров как в памяти программ, так и в памяти данных. Система команд оптимизирована для эффективного решения задач управления реального времени и цифровой фильтрации, в частности легко реализуются кольцевые буфера на основе использования специального метода адресации операндов с реверсивным распространением бита переноса.

Процессоры семейства '240x имеют гибкую 32-уровневую систему прерываний, являющуюся расширением системы прерываний семейства '24x. Запросы прерываний поступают как от встроенных периферийных устройств, так и от внешних источников. Каждый из двух менеджеров событий вырабатывает свои собственные запросы прерываний.

Процессор 'LF2407 предназначен для использования в самых сложных системах привода, включая двухдвигательные и двухинверторные привода транспортных средств, где на принципиально новом уровне могут быть решены вопросы синхронизации вращения колес, разворота на месте, защиты от юза и другие специфические вопросы, присущие современным тяговым приводам.

Сдвоенный менеджер событий предоставляет в распоряжение пользователя уникальные возможности по управлению инверторами сразу двух преобразователей частоты. Причем имеется возможность одновременного ввода в «квадратурном» режиме сигналов положения сразу с двух импульсных датчиков. Используя таймеры, программист может программно идентифицировать скорость каждой оси и по разности скоростей организовывать эффективное управление, например демпфирование упругих колебаний в механической части привода, безударную выборку зазоров и т.п. Указанные выше возможности несомненно оценят разработчики механизмов поворота и передвижения кранов, экскаваторов, приводов радиотелескопов и других сложных систем, где успешная борьба с упругими связями и зазорами в передачах является решающим фактором успеха проекта.

Наличие значительного числа каналов формирования ШИМ-сигналов (16) открывает новые перспективы перед разработчиками систем управления приводами, способными рекуперировать энергию торможения в сеть – приводами подъемных механизмов кранов, лифтов, шахтных механизмов и т.д. В этом случае микроконтроллер может одновременно в реальном времени управлять входным и выходным преобразователями, обеспечивая эффективную работу привода во всех четырех квадрантах.

АЦП может работать в режиме автоматического сканирования, когда выполняется последовательный запуск аналого-цифрового преобразования по заранее заданным пользователем каналам, т.е. реализуется так называемая «измерительная сессия». Причем в отличие от ЦСП серии '24x, где результаты преобразования размещались в двухуровневом стеке, в новых ЦСП '240x для каждого из 16 каналов предусмотрены свои индивидуальные регистры результата. В течение одной сессии можно выполнить до 16 преобразований. Уникальные возможности АЦП позволяют ставить и эффективно решать сложные задачи прямого цифрового управления током и моментом в современных приводах переменного тока. Прежде всего речь идет о возможности реализации нескольких «псевдоодновременных» выборок данных, а также о возможности получения множественных выборок по одним и тем же каналам в целях либо увеличения разрешения, либо эффективной цифровой фильтрации.

ЦСП 'LC2404, 'LF2402 и 'LC2402 оптимизированы по критерию максимум производительности и функциональных возможностей за минимум цены и предназначены для использования в составе приводов изделий массового спроса, в первую очередь для бытовой техники: пылесосов, холодильников, стиральных машин. Учитывая их невысокую стоимость, особенно версий с масочным ПЗУ (около 4 долларов), можно предсказать большую популярность данных ЦСП уже в ближайшем будущем.

20 марта 2000 г. на Интернет-страничке Texas Instruments (www.ti.com) появилось сообщение о планах выпуска нового семейства ЦСП TMS320C28x, программно-совместимого с 'C24x. В сочетании с объявленными в феврале семействами TMS320C64x и TMS320C55x данные ЦСП показывают перспективы развития фирмы в направлении достижения трудно-представимых сегодня характеристик цифровых сигнальных процессоров. Но это уже тема для отдельного разговора.

ЛИТЕРАТУРА

1. *Козаченко В.Ф., Грибачев С.А.* Новые микроконтроллеры фирмы Texas Instruments TMS32x24x для высокопроизводительных встроенных систем управления электроприводами // CHIP NEWS. 1998, N11 – 12, с. 2 – 6.
2. *Козаченко В.Ф., Грибачев С.А.* Перспективная серия микроконтроллеров фирмы Texas Instruments '240x для систем цифрового управления двигателями // CHIP NEWS. 1999, N9, с. 7 – 14.

ИНФОРМАЦИЯ

Новости рынка цифровых сигнальных процессоров (по материалам Internet)

В феврале – марте 2000 г. на web-страничке компании Texas Instruments (TI) появились сообщения о новых семействах цифровых сигнальных процессоров (ЦСП), планируемых к выпуску в текущем году. Многими эта информация была воспринята с энтузиазмом. Огорчает лишь тот факт, что такие технологические новинки появляются порой быстрее, чем удается освоить хотя бы малосерийное производство на базе уже известных ЦСП. Тем не менее новая продукция TI должна вызвать заслуженный интерес, поскольку объявленные характеристики трех новых семейств кажутся сегодня поистине фантастическими.

Семейство TMS320C28x
Дальнейшая оптимизация под решение задач цифрового управления

- **400 MIPS**
- **MAC 32x32 за такт**
- Адр. простр. до **8 Gbyte**
- Отклик на прерывание : **20 – 40 нс**
- Чтение/Модификация/ /Запись за **1 такт**
- Повышенная **эффективность кода**
- **Совместимость** с семейством 'C24x
- Стоимость от $2
- Возможность отладки программ на имеющихся средствах 'C24x
- **Code Composer с RTDX**
- Порт для трассировки

Семейство TMS320C55x
Оптимизация для портативных приложений и Телекома

- **800 MIPS**
- **600 – 800 MMACs**
- **0.05 mW/MIPS**
- **Uпит = 0.9 В**
- Усовершенствованная подсистема энергосбережения
- Регистр режимов холостого хода
- $^1/_2$ количества тактов 'C54x для реализации тех же функций
- Переменная длина команды **(8 – 48 разрядов)**
- **Доп. 16-разр. АЛУ** + регистры данных
- Сокращение кода **на 30%**
- **eXpressDSP**
- Совместимость с семейством 'C54x

Семейство TMS320C64x
Ориентация на предельное быстродействие

- **9000 MIPS**
- **До 4400 16-разр MMACs До 8800 8-разр MMACs**
- **Uпит = 1 В**
- **Спец. команды ветвления**
- Сокращение кода **на 25%**
- Переменная длина слова данных
- Доп. команды специального назначения
- Расширенный 2-уровневый кэш
- 2 16-разр. операции/4 8-разр. операции за такт
- **eXpressDSP**
- Совместимость с семейством 'C62x

Благодаря программной совместимости с существующими семействами ЦСП уже в ближайшее время разработчики имеют возможность начать отладку программ для новых процессоров. Это позволило бы существенно сократить сроки появления готовых изделий, поскольку к моменту появления образцов ЦСП можно иметь уже частично отлаженный программный код. А пока будем ждать подробностей от Texas Instruments и других производителей.

Грибачев С.А.

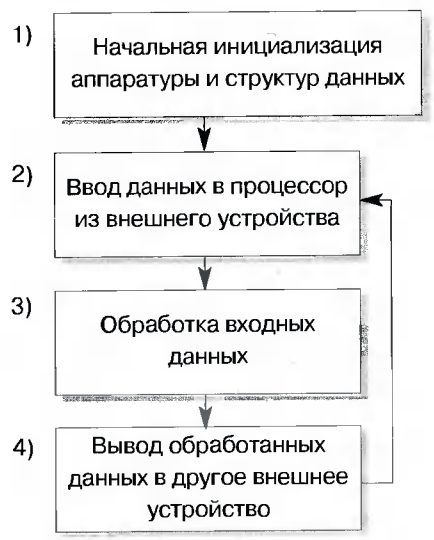
Д.В. Эккоре

Оптимизация программного обеспечения для TMS320C6201

Когда разработчики говорят об оптимизации прикладного программного обеспечения (ПО), то, как правило, имеют в виду, во-первых, сокращение времени выполнения критических участков программы и, во-вторых, экономию аппаратных ресурсов (чаще всего приходится экономить внутрикристальную память). Но прежде чем начать обсуждение того, как выполнять подобную оптимизацию на примере процессора TMS320C6201, хотелось бы обсудить применение языков высокого уровня при разработке оптимизированных алгоритмов ЦОС.

Традиционно в алгоритмах ЦОС в качестве языка высокого уровня используется язык Си. В настоящее время все производители ЦПОС поставляют компилятор языка Си, включающий в свой состав оптимизатор. Например, оптимизатор процессора 'C6201 настолько эффективно реализует некоторые вычисления, что вручную на ассемблере не удается получить более быстрый код. Конечно, большая часть вычислений реализуется компилятором Си не столь эффективно, но это вовсе не означает, что всю программу надо писать на ассемблере, а язык Си применять не надо. Его обязательно надо применять, и вот почему. Когда мы начали обсуждать оптимизацию, то забыли, что оптимизировать надо не только производительность программы, но и рабочее время программиста. И в этом случае Си незаменим. Например, если ваша программа требует начальной инициализации аппаратуры, то с помощью языка Си это сделать гораздо проще, а оптимизации никакой не требуется. Значит, на Си нужно писать те части программы, которые не требуют оптимизации или требуют незначительной оптимизации, и только наиболее критичные ко времени участки кода писать с использованием ассемблера. В этом случае программа Си будет работать так же быстро, как и программа, целиком написанная на ассемблере.

Рис. 1. Обобщенная блок-схема алгоритма ЦОС



Рассмотрим, как реализовать подобный подход на следующем примере.

Обобщенный алгоритм ЦОС

На *рис. 1* приведена обобщенная блок-схема алгоритма ЦОС. Практически любая программа может быть после упрощения приведена к подобному виду. Текст программы, соответствующей обобщенной блок-схеме, показан на *рис.2*. Очевидно, что критичная ко времени часть кода – это цикл, включающий функции InputBuf(), ProcessBuf() и OutputBuf(), поскольку именно здесь выполняется работа с сигналом в реальном масштабе времени. А функцию Init() можно спокойно писать на Си, тем более что очень часто инициализация и подготовка процессора к работе являются довольно сложной задачей, требующей поддержания множества режимов и подрежимов, и язык Си здесь будет очень кстати.

Рис. 2. Обобщенная программа ЦОС

```
main()
{
    int     *inBuf, *outBuf;
    int     size=512;

    Init();  // Соответствует блоку 1 (рис. 1)
    for(;;)
    {
       InputBuf(&inBuf, size);           // Блок 2
       ProcessBuf(inBuf, outBuf, size); // Блок 3
       OutputBuf(&outBuf, size);         // Блок 4
    }
}
```

Рассмотрим функции InputBuf() и OutputBuf(). Конечно, ввод/вывод данных необходимо выполнять максимально быстро, чтобы, во-первых, успевать вводить быстрые потоки данных и, во-вторых, минимизировать использование ЦПУ процессора. Но для этого совсем не обязательно использовать ассемблер.

Большинство ЦПОС содержат внутрикристальные контроллеры ПДП, которые аппаратно (а значит, быстро), не используя ресурсов ЦПУ, могут закачивать входные данные во внутреннюю память или выкачивать их из памяти во внешнее устройство. Надо только выделить буфер во внутренней памяти, запрограммировать контроллер ПДП и запустить

его на ввод или вывод. Функции InputBuf() останется только забрать заполненный буфер и передать контроллеру ПДП другой пустой буфер. Фактически эта функция оперирует с указателями на буфера, меняя их местами. Значит, она выполняется быстро и ее можно спокойно писать на Си, тем более что если размер буфера достаточно большой (например, size=512), то действие этой функции «размажется» на 512 отсчетов. Все сказанное полностью относится и к функции OutputBuf().

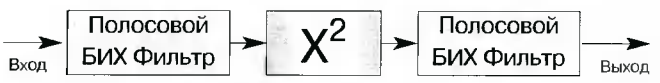
К сожалению, некоторые разработчики ПО привыкли работать не с буферами, а с отдельными отсчетами. Их программы работают так: сначала они вводят один отсчет, потом его обрабатывают и по мере появления выходных данных выводят их на внешнее устройство. К сожалению, программу, составленную подобным образом, невозможно оптимизировать для процессора 'C6201. Для таких программ лучше использовать другие процессоры, например TMS320C54x, потому что вполне может оказаться, что, несмотря на более высокую тактовую частоту, 'C6201 будет работать медленнее.

Сказанное является следствием одного важного факта: **процессор 'C6201 очень быстро обрабатывает буфера данных и довольно медленно работает с отдельными отсчетами**. Ниже в этой статье при рассмотрении методов оптимизации можно найти объяснение этого факта. Но если алгоритм должен работать с каждым отдельным отсчетом, то спокойно можете считать, что дальнейшее изложение вряд ли будет полезно.

Вернемся к программе на *рис.2*. У нас осталась функция ProcessBuf(), которая выполняет обработку входного буфера и формирование выходного буфера. Большую часть времени процессор будет выполнять код этой функции, поэтому именно она требует оптимизации. Конечно, ее тоже можно попытаться написать на Си. Но если сложность обработки велика, то компилятор Си даже с включенным оптимизатором, скорее всего, не сможет сгенерировать быстрый код, и функцию ProcessBuf() придется писать на ассемблере. Точнее говоря, на ассемблере надо писать не всю функцию ProcessBuf(), а отдельные ее элементы. Приведем в качестве примера функцию, выполняющую удвоение частоты (*рис. 3*). Она состоит из трех последовательных блоков: 1-й блок выполняет полосовую фильтрацию, 2-й возводит все отсчеты буфера в квадрат, 3-й снова выполняет полосовую фильтрацию.

Конечно, эту функцию можно побробовать целиком написать на ассемблере. Но, скоре всего, внутренних регистров процессора может оказаться недостаточно, и поэтому код получится недостаточно быстрым. Кроме того, большие алгоритмы довольно сложно оптимизировать, особенно вручную.

Рис. 3. Удвоитель частоты



Разумнее разбить этот алгоритм на три функции, каждую из которых писать на ассемблере. В этом случае функция ProcessBuf() будет выглядеть следующим образом:

Рис. 4. Примерный код удвоителя частоты

```
SIir2   sIir[2]; // Две глобальные структуры

void ProcessBuf( short inBuf[], short outBuf[], in size )
{
    Iir2( inBuf,  tmpBuf, size, &sIir2[0] );
    Sqr(  tmpBuf, tmpBuf, size, 16 );
    Iir2( tmpBuf, outBuf, size, &sIir2[1] );
};
```

Функции Iir2() и Sqr() рассматриваются ниже.

Как разрабатывать функции на ассемблере

Каждый компилятор Си определяет собственные правила разработки функции на ассемблере. Поэтому, прежде чем взяться за это нелегко дело, необходимо познакомиться с соответствующей документацией. Для процессора 'C6201 необходимую информацию следует искать в главе 8 (Runtime Environment) руководства TMS320C6000 Optimizing C Compiler User's Guide (SPRU187). Там приведены соглашения об использовании регистров и описана структура функции, которая может вызываться из программы Си.

В принципе любая функция, написанная на ассемблере, должна выполнить примерно одни и те же действия. Сначала она должна выполнить подготовительные операции, во время которых выделяются и инициализируются необходимые регистры и, если необходимо, запрещаются прерывания. Затем выполняется основной цикл, во время которого циклически обрабатывается входной буфер данных и формируется выходной буфер. И наконец, выполняются завершающие операции, которые восстанавливают все необходимые регистры, разрешают прерывания и возвращают управление вызывающей функции.

Примерный код функции Fnx() показан на *рис. 5*. Сделаем к нему несколько замечаний:

1. Функция на ассемблере должна называться _Fnx, с начальным символом подчеркивания, иначе программа Си не сможет к ней обратиться. Ее имя обязательно надо сделать глобальным с помощью директивы .global.

2. С помощью регистра B3 в функцию передается точка возврата из функции.

3. Все аргументы передаются в функцию через регистры (если, конечно, вы не хотите усложнить себе жизнь и не разрабатываете функцию с переменным числом аргументов). Первый аргумент передается в регистре A4, второй – в B4, третий – в A6, затем – B6, A8, B8 и т. д. Удобнее всего передавать в функцию только указатели, а все параметры объединить в одну структуру и передать указатель на нее.

4. В регистр AMR при возвращении из функции необходимо записать нуль, а регистры A10 – A15,

B10-B15 и CSR должны восстановить свои значения. Для их временного хранения в сегменте .bss выделяется область stack.

5. Прерывания внутри функции надо запрещать в двух случаях. Во-первых, если вы выполняете оптимизацию с помощью программного конвейера (см. ниже). Во-вторых, если вы изменяете содержимое регистров A15, B14, B15, поскольку эти же регистры использует обработчик прерываний Си.

6. Регистры общего назначения для наглядности удобнее заменять символическими именами.

7. В ассемблерном коде совсем не обязательно указывать, какое вычислительное устройство выполняет ту или иную команду. Например, можно вместо MPY.M1 A1,A2,A3 написать просто MPY A1,A2,A3. Ассемблер сам определит, что эту команду должен выполнить умножитель .M1.

Приведенный код для наглядности не оптимизирован с помощью параллельных команд. Можно выполнить такую оптимизацию, но при этом следует быть внимательным, так как вы легко можете изменить содержимое регистра до того, как воспользоваться им. Совершить такую ошибку легко, а найти очень трудно.

Рис. 5. Ассемблерный код функции

```
;----------------
; A4 - указатель на inBuf[]
; B4 - указатель на outBuf[]
; A6 - размер входного буфера
; B6 - указатель на параметры
;
; typedef struct
; {
;     int  par0;
;     int  par1;
;     int  par2;
; } SParam;
;
; int Fnx( int *inBuf, int *outBuf,
;          int size, SParam *pParam );
;
; Return - размер выходного буфера
;----------------
;Присвоение регистрам символических имен

cnt         .set    A1    ;счетчик циклов
inp         .set    A2    ;входной отсчет
ptrIn       .set    A4    ;указатель на inBuf[]
retVal      .set    A5    ;возвращаемое значение
par0        .set    A6    ;хранит pParam->par0
par1        .set    A7    ;хранит pParam->par1
par2        .set    A8    ;хранит pParam->par2
outp        .set    B2    ;выходной отсчет
retAdr      .set    B3    ;точка выхода из функции
ptrOut      .set    B4    ;указатель на inBuf[]
pParam      .set    B6    ;указатель на параметры
keeCSR      .set    B7    ;старое значение CSR

;----------------
; Fnx() - точка входа
;----------------
        .global _Fnx
_Fnx
        SUB     A6, 1, cnt

; Запретить прерывания

        MVC     CSR, keeCSR
        AND     keeCSR, -2, B9
        MVC     B9, CSR

; Освободить регистры A10 - A15, B10 - B15
; для временного использования

        .bss    stack, 12*4

        MVK     stack, A9
||      MVK     stack, B9

        MVKH    stack, A9
||      MVKH    stack, B9

            STW     A10,    *+A9[0]
||          STW     B10,    *+B9[1]
            STW     A11,    *+A9[2]
||          STW     B11,    *+B9[3]
            STW     A12,    *+A9[4]
||          STW     B12,    *+B9[5]
            STW     A13,    *+A9[6]
||          STW     B13,    *+B9[7]
            STW     A14,    *+A9[8]
||          STW     B14,    *+B9[9]
            STW     A15,    *+A9[10]
||          STW     B15,    *+B9[11]

; Подгрузить параметры

            LDW     *+pParam[0],    par0
            LDW     *+pParam[1],    par1
            LDW     *+pParam[2],    par2

; Подготовить возвращаемое значение

            ZERO    retVal

;----------------
; Выполнять основной цикл

Loop:
            LDW     *ptrIn++, inp
            NOP     4
  ..............................

            STW     outp,    *ptrOut++
||          ADD     retVal, 1, retVal
  [cnt]     SUB     cnt, 1, cnt
||[cnt]     B       Loop

        NOP     5
;>>>> BRANCH OCCURS

; Основной цикл закончен
;----------------
```

```
; Выгрузить параметры
        STW     par0, *+pParam[0]
        STW     par1, *+pParam[1]
        STW     par2, *+pParam[2]

; Поместить возвращаемое значение в A4

        MV      retVal, A4

; Восстановить регисрты A10 - A15, B10 - B15

        MVK     stack, A9
||      MVK     stack, B9

        MVKH    stack, A9
||      MVKH    stack, B9

        LDW     *+A9[10], A15
||      LDW     *+B9[11], B15
        LDW     *+A9[8], A14
||      LDW     *+B9[9], B14
        LDW     *+A9[6], A13
||      LDW     *+B9[7], B13
        LDW     *+A9[4], A12
||      LDW     *+B9[5], B12
        LDW     *+A9[2], A11
||      LDW     *+B9[3], B11
        LDW     *+A9[0], A10
||      LDW     *+B9[1], B10

; Восстановить CSR и
; AMR, если он использовался

        MVC     keeCSR, CSR
        ZERO    B2
        MVC     B2, AMR

; Вернуться в вызывающую функцию

        B       retAdr
        NOP     5
```

Вообще говоря, всего этого можно не делать, а воспользоваться ассемблерным оптимизатором. Это очень удобное новое средство, включенное фирмой Texas Instruments в состав компилятора Си.

Очень полезное упражнение – воспользоваться ассемблерным оптимизатором, а затем проанализировать сгенерированный им код. Даже при оптимизации «ручкой», рекомендуется это упражнение проделать несколько раз с различными кодами.

Программный конвейер

Программный конвейер (англ. Software pipelining) – это основной метод, с помощью которого осуществляется вся оптимизация для процессора 'C6201. Метод сложен для начального понимания, однако если научиться им пользоваться, то производительность ваших программ может повыситься во много раз. Добавим, что программный конвейер предназначен для функций, циклически обрабатывающих буфер данных.

Посмотрим, как он работает. На *рис. 6.* показан пример циклического кода, выполняющего загрузку отсчета из входного буфера, суммирование его с содержимым регистра B4 и вывод результата в выходной буфер. Каждый отсчет здесь обрабатывается за 14 тактов.

Рис. 6. Пример кода без программного конвейера

```
;A1 - счетчик циклов
;A2 - указатель входного буфера
;B2 - указатель выходного буфера
;B4 - константа
Loop:
        LDW     .D1     *A2++,A3
        NOP             4
        ADD     .L2X    A3, B4, B3
        STW     .D2     B3, *B2++
 [A1]   SUB     .L1     A1, 1, A1
 [A1]   B       .S2     Loop
        NOP             5
;>>>> BRANCH OCCURS
```

Следует вспомнить, что процессор 'C6201 параллельный и пишет код, показанный на *рис. 7.* Каждый отсчет будет обрабатываться за семь тактов, но быстрее без программного конвейера уже не получится.

Рис. 7. Пример кода без программного конвейера, но с параллельными командами

```
Loop:
        LDW     .D1     *A2++,A3
  [A1]  SUB     .L1     A1, 1, A1
||[A1]  B       .S2     Loop
        NOP             3
        ADD     .L2X    A3, B4, B3
        STW     .D2     B3, *B2++
;>>>> BRANCH OCCURS
```

А с программным конвейером каждый отсчет можно обрабатывать за один такт. Это показано в примере на *рис. 8.* Рассмотрим этот пример. Для начала укажем, что команды SUB и B поддерживают работу цикла. Подобные команды присутствуют практически в любом циклическом коде, поэтому их подробно рассматривать не будем. Рассмотрим только работу команд LDW, ADD и STW.

Рис. 8. Пример кода с программным конвейером (основной цикл)

```
Loop:
        LDW     .D1     *A2++,A3
||      ADD     .L2X    A3, B4, B3
||      STW     .D2     B3, *B2++
||[A1]  SUB     .L1     A1, 1, A1
||[A1]  B       .S2     Loop
;>>>> BRANCH OCCURS
```

Для удобства объяснения будем считать, что: 1) цикл выполняется N раз, 2) регистр A2 указывает на входной буфер inBuf[N], 3) регистр B2 указывает на выходной буфер outBuf[N]. В этом случае работу можно представить в виде следующей таблицы:

0-й такт	LDW: начинает загружать inBuf[0] в A3 ADD: пока не работает STW: пока не работает
1-й такт	LDW: начинает загружать inBuf[1] в A3 ADD: пока не работает STW: пока не работает
.....	
5-й такт	LDW: начинает загружать inBuf[5] в A3 в регистре A3 появилось inBuf[0] ADD: суммирует B3=A3+B4 STW: пока не работает
Здесь начинается основной цикл, когда работают все три команды:	
6-й такт	LDW: начинает загружать inBuf[6] в A3 в регистре A3 появилось inBuf[1] ADD: суммирует B3=A3+B4 B3 хранит сумму inBuf[0]+B4 STW: выводит B3 в outBuf[0]
.....	
N-1-й такт	LDW: начинает загружать inBuf[N-1] в A3 в регистре A3 появилось inBuf[N-6] ADD: суммирует B3=A3+B4 B3 хранит сумму inBuf[N-7]+B4 STW: выводит B3 в outBuf[N-7]
Здесь заканчивается основной цикл	
N-й такт	LDW: уже не работает в регистре A3 появилось inBuf[N-5] ADD: суммирует B3=A3+B4 B3 хранит сумму inBuf[N-6]+B4 STW: выводит B3 в outBuf[N-6]
.....	
N+4-й такт	LDW: уже не работает в регистре A3 появилось inBuf[N-1] ADD: суммирует B3=A3+B4 B3 хранит сумму inBuf[N-2]+B4 STW: выводит B3 в outBuf[N-2]
N+5-й такт	LDW: уже не работает ADD: уже не работает B3 хранит сумму inBuf[N-1]+B4 STW: выводит B3 в outBuf[N-1]

Из приведенной таблицы видно, что все три команды работают одновременно только с 6-го по N-1-й такты. Это так называемый основной цикл программного конвейера (Pipe Loop Kernel). Но как может быть, чтобы команда «еще не работала» или «уже не работала»? Для этого программный конвейер имеет пролог (Pipe Loop Prolog), когда некоторые команды «еще не работают», и эпилог (Pipe Loop Epilog), когда некоторые другие команды «уже не работают». Код предыдущего примера с прологом и эпилогом показан на *рис .9.*

Рис. 9. Пример кода с программным конвейером (пролог, основной цикл, эпилог)

```
;>>>>>>>> PIPE LOOP PROLOG >>>>>>>>>
        LDW     .D1     *A2++,A3        ;такт 0
        LDW     .D1     *A2++,A3        ;такт 1
||      B       .S2     Loop
        LDW     .D1     *A2++,A3        ;такт 2
||      B       .S2     Loop
        LDW     .D1     *A2++,A3        ;такт 3
||      B       .S2     Loop
        LDW     .D1     *A2++,A3        ;такт 4
||      B       .S2     Loop
        LDW     .D1     *A2++,A3        ;такт 5
||      ADD     .L2X    A3, B4, B3
||      B       .S2     Loop

;>>>>>>>> PIPE LOOP KERNEL >>>>>>>>>
Loop:                                   ;такты 6..N-1
        LDW     .D1     *A2++,A3
||      ADD     .L2X    A3, B4, B3
||      STW     .D2     B3, *B2++
||[A1] SUB     .L1     A1, 1, A1
||[A1] B       .S2     Loop
;>>>> BRANCH OCCURS

;>>>>>>>> PIPE LOOP PROLOG >>>>>>>>>
        ADD     .L2X    A3, B4, B3      ;такт N
||      STW     .D2     B3, *B2++
        ADD     .L2X    A3, B4, B3      ;такт N+1
||      STW     .D2     B3, *B2++
        ADD     .L2X    A3, B4, B3      ;такт N+2
||      STW     .D2     B3, *B2++
        ADD     .L2X    A3, B4, B3      ;такт N+3
||      STW     .D2     B3, *B2++
        ADD     .L2X    A3, B4, B3      ;такт N+4
||      STW     .D2     B3, *B2++
        STW     .D2     B3, *B2++       ;такт N+5
```

Обратим внимание, что команды ветвления B начинают появляться в каждом такте, начиная уже с 1-го. Благодаря этому удается «закрутить» основной цикл так, чтобы на каждом такте вплоть до N-1-го выполнялся переход на метку Loop. Необходимо только не ошибиться и поместить в регистр A1 (счетчик циклов) правильное значение. В нашем случае в регистр A1 на этапе инициализации надо поместить N-13-й.

Если быть достаточно наблюдательным, то можно заметить, что на каждом такте основного цикла происходит два обращения к памяти: одно чтение и одна запись. Поскольку память 'C6201 построена по чересстрочному принципу, то за один такт можно обратиться к двум 32-разрядным ячейкам только в том случае, если одна из них имеет четный адрес, а другая – нечетный. В противном случае каждая команда основного цикла будет выполняться за два такта процессора. Чтобы учесть это свойство памяти в нашем примере, необходимо, чтобы входной буфер inBuf[N] начинался с четного адреса, а выходной outBuf[N] – с нечетного

(можно и наоборот, но они обязательно должны быть разной четности). Иногда это сделать не удается, например, входной и выходной буфер совпадают. Тогда в основной цикл следует внести перенос на один цикл с помощью регистра B5 командой MV (*рис. 10*). В этом случае входной и выходной буфера должны начинаться с адресов одинаковой четности.

Рис. 10. Пример кода с программным конвейером (основной цикл с переносом вывода)

```
Loop:
          LDW     .D1     *A2++,A3
||        ADD     .L2X    A3, B4, B3
||        MV      .S2     B3, B5 ;перенос
||        STW     .D2     B5, *B2++
||[A1]    SUB     .L1     A1, 1, A1
||[A1]    B       .S1     Loop
;>>>> BRANCH OCCURS
```

Заметим, что команда MV осуществляет перенос значения регистра B3 на один цикл, а не на один такт, хотя в данном примере это одно и то же. В данном случае под циклом понимается последовательность из одной или нескольких циклически повторяющихся команд, а под тактом – машинный такт процессора.

Конечно, приведенный пример не имеет практического смысла. Но если понять, как работает программный конвейер, то тогда можно перейти к следующему этапу.

Разработка программ с помощью программного конвейера

Еще совсем недавно процессор 'C6201 казался последним словом техники, а сейчас, после появления серийных процессоров 'C6701, 'C6211, 'C6202, 'C6203 и ожидания нового 'C64x, он кажется старым и даже допотопным. И все же разработку программного конвейера удобнее всего начинать именно с 'C6201, поскольку, с одной стороны, его ЦПУ является базовым для всех остальных процессоров, а с другой стороны, он имеет наименьшее количество факторов, которое следует учитывать в процессе разработки. Перечислим эти факторы:

- выполнение команды ветвления B отложено на пять тактов;
- выполнение команд загрузки LDB/LDH/LDW отложено на четыре такта;
- выполнение команд умножения MPY/SMPY и другое отложено на один такт;
- внутренняя память данных имеет чересстрочную организацию, позволяющую одновременное обращение к двум ячейкам, находящимся в разных столбцах.

Пример 1. Возведение в квадрат

Начнем программирование с простого примера: возвести все отсчеты буфера в квадрат (отсчеты 16-разрядные). После возведения в квадрат результаты сдвигать вправо на указанное число разрядов и младшую часть результата сохранять в том же самом месте. Выполняющая эти действия функция может быть объявлена на языке Си следующим образом:

```
void    Sqr( short *buf, int size, int shift );
```

Указатель на входной (и выходной) буфер будет передаваться в регистре A4, размер буфера – в B4, величина правого сдвига – в A6.

Прежде чем приступить к оптимизации, надо написать основной цикл без оптимизации (*рис. 11*). Как говорилось выше, все регистры следует заменить символическими именами, чтобы не думать, на какой стороне, A или B, должен лежать тот или иной регистр. Если код получился сложным, можно отладить его в отладчике, чтобы убедиться, что алгоритм реализован правильно. В нашем случае это не обязательно.

Рис. 11. Возведение буфера в квадрат (без оптимизации)

```
cnt         .set    B1
ptri        .set    B4
inp         .set    B5
sqr         .set    B6


ptro        .set    A4
outp        .set    A5
shift       .set    A6

;>>>>>>>> Точка входа в Sqr() >>>>>>
        .global _Sqr
_Sqr:
        MV      A4, ptri
        MV      A4, ptro
        SUB     B4, 1, cnt
        MV      A6, shift


;>>>>>>>> UNPIPE LOOP KERNEL >>>>>>>>>
Loop:
   LDH   *ptri++, inp     ;считать отсчет
   NOP   4
   MPY   inp, inp, sqr    ;возвести в квадрат
   NOP
   SHR   sqr, shift, outp ;сдвинуть квадрат
   STH   outp, *ptro++    ;сохранить в той же ячейке

  [cnt]     SUB   cnt, 1, cnt;уменьшить счетчик
||[cnt]     B     Loop;перейти на следующий цикл
        NOP   5
;>>>> BRANCH OCCURS

;// Завершить выполнение функции Sqr()
        B       B3
        NOP     5
;>>>> BRANCH OCCURS
```

Далее основной цикл можно начать оптимизировать. Он содержит шесть исполняемых команд –

LDH, MPY, SHR, STH, SUB и B. После внимательного изучения можно предположить, что он может быть выполнен за один или за два такта процессора. Давайте с учебной целью напишем его за два такта. Для этого расчертим лист бумаги на две колонки, в первой колонке будут стоять команды, выполняющиеся в первом такте, а во второй колонке – команды для второго такта (*рис. 12*). А затем начнем заполнять лист командами слева направо, т.е. первую команду пишем в первой строке первой колонки, вторую команду – в первой строке второй колонки, третью команду, во второй строке первой колонки и т. д.

Следите за тем, чтобы каждый регистр считывался не позже, чем через 2 такта после того, как в нем появится значение, иначе следующий цикл успеет изменить его содержимое. Например, в регистре sqr появилось значение в 4 строке на 2-м такте, значит он должен быть использован либо в этой ячейке, либо в следующей (5-я строка 1-й такт), но не позже.

Рис. 12. «Расщепление» основного цикла на два такта

Первый такт	Второй такт
LDH *ptri++, inp	(nop)
(nop)	(nop)
(nop)	(появился inp) MPY inp,inp,sqr
(nop)	(появился sqr) SHR sqr,shift,outp
(перенос outp на 1 такт)	STW outp1, *ptro++

Перенос значения регистра outp на один такт нужен для того, чтобы команды обращения к памяти LDW и STW выполнялись в разных тактах и не мешали друг другу.

Можно добавить две команды, обеспечивающие работу цикла:

```
[cnt]       SUB     cnt, 1, cnt
[cnt]       B       Loop
```

Команду B надо обязательно вписать в первый столбик, поскольку ее выполнение отложено на пять тактов. Команду SUB можно писать в тот столбик, где больше свободного места (*рис. 13*).

Рис. 13. Добавление команд SUB и B

Первый такт	Второй такт
. . .	. . .
[cnt] SUB cnt, 1, cnt [cnt] B Loop	(nop)
(nop)	(nop)
(nop)	(nop)
(здесь выполнится переход на метку Loop)	

Таким образом, основной цикл с программным конвейером будет выглядеть следующим образом:

Рис. 14. Основной цикл с программным конвейером

```
;>>>>>>>> PIPE LOOP KERNEL >>>>>>>>>
Loop:
          LDH       *ptri++, inp
||[cnt]               SUB     cnt, 1, cnt
||[cnt]               B       Loop

          MPY       inp, inp, sqr
||        SHR       sqr, shift, outp
||        STH       outp, *ptro++
;>>>> BRANCH OCCURS
```

Следующий шаг заключается в том, чтобы решить, к каким регистровым файлам должны относиться регистры основного цикла. При этом надо следить, чтобы не было попыток использовать одно и то же вычислительное устройство или кросс-путь в одной команде дважды. В нашем случае возможен такой вариант: регистры ptri, inp, sqr, cnt – на стороне B; регистры shift, outp, ptro – на стороне A. Тогда устройства будут распределены таким образом:

Рис. 15. Использование вычислительных устройств в основном цикле

```
cnt         .set    B1
ptri        .set    B4
inp         .set    B5
sqr         .set    B6

ptro        .set    A4
outp        .set    A5
shift       .set    A6

;>>>>>>>> PIPE LOOP KERNEL >>>>>>>>>
Loop:
          LDH     .D1     *ptri++, inp
||[cnt]   SUB     .L1     cnt, 1, cnt
||[cnt]   B       .S1     Loop

          MPY     .M1     inp, inp, sqr
||        SHR     .S2X    sqr, shift, outp
||        STH     .D2     outp, *ptro++
;>>>> BRANCH OCCURS
```

Полученный основной цикл следует обработать компилятором, чтобы убедиться, что конфликты отсутствуют. И наконец, можно выполнить последние действия – «развернуть» основной цикл в пролог и эпилог, добавить вход в функцию и выход из функции и отладить функцию с помощью отладчика (*рис. 16*).

Рис. 16. Окончательный вид функции возведения в квадрат

```
cnt         .set    B1
keepCSR     .set    B2
ptri        .set    B4
inp         .set    B5
sqr         .set    B6
```

```
ptro        .set    A4
outp        .set    A5
shift       .set    A6

;>>>>>>>> Точка входа в Sqr() >>>>>>
            .global _Sqr
_Sqr:
            MV      A4, ptri
||          MV      A4, ptro
||          SUB     B4, 7, cnt
||          MV      A6, shift

;// Запретить прерывания
            MVC     CSR, keepCSR
            AND     keepCSR, -2, B7
            MVC     B7, CSR
;>>>>>>>> PIPE LOOP PROLOG >>>>>>>>>

            LDH     *ptri++, inp    ;1
            NOP
            LDH     *ptri++, inp    ;2
            NOP
            LDH     *ptri++, inp    ;3
||          B       Loop
            MPY     inp, inp, sqr   ;1
            LDH     *ptri++, inp    ;4
||          B       Loop
            MPY     inp, inp, sqr   ;2
||          SHR     sqr, shift, outp;1

;>>>>>>>> PIPE LOOP KERNEL >>>>>>>>>
Loop:
            LDH     *ptri++, inp    ;5
||[cnt]     SUB     cnt, 1, cnt
||[cnt]     B       Loop

            MPY     inp, inp, sqr   ;3
||          SHR     sqr, shift, outp;2
||          STH     outp, *ptro++   ;1
;>>>> BRANCH OCCURS

;>>>>>>>> PIPE LOOP EPILOG >>>>>>>>>
            NOP
            MPY     inp, inp, sqr   ;4
||          SHR     sqr, shift, outp;3
||          STH     outp, *ptro++   ;2
            NOP
            MPY     inp, inp, sqr   ;5
||          SHR     sqr, shift, outp;4
||          STH     outp, *ptro++   ;3
            NOP
            SHR     sqr, shift, outp;5
||          STH     outp, *ptro++   ;4
            NOP
            STH     outp, *ptro++   ;5

;// Завершить выполнение функции Sqr()
;// Разрешить прерывания
            B       B3
||          MVC     keepCSR, CSR
            NOP     5
;>>>> BRANCH OCCURS
```

Сделаем несколько замечаний, облегчающих программирование.

1. В счетчик циклов cnt командой SUB загружается значение size-7, потому что команды B начинают «закручивать» цикл до того, как начинает работать команда SUB cnt,1,cnt. Чтобы определить, на сколько надо уменьшить значение size в каждом конкретном случае, лучше всего поставить следующий мысленный эксперимент. Представить себе, что в регистре cnt находится значение 1, и посчитать, сколько раз программа выполнит команду LDH. В нашем случае команда LDH выполнится 8 раз, значит, size надо уменьшить на 8-1, т. е. на 7.
2. При «разворачивании» основного цикла в пролог и эпилог удобно нумеровать каждое появление каждой команды с помощью комментариев, а затем проверить, что все команды появились одинаковое число раз. В нашем случае все команды появились по 5 раз. Сказанное не относится только к командам B и SUB, реализующим циклическую работу, а также к любым командам NOP. Эту функцию можо оптимизировать так, чтобы каждый цикл выполнялся за один такт процессора. Но с методической точки зрения это не очень интересная задача.

Последовательность шагов

Итак, для того чтобы оптимизировать функцию с помощью программного конвейера, надо выполнить следующую последовательность шагов:

1. Написать неоптимизированную функцию и убедиться в правильности выполнения алгоритма.
2. Решить, за сколько тактов можно выполнить основной цикл. Назовем это число M.
3. Нарисовать на листе бумаги M колонок.
4. Заполнить лист командами из основного цикла слева направо. Учесть, что команды STx, MPY/SMPY и B выполняются с задержкой. Указывать, когда появляются задержанные этими командами данные. Команды NOP можно заменять прочерками. Следите, чтобы один и тот же регистр не использовался больше, чем в M последовательных ячейках. При необходимости выполнять перенос значения с помощью дополнительных регистров, как показано на *рис. 8*.
5. Добавить команды SUB и B, поддерживающие работу цикла.
6. Определить, к каким сторонам, A или B, следует отнести использованные в основном цикле регистры. Если конфликты между вычислительными устройствами избежать не удается, можно попробовать передвинуть некоторые команды из одной колонки в другую и даже организовать перенос на один цикл с помощью дополнительных регистров.
7. Проверить с помощью компилятора, что все конфликты устранены.
8. Если все же конфликты устранить не удается, увеличить N на единицу и вернуться к п. 3.
9. Развернуть пролог и эпилог. Следите, чтобы все команды (кроме SUB/B и NOP) появились в прологе, основном цикле и эпилоге одинаковое число раз.

10. Определить, на сколько надо уменьшить размер буфера, прежде чем загрузить его в счетчик циклов.
11. Дописать вход в функцию и выход из нее. Не забыть запретить прерывания.
12. Отладиться.

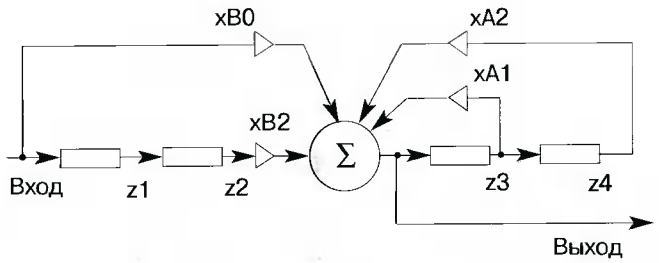
Пример 2. Рекурсивный фильтр

И все-таки предыдущий пример полезен скорее с учебной, чем с практической точки зрения. Любой оптимизатор выполнит этот пример быстрее и лучше, чем программист, потому что в нем отсутствуют зависимости. Но если в основном цикле имеется зависимость, то оптимизатор может не справиться и сгенерировать медленный код.

Зависимость – это когда следующий цикл не может начаться, пока в текущем цикле не вычислена некоторая переменная. Например, рекурсивный фильтр, звено 2-го порядка, показанное на *рис. 17*. Это звено полосового фильтра, рассчитанного по Баттерворту, имеет следующие особенности:

- множитель xB1 равен 0;
- множители xB0, xB2, xA2 меньше 1;
- множитель xA1 больше 1, но меньше 2.

Рис. 17. Звено 2-го порядка полосового фильтра Баттерворта



Реализуем одно звено 2-го порядка в виде оптимизированной функции. Будем считать, что все входные и выходные отсчеты, промежуточные значения z1 – z4 и множители xB0, xB2, xA1, xA2 лежат в диапазоне от +1 до –1 и представлены 16-разрядными целыми числами с фиксированной запятой. Этот формат чисел называется форматом Q15 и представлен на *рис. 18*. Знак занимает бит 15, мантисса – биты от 14 до 0, а фиксированная запятая расположена между битами 15 и 14.

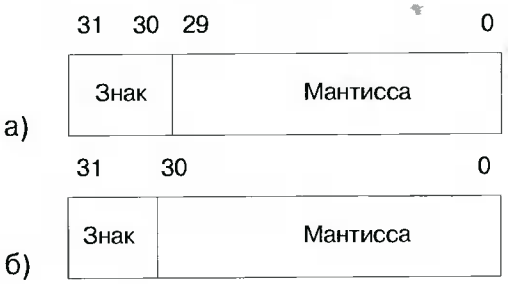
Рис. 18. Действительное число с фиксированной запятой формата Q15

Из-за использования формата Q15 необходимо учитывать следующие особенности:

1. Если перемножить два числа формата Q15 (например, z2 на B2) командой MPY, то результатом будет 32-разрядное число в формате Q30, т. е. два бита 31 и 30 будут содержать знак, а биты с 29 по 0 – мантиссу (*рис. 19*). Фактически это означает, что результат имеет вдвое меньшее значение. Поэтому при умножении лучше использовать команду SMPY, которая после умножения выполняет сдвиг влево и результат получается в формате Q31.

2. Поскольку множитель xA1 больше 1, то его придется разделить на 2, а затем результат умножения при суммировании прибавить дважды.

Рис. 19. Действительные числа с фиксированной запятой: а) формат Q30; б) формат Q31



Разрабатываемая функция будет называться Iir2() и иметь следующее описание:

```
   typedef struct
{
    int  z1, z2, z3, z4;
    int  xB2B0, xA2A1;
} SIir2;

void  Iir2 ( short inBuf[], short outBuf[],
             int size, SIir2 *pIir2 );
```

Итак, начнем выполнять оптимизацию по шагам.

Шаг 1. Напишем неоптимизированную функцию (*рис. 20*).

Рис. 20. Неоптимизированная функция Iir2()

```
;>>>>>>>> Точка входа в Iir2() >>>>>>
        .global _Iir2
_Iir2:
        SUB       A6, 1, cnt
        MV        A4, ptri
        MV        B4, ptro


;>>>> Запретить прерывания
        MVC       CSR, B1
        AND       B1, -2, B1
||      MV        B1, keepCSR
        MVC       B1, CSR

;>>>> Загрузить параметры
        MV        B6, A1
||      MV        B6, B1
        LDW       *+B1[0], z1
||      LDW       *+A1[1], z4
        LDW       *+B1[2], z3
||      LDW       *+A1[3], z2
        LDW       *+B1[4], xB2B0
||      LDW       *+A1[5], xA2A1

;>>>>>>>> Основной цикл
Loop:
        LDH       *ptri++, inp
```

```
        NOP       3
        SMPYLH    z2, xB2B0, mpB2
||      SMPYH     z4, xA2A1, mpA2
||      MV        z1, z2

        SMPY      inp, xB2B0, mpB0
||      MV        inp, z1

        SMPYHL    z3, xA2A1, mpA1
||      ADD       mpB2, mpA2, mpB2
||      MV        z3, z4

        ADD       mpB2, mpB0, mpB0

        ADD       mpB0, mpA1, z3

        ADD       z3, mpA1, z3

        SHR       z3, 16, outp
||[cnt] SUB       cnt, 1, cnt
||[cnt] B         Loop

        STH       outp, *ptro++

        NOP       4
;>>>> BRANCH OCCURS
;>>>> Конец основного цикла

;>>>> Выгрузить парметры
        STW       z1, *+B1[0]
||      STW       z2, *+A1[0]

;>>>> Разрешить прерывания и
;>>>> завершить функцию
        B         B3
||      MVC       keepCSR, CSR
        NOP       5
```

Шаг 2. Определим количество тактов M, необходимое для выполнения основного цикла. При подробном анализе алгоритма можно обнаружить, что M не может быть меньше 4. Это происходит из-за зависимости: значение z3 не может быть использовано в следующем цикле, пока не определено в текущем. А на вычисление z3 в текущем цикле требуется четыре такта; два такта занимает команда SMPY и по одному такту две команды ADD (напомним, что суммировать произведение xA1×z3 приходится дважды, так как ранее мы разделили множитель xA2 пополам). Будем считать что M равно 4.

Шаг 3. Рисуем на бумаге четыре колонки (*рис. 21*).

Шаг 4. Заполняем лист командами слева направо (*рис. 21*). Сразу получилось все правильно просто потому, что я заранее все продумано. Обычно так не получается и приходится тратить немало времени, переставляя некоторые команды из столбца в столбец, пока не получится удовлетворительный результат. Для этой работы удобно пользоваться простым карандашом и ластиком.

Шаг 5. Добавим команды SUB и B (*рис. 22*). Команда B обязательно должна выполняться на третьем такте, а команду SUB можно поместить, куда наиболее удобно.

Шаг 6. Определим, к каким регистровым файлам принадлежат используемые регистры. Возможен следующий вариант:

Сторона A: регистры z3, z4, xA2A1, mpA1, mpA2, outp, ptro.

Сторона B: регистры cnt, z1, z2, xB2B0, mpB0, mpB2, inp, ptri.

Шаг 7. В итоге основной цикл будет иметь вид, показанный на *рис. 23*. Проверим его с помощью компилятора на отсутствие конфликтов между вычислительными устройствами.

Рис. 21. «Расщепление» основного цикла на четыре такта

Первый такт	**Второй такт**	**Третий такт**	**Четвертый такт**
LDH *ptri++, inp	(nop)	(nop)	(nop)
	(появился inp)	**(появились mpB2 и mpA2; и z3- из предыдущего цикла)**	**(появился mpB0)**
SMPYLH z2, xB2B0, mpB2	SMPY inp, xB2B0, mpB0	SMPYHL z3, xA2A1, mpA1	ADD mpB2, mpB0, mpB0
SMPYH z4, xA2A1, mpA2	MV inp, z1	ADD mpB2, mpA2, mpB2	
MV z1,z2		MV z3, z4	
(появился mpB0)			
ADD mpB0, mpA1, z3	ADD z3, mpA1, z3	SHR z3,16, outp	STH outp, *ptro++

Рис. 22. Добавление команд SUB и B

Первый такт	**Второй такт**	**Третий такт**	**Четвертый такт**
		[cnt] SUB cnt,1, cnt [cnt] B Loop	(nop)
(nop)	(nop)	(nop)	(nop)
(здесь выполнится переход на метку Loop)			

Рис. 23. Оптимизированный основной цикл функции Iir2()

```
mpA1        .set    A0
mpA2        .set    A2
ptro        .set    A4
z3          .set    A5
z4          .set    A6
xA2A1       .set    A7
outp        .set    A8

cnt         .set    B0
mpB0        .set    B1
mpB2        .set    B2
ptri        .set    B4
z1          .set    B5
z2          .set    B6
xB2B0       .set    B7
inp         .set    B8

;>>>> PIPE LOOP KERNEL >>>>
Loop:
            LDH     *ptri++, inp
||          SMPYLH  z2, xB2B0, mpB2
||          SMPYH   z4, xA2A1, mpA2
||          MV      z1, z2
||          ADD     mpB0, mpA1, z3

            SMPY    inp, xB2B0, mpB0
||          MV      inp, z1
||          ADD     z3, mpA1, z3

            SMPYHL  z3, xA2A1, mpA1
||          ADD     mpB2, mpA2, mpB2
||          MV      z3, z4
||[cnt]     SUB     cnt, 1, cnt
||[cnt]     B       Loop
||          SHR     z3, 16, outp

            ADD     mpB2, mpB0, mpB0
||          STH     outp, *A4++
;>>>> BRANCH OCCURES
```

Шаг 8. Этот шаг можно пропустить.

Шаг 9. Развернем пролог и эпилог, как показано на *рис. 24*. Каждая команда, за исключением команд SUB/B и NOP, была использована по 3 раза.

Рис. 24. Основной цикл функции Iir2() с прологом и эпилогом

```
;>>>> PIPE LOOP PROLOG >>>>
            LDH     *ptri++, inp        ;1

            NOP     3

            LDH     *ptri++, inp        ;2
||          SMPYLH  z2, xB2B0, mpB2      ;1
||          SMPYH   z4, xA2A1, mpA2      ;1
||          MV      z1, z2              ;1

            SMPY    inp, xB2B0, mpB0    ;1
||          MV      inp, z1             ;1

            SMPYHL  z3, xA2A1, mpA1     ;1
||          ADD     mpB2, mpA2, mpB2    ;1
||          MV      z3, z4              ;1
||[cnt]     SUB     cnt, 1, cnt
||[cnt]     B       Loop

            ADD     mpB2, mpB0, mpB0    ;1
;>>>> PIPE LOOP KERNEL >>>>
Loop:
            LDH     *ptri++, inp        ;3
||          SMPYLH  z2, xB2B0, mpB2      ;2
||          SMPYH   z4, xA2A1, mpA2      ;2
||          MV      z1, z2              ;2
||          ADD     mpB0, mpA1, z3      ;1

            SMPY    inp, xB2B0, mpB0    ;2
||          MV      inp, z1             ;2
||          ADD     z3, mpA1, z3        ;1

            SMPYHL  z3, xA2A1, mpA1     ;2
||          ADD     mpB2, mpA2, mpB2    ;2
||          MV      z3, z4              ;2
||          SHR     z3, 16, outp        ;1
||[cnt]     SUB     cnt, 1, cnt
||[cnt]     B       Loop

            ADD     mpB2, mpB0, mpB0    ;2
||          STH     outp, *A4++         ;1
;>>>> BRANCH OCCURES

;>>>> PIPE LOOP EPILOG >>>>
            SMPYLH  z2, xB2B0, mpB2      ;3
||          SMPYH   z4, xA2A1, mpA2      ;3
||          MV      z1, z2              ;3
||          ADD     mpB0, mpA1, z3      ;2

            SMPY    inp, xB2B0, mpB0    ;3
||          MV      inp, z1             ;3
||          ADD     z3, mpA1, z3        ;2

            SMPYHL  z3, xA2A1, mpA1     ;3
||          ADD     mpB2, mpA2, mpB2    ;3
||          MV      z3, z4              ;3
||          SHR     z3, 16, outp        ;2

            ADD     mpB2, mpB0, mpB0    ;3
||          STH     outp, *A4++         ;2

            ADD     mpB0, mpA1, z3      ;3

            ADD     z3, mpA1, z3        ;3

            SHR     z3, 16, outp        ;3

            STH     outp, *A4++         ;3
```

Шаг 10. Определим, какое значение надо поместить в счетчик циклов cnt. Предположим, что size равен 2. Тогда программа выполнит команду LDH 3 раза. Значит, в cnt надо поместить значение size-1.

Шаг 11. Вход в функцию и выход из нее мы уже написали (*рис. 20*).

Шаг 12. Самостоятельная отладка.

Теперь, когда оптимизированная функция уже написана кажется, что более быстрый код написать уже не возможно из-за зависимости, связанной с переменной z3. Однако это не так. Обычно рекурсивные фильтры состоят из нескольких звеньев 2-го порядка, последовательно следующих одна за другой. Здесь удалось поместить два последовательных звена 2-го порядка в один основной цикл, который выполняется тоже за четыре такта. При этом производительность процессора достигает 1450 MIPS.

В заключение следует отметить, что программирование процессора 'C6201 (как и программирование вообще) подобно взятию интеграла – помимо знания множества технических аспектов требуется еще и значительная доля творчества.

А.А. Глуговский

Компактная процедура взвешивания

Анализируются свойства сглаживающих ступенчатых окон. Показано, что применение ступенчатых окон в сочетании с процедурой группового взвешивания в некоторых случаях позволяет резко сократить объем вычислительных затрат.

Показано, что применение ступенчатых окон позволяет использовать метод группового взвешивания, заключающийся в том, что взвешиванию подвергаются частичные, предварительно сгруппированные суммы дискретного преобразования Фурье. Метод позволяет в некоторых случаях сократить вычислительные затраты, повысить быстродействие выполнения процедуры взвешивания, значительно сокращает объем необходимой памяти.

Гармонический анализ конечных последовательностей данных связан с задачей проекции наблюдаемого сигнала на базисные векторы. При этом из континуума возможных частот только совпадающие с частотами базиса будут проецироваться на единственный базисный вектор, а все остальные частоты будут иметь ненулевые проекции на любой из векторов базисного разложения. Это явление обычно называют размыванием или просачиванием спектра. Для подавления эффекта просачивания обычно используют взвешивающие (или что то же самое – сглаживающие) окна. Классическая процедура взвешивания временной последовательности заключается в том, что каждое значение входного процесса умножается на соответствующий весовой коэффициент.

Групповое взвешивание. Наряду с применением гладких взвешивающих окон в некоторых приложениях возможно применение ступенчатых окон. Их особенность – неизменность значений взвешивающих коэффициентов на определенном интервале взвешивания. Это позволяет разбивать полную сумму вычисления коэффициентов ДПФ на несколько частичных сумм (групп), соответствующих количеству ступеней окна, производя операцию взвешивания (умножения на взвешивающий коэффициент) не по каждой отдельной входной точке ДПФ, а по результатам частичного суммирования. Далее результаты группового взвешивания суммируются. В некоторых случаях это может существенно сократить вычислительные затраты. Алгоритм группового взвешивания выглядит следующим образом:

$$Aj=\sum_{i=1}^{N} WiS_i = W_1\sum_{i=1}^{m}S_1 + W_2\sum_{i=m+1}^{p}S_2 + W_3\sum_{i=p+1}^{l}S_3 + \ldots + W_N\sum_{i=r+1}^{N}S_N ,$$

где Aj – j – коэффициент ДПФ; Si – i-е произведение значений входного потока данных Xi на комплексные экспоненты вида Exp(2πij/N); Wi – i-е значение взвешивающего множителя;

$\sum_{i=1}^{m}S_1$, $\sum_{i=m+1}^{p}S_2$, $\sum_{i=p+1}^{l}S_3$, $\sum_{i=r+1}^{N}S_N$ – групповые (частичные) суммы,

а $W_1, W_2, \ldots W_N$ – групповые взвешивающие коэффициенты.

В силу симметричности окна $W_1=W_N$, $W_2=W_{N-1}$ и т. д. Можно провести еще одну перегруппировку частичных сумм. В этом случае выражение для A_J приобретает вид

$$A_j= W_1(\sum_{i=1}^{m}S_1+\sum_{i=r+1}^{N}S_N) + W_2(\sum_{i=m+1}^{p}S_2+\sum_{i=m+a}^{q}S_{N-1}) + \ldots. +$$

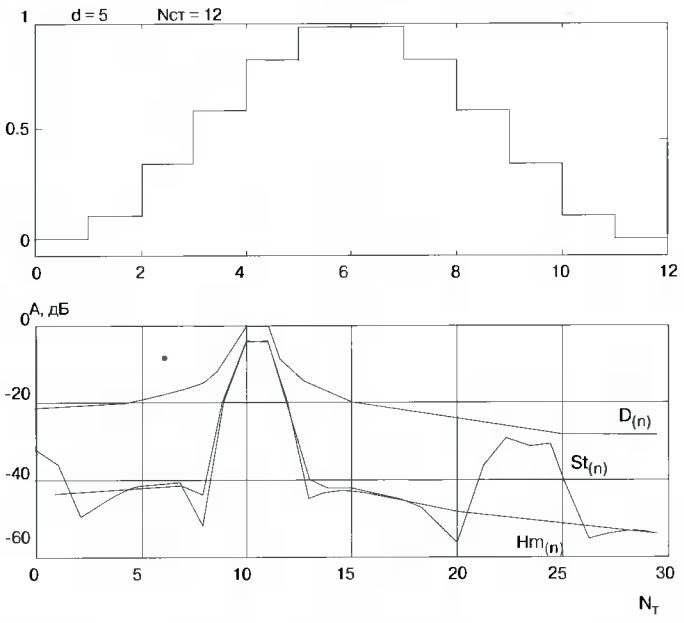
$$W_{N/2}(\sum_{jme}^{f}S_{N/2} + \sum_{i=g}^{d}S_{N/2+1}),$$

где m,p,l,e,f,d,g – параметры разбиения окна.

Алгоритм группового взвешивания может эффективно использоваться в задачах обнаружения сигнала на нескольких заранее известных частотах, например в приемниках двухтонального силового сигнала (DTMF) систем телефонной связи.

Рассмотрим пример реализации алгоритма группового взвешивания применительно к ступенчатому окну Хэмминга. На *рис.* 1 приведены сравнительные результаты расчета воздействия сглаживающих окон различной формы на монохроматический сигнал с частотой, лежащей точно посередине между 11 и 12 бинами (выходными точками) ДПФ. При этом D(n) – результат для прямоугольного окна, H(n) – для гладкого окна Хэмминга, St(n)– для 12-ступенчатого окна Хэмминга с использованием алгоритма группового взвешивания.

Рис. 1



Сравнение результатов показывает, что характеристики ступенчатого окна Хэмминга в процедуре группового взвешивания сопоставимы по качеству сглаживания с некоторыми гладкими классическими окнами: треугольным, Рисса и другим, и дают подавление боковых лепестков до 30 дБ. При этом не происходит заметной потери разрешающей способности по сравнению с гладким классическим окном Хэмминга.

Оценим эффективность применения процедуры группового взвешивания для DTMF-приемника. Если DTMF-приемник работает, например, со 180-точечным сигналом, то процессор перед началом анализа сигнала должен выполнить процедуру сглаживания, заключающуюся в 180 умножениях входного потока на 180 весовых коэффициентах. Для хранения 180 весовых коэффициентов необходимы 180 ячеек памяти сигнального процессора.

Использование ступенчатого окна в процедуре группового взвешивания предполагает оперирование с N/2 коэффициентами , где N – количество ступеней окна. Отсюда следует, что для 12-ступенчатого окна необходимый объем памяти для хранения взвешивающих коэффициентов составляет 6 ячеек.

В DTMF-приемниках анализ входного потока ведется в 8 точках частотной оси. Применение процедуры группового взвешивания в комбинации с N-ступенчатым окном требует выполнения N/2 операций умножения, что в случае 12-точечного окна и DTMF-приемника составляет 48 операций.

Таким образом, применение 12-ступенчатого окна и процедуры группового взвешивания в DTMF-приемнике позволяет сократить соответствующий объем умножений в 4 раза, а объем необходимой памяти для хранения весовых коэффициентов в 30 раз.

ЛИТЕРАТУРА

1. *Хэррис Ф.Дж.* Использование окон при гармоническом анализе методом дискретного преобразования Фурье. – ТИИЭР, 1978, т. 66, №1, январь, с. 60 – 96.
2. *Глуговский А.А.* Исследование некоторых сглаживающих окон ступенчатой формы. 2-я Международная конференция «Цифровая обработка сигналов и ее применение». М., 1999, т. II, с. 521.

В.Б. Стешенко,
к. т. н., доцент кафедры СМ-5 "Автономные информационные и управляющие системы" МГТУ им. Н.Э.Баумана

Программируемые логические интегральные схемы: обзор архитектур и особенности применения в аппаратуре ЦОС

Программируемые логические интегральные схемы (ПЛИС) становятся в последнее время все более распространенной и привычной элементной базой для разработчиков цифровых устройств. Последние годы характеризуются резким ростом плотности упаковки элементов на кристалле, многие ведущие производители либо начали серийное производство, либо анонсировали ПЛИС с эквивалентной емкостью более 1 миллиона логических вентилей. Цены на ПЛИС неуклонно падают. Так, еще год-полтора назад ПЛИС емкостью 100 000 вентилей стоила в Москве в зависимости от производителя, приемки, быстродействия от 1500 до 3000 у.е., то сейчас такая микросхема стоит от 50 до 350 у.е., то есть цены упали практически на порядок, и эта тенденция устойчива. Что касается ПЛИС емкостью 10 000 – 30 000 логических вентилей, то появились микросхемы стоимостью менее 10 у.е.

В этой связи появляется ряд вопросов, связанных с тем, какую элементную базу и как использовать в новых разработках, а также при проведении модернизации существующих систем.

Рассмотрим особенности выбора элементной базы на примере проектирования устройств цифровой обработки сигналов.

Современные алгоритмы обработки сигналов функционально можно разделить на следующие основные классы.

1. Алгоритмы цифровой фильтрации (в т.ч. алгоритмы нелинейной, оптимальной, адаптивной фильтрации, эвристические алгоритмы, полиноминальные фильтры, алгоритмы фильтрации изображений и др.). Подробная классификация алгоритмов цифровой фильтрации и перспективы путей реализации алгоритмов на ПЛИС приведены в работе [2].
2. Алгоритмы, основанные на применении ортогональных преобразований (быстрые преобразования Фурье, Хартли, Уолша, Адамара, преобразование Карунена – Лоэва и др.).
3. Алгоритмы, реализующие кодирование и декодирование, модуляторы и демодуляторы, в том числе сложных сигналов (псевдослучайных, хаотических и др.).
4. Алгоритмы интерфейсов и стандартных протоколов обмена и передачи данных.

Далее рассмотрим перспективы тех или иных путей реализации алгоритмов ЦОС на базе ПЛИС.

Реализация алгоритмов ЦОС на базе ПЛИС

Основными преимуществами ПЛИС при применении в средствах обработки сигналов являются:

- ❑ высокое быстродействие;
- ❑ возможность реализации сложных параллельных алгоритмов;
- ❑ наличие средств САПР, позволяющих провести полное моделирование системы;
- ❑ возможность программирования или изменения конфигурации непосредственно в системе;
- ❑ совместимость при переводе алгоритмов на уровне языков описания аппаратуры (VHDL, AHDL, Verilog и др.).
- ❑ совместимость по уровням и возможность реализации стандартного интерфейса;
- ❑ наличие библиотек мегафункций, описывающих сложные алгоритмы;
- ❑ архитектурные особенности ПЛИС как нельзя лучше приспособлены для реализации таких операций, как умножение, свертка, и т.п.

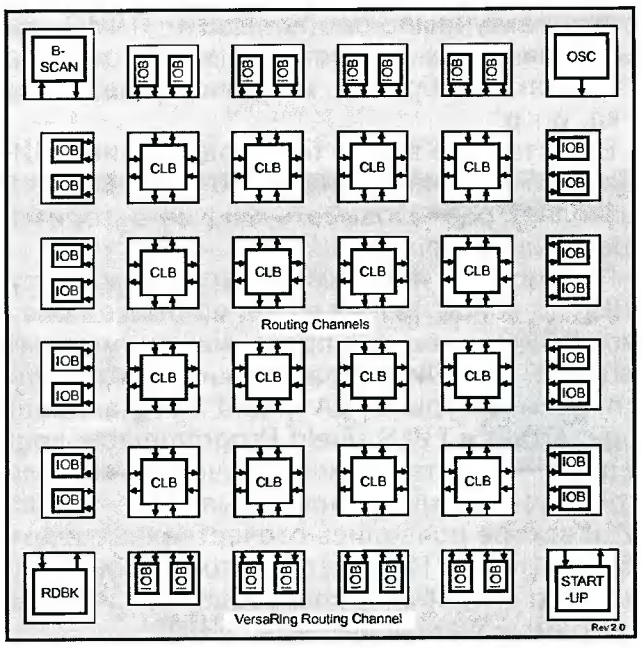
В настоящее время быстродействие ПЛИС достигло величин порядка 250 – 300 МГц, что позволяет реализовывать многие алгоритмы в радиодиапазоне.

Рассмотрим историю развитию архитектур ПЛИС. В конце 1970-х годов на рынке появились ПЛИС, имеющие программируемые матрицы “И” и “ИЛИ”. В зарубежной литературе это архитектуры FPLA (Field Programmable Logic Array) и FPLS (Field Programmable Logic Sequensers). В те времена отечественная электронная промышленность была еще “на плаву” и вскоре появились отечественные схемы К556РТ1,РТ2,РТ21. Недостаток такой архитектуры – слабое использование ресурсов программируемой матрицы “ИЛИ”.

Идя по пути совершенствования такой архитектуры, разработчики ПЛИС предложили более простую и изящную архитектуру – архитектуру программируемой матричной логики (PAL – Programmable Array Logic и GAL – Gate Array Logic). Это ПЛИС, имеющие программируемую матрицу "И" и фиксированную матрицу "ИЛИ", у ПЛИС GAL на выходе имеется триггер. К этому классу относятся широкая номенклатура ПЛИС небольшой степени интеграции. В качестве примеров можно привести отечественные ИС КМ1556ХП4, ХП6, ХП8, ХЛ8, ранние разработки (середина – конец 1980-х годов) ПЛИС фирм INTEL, ALTERA, AMD, LATTICE и др. Помимо PAL- и GAL-архитектур, были разработаны ПМЛ, имеющие только одну программируемую матрицу "И", например схема 85С508 фирмы INTEL. Другим подходом к уменьшению избыточности программируемой матрицы "ИЛИ" является программируемая макрологика. ПЛИС, построенные по данной архитектуре, содержат единственную программируемую матрицу "И-НЕ" или "ИЛИ-НЕ", но за счет многочисленных инверсных обратных связей способны формировать сложные логические функции. К этому классу относятся, например, ПЛИС PLHS501 и PLHS502 фирмы SIGNETICS, имеющие матрицу "И-НЕ", а также схема XL78C800 фирмы EXEL, основанная на матрице "ИЛИ-НЕ.

Вышеперечисленные архитектуры ПЛИС содержат небольшое число ячеек, к настоящему времени морально устарели и применяются для реализации относительно простых устройств, для которых не существует готовых ИС средней степени интеграции. Естественно, для реализации серьезных алгоритмов управления или ЦОС они не пригодны.

Рис. 1 FPGA-архитектура



В начале 1980-х годов на мировой рынок микроэлектронных изделий выходят три ведущие фирмы – производители ПЛИС. В июне 1983 года основана фирма Altera Corporation (101 Innovation Drive, San Jose, CA 95134, USA, www.altera.com), в феврале 1984 – компания Xilinx, Inc. (2100 Logic Drive, San Jose, CA 95124-3400, USA, www.xilinx.com), в 1985 году – Actel Corporation (955 East Arques Avenue, Sunnyvale, CA 94086-4533, USA, www.actel.com). Эти три компании занимают до 80% всего рынка ПЛИС и являются основными разработчиками идеологии их применения. Если ранее ПЛИС являлись одним из множества продуктов, выпускаемых такими гигантами, как Intel, AMD, и др., то начиная с середины 1980-х годов на рынке ПЛИС происходит специализация и законодателями моды являются фирмы, специализирующиеся только на разработке и производстве ПЛИС.

С появлением новых производителей появились и новые архитектуры. ИС ПМЛ имеют архитектуру, весьма удобную для реализации цифровых автоматов. Развитие этой архитектуры – CPLD (Complex Programmable Logic Devices) – ПЛИС, содержащие несколько логических блоков (ЛБ), объединенных коммутационной матрицей. Каждый ЛБ представляет собой структуру типа ПМЛ, т.е. программируемую матрицу "И" и фиксированную матрицу "ИЛИ". ПЛИС типа CPLD, как правило, имеют довольно высокую степень интеграции (до 10 000 эквивалентных вентилей, до 256 макроячеек). К этому классу относятся ПЛИС семейства MAX5000 и MAX7000 фирмы ALTERA, схемы XC7000 и XC9500 фирмы XILINX, а также большое число микросхем других производителей (Atmel, Vantis, Lucent и др.).

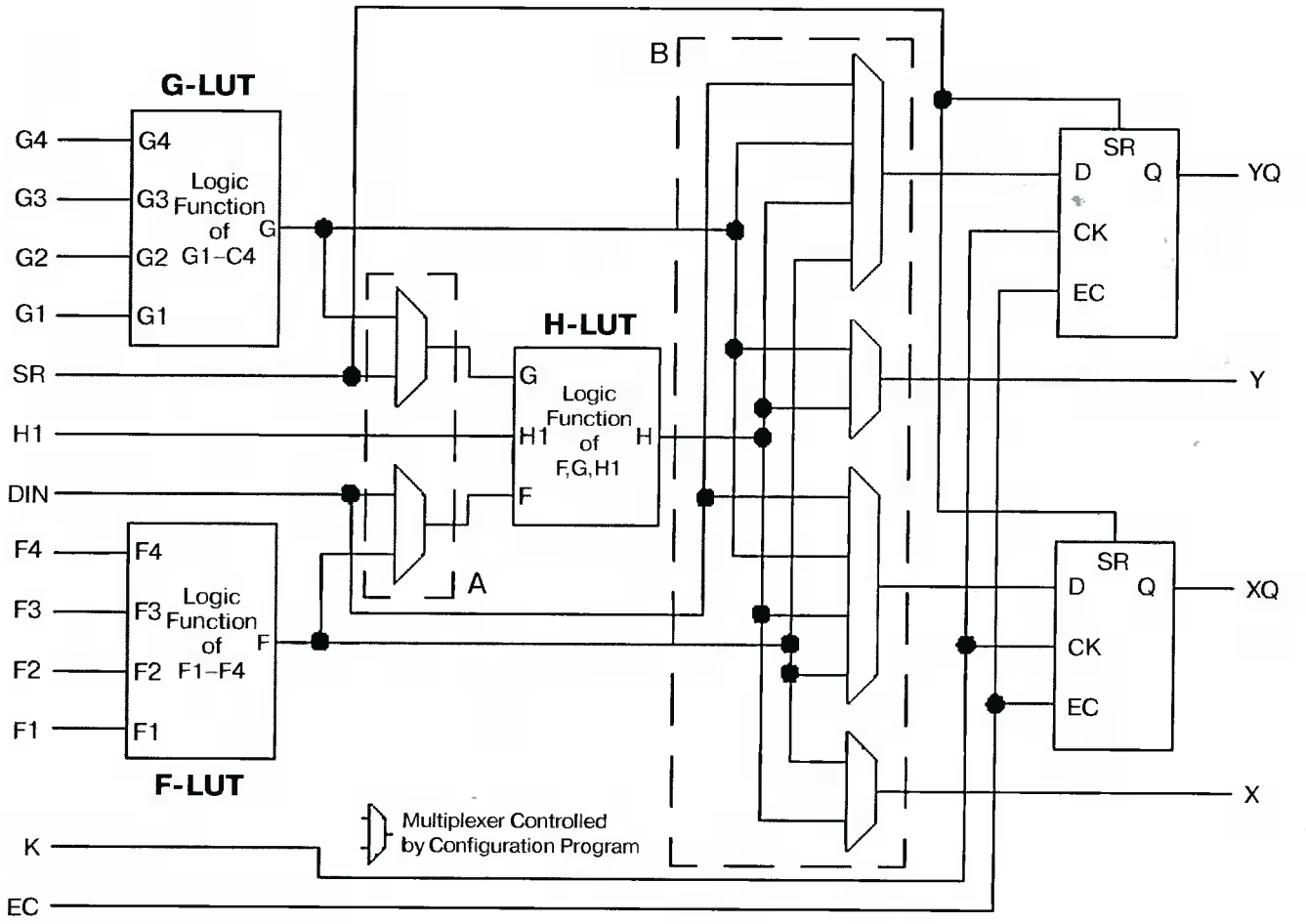
Сходную с семействами MAX3000 MAX7000 фирмы ALTERA архитектуру имеют ПЛИС ATF1500 фирмы ATMEL. Более того, они выпускаются в совместимых корпусах и поддерживают САПР MAX+PLUS II от Altera.

В качестве другого примера построения архитектур микросхем CPLD можно назвать микросхемы фирмы VANTIS (бывшая AMD) MACH4 и MACH5. Ныне VANTIS куплена Lattice.

Архитектура CPLD является весьма привлекательной для реализации цифровых автоматов, поскольку позволяет легко воплотить функции, заданные в виде совершенных дизъюнктивных нормальных форм. Они незаменимы при замене сложных схем, реализованных на обычной логике. Однако следует помнить, что несмотря на наличие в ПЛИС многих производителей режима эмуляции открытого коллектора, использовать его не всегда разумно и для интерфейса с внешними узлами удобно использовать ИС стандартных серий.

Однако CPLD-ПЛИС не очень удобны для реализации алгоритмов цифровой обработки

Рис.2. Структура CLB семейства Spartan фирмы XILINX



сигналов. Дело в том, что практически при реализации алгоритмов ЦОС требуется выполнение операций задержки на такт, перемножения и суммирования многоразрядных чисел. Настоящая революция в средствах ЦОС произошла с появлением ПЛИС, имеющих архитектуру Field Programmable Gate Array (FPGA). К FPGA относятся ПЛИС XC2000, XC3000, XC4000, Spartan фирмы XILINX, ACT1, ACT2 фирмы ACTEL, а также семейства FLEX8000 фирмы ALTERA, некоторые ПЛИС Atmel и Vantis.

Типичным примером FPGA-ПЛИС могут служить микросхемы семейства Spartan фирмы XILINX (*рис.1*).

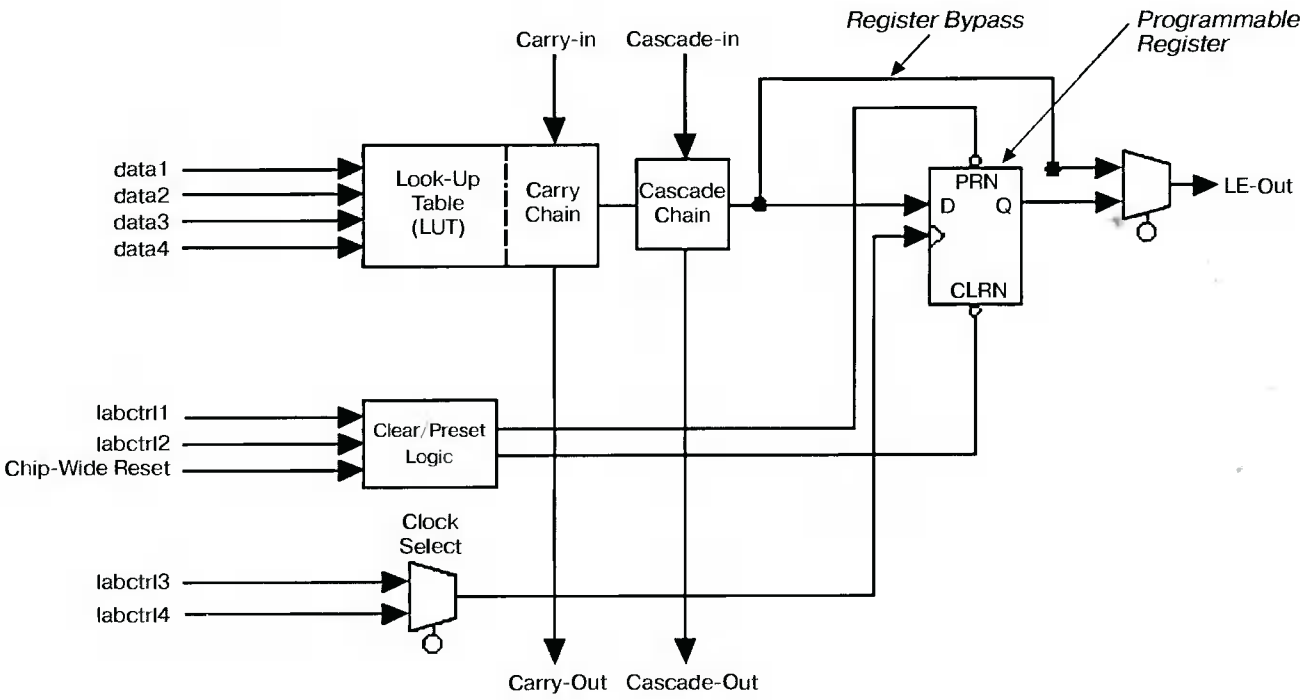
Множество конфигурируемых логических блоков (Configurable Logic Blocks (CLBs) объединяются с помощью матрицы соединений. Характерными для FPGA-архитектур являются элементы ввода-вывода (input/output blocks (IOBs), позволяющие реализовать двунаправленный ввод/вывод, третье состояние и т.п. На *рис.2* приведена структура CLB семейства Spartan фирмы XILINX.

Особенностью современных FPGA-ПЛИС является возможность тестирования узлов с помощью порта JTAG (B-scan), а также наличие внутреннего генератора (Osc) и схем управления последовательной конфигурацией.

ПЛИС, построенные по архитектуре FPGA, состоят из логических блоков (ЛБ) и коммутирующих путей – программируемых матриц соединений. Логические блоки таких ПЛИС состоят из одного или нескольких относительно простых логических элементов, в основе которого лежит таблица перекодировки (ТП, Look-up table, LUT), программируемый мультиплексор, D-триггер, а также цепи управления.

На *рис.3* приведена структура ЛЭ ПЛИС семейства FLEX6000 фирмы Altera. В основе ЛЭ лежит четырехвходовая таблица перекодировок (ТП, LUT, Look-up Table). Кроме того, в состав ЛЭ входят цепи ускоренного цепочечного переноса (carry-in, carry-out) и каскадирования (cascade-in, cascade-out). Триггер ЛЭ может быть сконфигурирован с помощью логики сброса-установки (clear/preset logic), тактируется одним из сигналов, выбираемых логикой тактирования (clock select). При необходимости сигнал с выхода ТП может быть по-

Рис.3. Структура ЛЭ ПЛИС семейства FLEX6000 фирмы Altera



дан на выход ЛЭ в обход триггера (register bypass).

Для обеспечения минимальной задержки при реализации сложных арифметических функций, таких, как счетчики, сумматоры, вычитатели, и т.п., используется организация ускоренных цепочечных переносов (carry chain) между ЛЭ. Логика ускоренных переносов автоматически формируется компилятором САПР MAX+PLUS II или вручную при описании проекта.

При организации цепочечных переносов первый ЛЭ каждого ЛБ не включается в цепочку цепочечных переносов, поскольку он формирует управляющие сигналы ЛБ. Вход первого ЛЭ в каждом ЛБ может быть использован для формирования сигналов синхронной загрузки или сброса счетчиков, использующих цепочечный перенос.

Большинство FPGA выпускаются по технологии SRAM, поэтому для их конфигурации требуется специальной ПЗУ или контроллер системы. В этом отношении выделяются FPGA фирмы Actel, выпускаемые по технологии Antifuse ("Антиперемычка") При программировании ПЛИС происходит образование области металлизации между слоями металлизации. По этой технологии выпускаются несколько семейств ПЛИС Actel

Дальнейшее развитие архитектуры FPGA-ПЛИС привело к появлению ПЛИС Altera FLEX10K, которые имеют встроенные реконфигурируемые модули памяти (РМП, embedded array block, EAB), позволяющие использовать ПЛИС для реализации различных устройств памяти внутри кристалла без использования внешних ЗУ.

Пожалуй, самой популярной элементной базой для реализации алгоритмов ЦОС, построения сложных устройств обработки данных и интерфейсов. Это объясняется тем, что благодаря большой логической емкости, удобной архитектуре, включающей встроенные блоки памяти (EAB, Embedded Array Block), достаточно высокой надежности и удачному соотношению цена – логическая емкость данные ПЛИС удовлетворяют разнообразным требо-

Рис.4. Архитектура ПЛИС FLEX10K

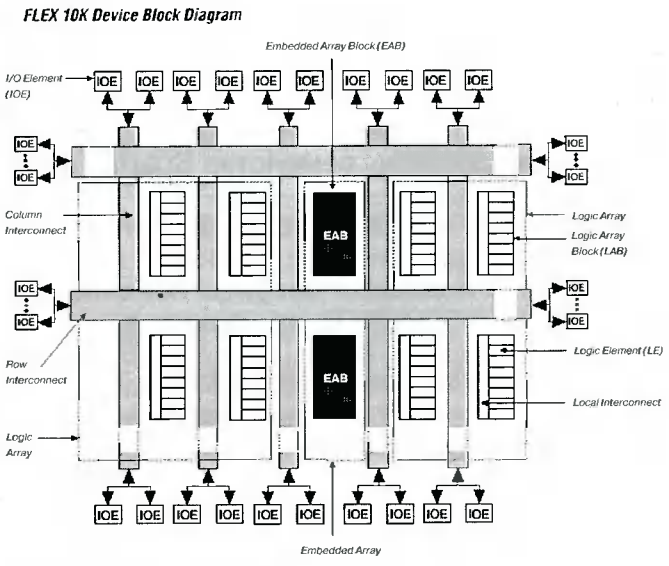


Таблица 1. Основные характеристики ПЛИС семейства APEX20K фирмы Altera

	EP20K100	EP20K160	EP20K200	EP20K300	EP20K400	EP20K600	EP20K1000
Максимальное число эквивалентных вентилей	263 000	404 000	526 000	728 000	1052000	1 537 000	2 670 000
Число лог. элементов	4160	6400	8320	11 520	16 640	24 320	42 240
Встроенные блоки памяти	26	40	52	72	104	152	264
Максимальный обьем памяти, бит	53 248	81 920	106 496	147 456	212 992	311 296	540 672
Число макроячеек	416	640	832	1152	1664	2432	4224
Число выводов пользователя	252	320	382	420	502	620	780

ваниям, возникающим у разработчика как систем ЦОС, так и устройств управления, обработки данных и т.п.

В настоящее время выпускаются ПЛИС семейств FLEX10K с напряжением питания 5 В, FLEX10KA (V) с напряжением питания 3,3 В и FLEX10KE с напряжением питания 2,5 В. Кроме того, ПЛИС семейства FLEX10KE имеют емкость встроенного блока памяти 4096 бит в отличие от ПЛИС остальных семейств, имеющих емкость EAB 2048 бит.

Обобщенная функциональная схема ПЛИС семейства FLEX10K приведена на *рис. 4*.

В основе архитектуры лежат логические блоки (ЛБ), содержащие 8 ЛЭ и локальную матрицу соединений.

Глобальная матрица соединений разделена на строки и столбцы, имеет непрерывную структуру (Fast Track Interconnect). Посередине строки располагаются встроенные блоки памяти (EAB).

Кроме того, имеются глобальные цепи управления, синхронизации и управления вводом-выводом.

Встроенный блок памяти (ВБП) представляет собой ОЗУ емкостью 2048 (4096) бит и состоит из локальной матрицы соединений, собственно модуля памяти, синхронных буферных регистров, а также программируемых мультиплексоров.

Сигналы на вход ЛМС ВБП поступают со строки ГМС. Тактовые и управляющие сигналы поступают с глобальной шины управляющих сигналов.

Выход ВБП может быть скоммутирован как на строку, так и на столбец ГМС.

Наличие синхронных буферных регистров и программируемых мультиплексоров позволяет конфигурировать ВБП как ЗУ с организацией 256x8, 512x4, 1024x2, 2048x1.

Наличие ВБП дает возможность табличной реализации таких элементов устройств ЦОС, как перемножители, АЛУ, сумматоры, и т.п., имеющих быстродействие до 100 МГц (конечно при самых благоприятных условиях реальное быстродействие арифметических устройств, реализованных на базе ВБП, составляет 10 – 50 МГц).

Все ПЛИС семейства FLEX10K совместимы по уровням с шиной PCI, имеют возможность как последовательной, так и параллельной загрузки, полностью поддерживают стандарт JTAG.

Развитие и разнообразие архитектур функциональных преобразователей, лежащих в основе базовых узлов ПЛИС, привели к тому, что в последние годы ПЛИС становятся основой для “систем на кристалле” (system-on-chip, SOC). В основе идеи SOC лежит интеграция всей электронной системы в одном кристалле (например, в случае ПК такой чип объединяет процессор, память, и т.д.). Компоненты этих систем разрабатываются отдельно и хранятся в виде файлов параметризируемых модулей. Окончательная структура SOC-микросхемы выполняется на базе этих “виртуальных компонентов”, называемых также “блоками интеллектуальной собственности” с помо-

Рис. 5. Архитектура ПЛИС, APEX20K

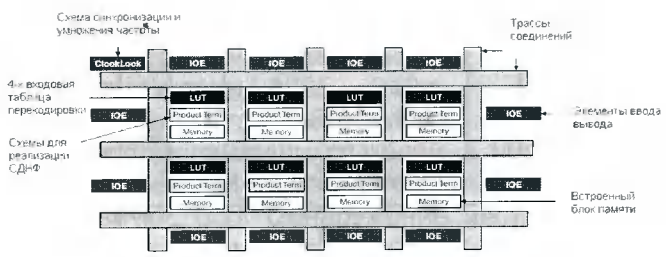


Таблица 2. Основные характеристики ПЛИС семейства Virtex фирмы Xilinx

	XCV50	XCV100	XCV150	XCV200	XCV300	XCV400	XCV600	XCV800	XCV1000
Максимальное число эквивалентных вентилей	57,906	108,904	164,674	236,666	322,970	468,252	661,111	888,439	1,124,022
Число лог. элементов	1,728	2,700	3,888	5,292	6,912	10,800	15,552	21,168	27,648
Максимальный объем памяти, бит	24,576	38,400	55,296	75,264	98,304	153,600	221,184	301,056	393,216
Число выводов пользователя	180	180	260	284	316	404	512	512	512

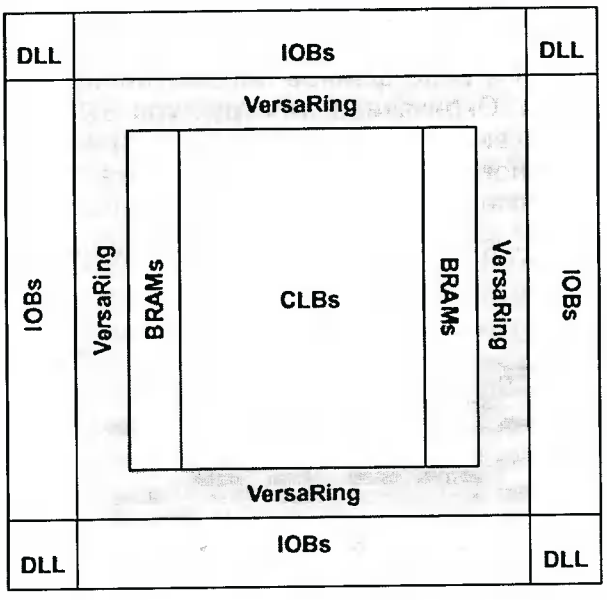
щью программ автоматизации проектирования электронных устройств – EDA (Electronic Design Automation). Благодаря стандартизации в одно целое можно объединять "виртуальные компоненты" от разных разработчиков.

Идеология построения "систем на кристалле" подстегнула ведущих производителей ПЛИС к выпуску в конце 1998 – начале 1999 года изделий с эквивалентной емкостью 1 000 000 эквивалентных вентилей и более.

Примером новых семейств ПЛИС, пригодных для реализации "систем на кристалле" является семейство APEX20K фирмы Altera, основные характеристики которого приведены в *табл. 1*.

Архитектура APEX20K сочетает в себе как достоинства FPGA ПЛИС с их таблицами перекодировок, входящими в состав логического элемента, логику вычисления СДНФ, характерную для ПЛИС CPLD, а также встроенные модули памяти (*рис.5*).

Рис.6. Архитектура ПЛИС Virtex



Сходной архитектурой обладает и семейство Virtex фирмы Xilinx, основные характеристики которого приведены в *табл. 2*.

Архитектура ПЛИС Virtex показана на *рис.6*.

Отличительной особенностью этих ПЛИС является наличие дополнительного ресурса для трассировки – VersaRing, позволяющего обеспечить более удобную трассировку входных и выходных сигналов. Так же, как и ПЛИС Altera, Virtex имеет встроенные блоки памяти.

Еще одним интересным примером FPGA архитектур являются ПЛИС VF1 фирмы Vantis.

Конфигурируемый ЛБ фирмы Vantis состоит из комбинационного (CCE) и последовательностного (CSE) ЛЭ.

ЛБ объединяются в "зерна", имеющие переменную размерность (Variable Grain Block), тем самым позволяя выполнять локальную трассировку узлов различного размера.

В дальнейшем зерна объединяются с использованием глобальной матрицы соединений.

На этом можно закончить обзор архитектур ПЛИС, хотя, конечно, за рамками этого обзора остался ряд достаточно интересных продуктов, например сверхбыстродействующие ПЛИС фирмы QuickLogick и ряда других производителей. Однако они пока еще не имеют российских дилеров и потеря не столь существенна.

Выбор ПЛИС для реализации проекта

Быстрые темпы роста продаж ПЛИС относительно высокой степени интеграции привели к тому, что они к началу 90-х годов стали практически единственной разумной альтернативой БМК и заказным ИСК к этому времени относятся и первые применения ПЛИС высокой степени интеграции в российских разработках. Этому способствовало немало причин. Во-первых, в конце 80-х годов в СССР происходил настоящий бум производства полузаказных БИС на базе БМК. Были закуплены

современные (на то время) средства САПР, такие, как COMPASS, SL2000, и др., предназначенные для разработки БМК и заказных БИС. Данные продукты имели неплохой графический интерфейс пользователя и были реализованы на мини-ЭВМ, таких, как MicroVAX, HP9000, и др. Разработчики начинали освоение матричной реализации алгоритмов и появлялся определенный опыт. Характерной особенностью процесса было то, что БМК осваивались не только предприятиями министерства электронной промышленности, но и предприятиями Миноборонпрома, Минобщемаша и др. ведомств – разработчиков аппаратуры. Таким образом разработка собственного изделия в виде БИС стала реальностью. Во-вторых, начатые при "перестройке" экономические преобразования позволили выйти на внешний рынок негосударственным фирмам-импортерам и западная элементная база перестала быть совершенно недоступной (к тому же были сняты ограничения КОКОМА). В-третьих, были освоены отечественные аналоги PAL-микросхем. Однако распад Союза и существующих производственных связей внес вои коррективы. Стало ясно, что прежних объемов финансирования уже не будет, поэтому требовалась альтернатива БМК в малосерийных и опытных разработках. Появление FPGA в России пришлось как нельзя кстати. В частности, на FPGA ПЛИС была реализована часть аппаратуры некоторых связных спутников, известно применение ПЛИС в разработках специального применения.

В настоящее время одним из активно развивающихся в России направлений разработок является аппаратура для телекоммуникаций. Известно, что, несмотря на то, что крупнейшие операторы коммуникаций в нашей стране используют в основном готовое западное оборудование, открытым остается вопрос о сопряжении его с существующими отечественными каналами связи, а также реализации дополнительных функций, необходимых потребителю. На базе технологии ПЛИС реализуются коммутаторы, системы защиты информации и т.п. Немаловажно, что специальная связь реализуется только на отечественном оборудовании, при разработке которого последние годы широко используется импортная элементная база, в том числе ПЛИС. При этом "пионерами" в применении ПЛИС высокой степени интеграции были разработчики из МО и спецслужб, в силу своей специфики первыми получившие доступ к элементной базе и системам автоматизированного проектирования.

Известно, что применение БМК и заказных ИС становится выгодным при больших объемах производства. Для того чтобы снизить затраты на "обкатку" разрабатываемого алгоритма, представляется целесообразным произвести его обкатку на ПЛИС, а затем приступать к проектированию БМК. В настоящее время в России существуют предприятия, способные выпускать БМК емкостью 30 – 50 тыс. вентилей, способные работать на частотах несколько десятков МГц. В частности, подобные разработки в состоянии выполнить коллектив АО "Микрон. Передовые технологии", возглавляемый Ю.И. Щетининым. В ряде случаев подобный подход является неплохой альтернативой применению ПЛИС в особо стойком исполнении, стоимость которых высока, да и возможность импорта в силу понятных причин затруднена.

Говорить о собственных российских разработках ПЛИС высокой степени интеграции пока, к сожалению, не приходится, однако при соответствующей позиции заказывающих ведомств данный вопрос не настолько неразрешим, как может показаться на первый взгляд.

Рассмотрим основные подходы при выборе ПЛИС для реализации проектов. Как известно, при выборе элементной базы руководствуются следующими критериями отбора:

- ❑ быстродействие;
- ❑ логическая емкость, достаточная для реализации алгоритма;
- ❑ схемотехнические и конструктивные параметры ПЛИС, надежность, рабочий диапазон температур, стойкость к ионизирующим излучениям и т.п.;

Таблица 3. Основные характеристики пакета MAX+PLUS II BASELINE ver. 9.4

Поддерживаемые устройства	EPF10K10, EPF10K10A, EPF10K20, EPF10K30, EPF10K30A, EPF10K30E (до 30 000 эквивалентных вентилей), EPM9320, EPM9320A, EPF8452A, EPF8282A, MAX7000, FLEX 6000, MAX 5000, MAX 3000A, Classic
Средства описания проекта	Схемный ввод, поддержка AHDL, средства интерфейса с САПР третьих фирм, топологический редактор, иерархическая структура проекта, наличие библиотеки параметризируемых модулей
Средства компиляции проекта	Логический синтез и трассировка, автоматическое обнаружение ошибок, поддержка мегафункций по программам MegaCore и AMPP
Средства верификации проекта	Временной анализ, функциональное и временное моделирование, анализ сигналов, возможность использования программ моделирования (симуляторов) третьих фирм

- стоимость владения средствами разработки, включающая как стоимость программного обеспечения, так наличие и стоимость аппаратных средств отладки;
- стоимость оборудования для программирования ПЛИС или конфигурационных ПЗУ;
- наличие методической и технической поддержки;
- наличие и надежность российских поставщиков;
- стоимость микросхем.

Рассмотрим с этих позиций продукцию ведущих мировых производителей ПЛИС, имеющих российских дилеров.

Фирма Altera Corporation, (101 Innovation Drive, San Jose, CA 95134, USA, www.altera.com) была основана в июне 1983 года. В настоящее время High-end-продуктом этой фирмы является семейство APEX20K, особенности архитектуры которого упоминались выше, а в *табл. 2* приведены основные параметры ПЛИС этого семейства.

Кроме того, Altera выпускает CPLD семейств MAX3000, MAX7000, MAX9000 (устаревшие серии специально не упоминаются), FPGA семейств FLEX10K, FLEX8000, FLEX6000.

Дополнительным фактором при выборе ПЛИС Altera является наличие достаточно развитых бесплатных версий САПР. В *табл. 3* приведены основные характеристики пакета MAX+PLUS II BASELINE ver. 9.6 фирмы Altera, который можно бесплатно "скачать" с сайта www.altera.com или получить на CD Altera Digital Library, на котором содержится также и полный набор документации по архитектуре и применению ПЛИС.

Кроме того, ПЛИС фирмы Altera выпускаются с возможностью программирования в системе непосредственно на плате. Для программирования и загрузки конфигурации устройств опубликована схема загрузочного кабеля ByteBlaster и ByteBlasteMV. Следует отметить, что новые конфигурационные ПЗУ EPC2 позволяют программирование с помощью этого устройства, тем самым отпадает нужда в программаторе, что, естественно, снижает стоимость владения технологией.

ПЛИС фирмы Altera выпускаются в коммерческом и индустриальном диапазоне температур.

Компания Xilinx, Inc. (2100 Logic Drive, San Jose, CA 95124-3400, USA, www.xilinx.com) была основана в феврале 1984 года, ее High-end-продуктом являются ПЛИС семейства Virtex, рассмотренные выше.

Архитектура семейства Virtex характеризуется широким разнообразием высокоскоростных трассировочных ресурсов, наличием выделенного блочного ОЗУ, развитой логикой ускоренного переноса. ПЛИС данной серии обеспечивают высокие скорости межкристального обмена – до 200 МГц (стандарт HSTL IV). Кристаллы серии Virtex за счет развитой технологии производства и усовершенствованного процесса верификации имеют достаточно низкую стоимость (до 40% от эквивалентной стоимости серии XC4000XL).

Помимо семейства Virtex, Xilinx выпускает FPGA семейств XC3000A, XC4000E, Spartan, XC5200, а также CPLD XC9500 и малопотребляющую серию CoolPLD.

Существует бесплатная версия САПР – WebPACK, поддерживающая CPLD XC9500 и CoolPLD, ввод описания алгоритма с помощью языка описания аппаратуры VHDL.

Следует заметить, что Xilinx существенно обновил модельный ряд как своих ПЛИС, так и программного обеспечения, которое теперь разрабатывается с участием фирмы Synopsys. Для ВУЗов предусмотрены значительные скидки на ПО.

ПЛИС Xilinx выпускаются как в коммерческом и индустриальном диапазоне температур, так и с военной (Military) и космической (Space) приемкой.

Компания Actel Corporation (955 East Arques Avenue, Sunnyvale, CA 94086-4533, USA, www.actel.com) была основана в 1985 году. Особенностью ПЛИС Actel является применение так называемой Antifuse-технологии, представляющей собой создание металлизированной перемычки при программировании. Данная технология обеспечивает высокую надежность и гибкие ресурсы трассировки, а также не требуется конфигурационное ПЗУ. По этой технологии выпускаются семейства ACT1, ACT2, 1200XL, а также новые семейства 54SX, A40MX и A42MX (со встроенными модулями памяти), имеющие хорошие показатели цена/логическая емкость (ПЛИС, заменяющая 300 – 350 корпусов ТТЛ, стоит $10, при частоте > 250 МГц).

Данные ПЛИС являются хорошей альтернативой БМК при среднесерийном производстве.

Новое семейство ProASIC фирмы Actel, емкостью до 500 000 эквивалентных логических вентилей, отличительной особенностью которого является энергонезависимость благодаря применению FLASH-технологии и наличие интегрированного на кристалле запоминающего устройства.

Для проектирования устройств на ПЛИС фирмы Actel бесплатно (до 31.01.2000) распространяется пакет Actel DeskTOP, содержащий средства ввода проекта, моделирования, генерации тестов разработки VeriBest и средства синтеза разработки Synplicity. Пожалуй, система проектирования Actel DeskTOP является наиболее мощным из всех бесплатных пакетов САПР ПЛИС.

К сожалению, микросхемы Actel, выпускаемые по Antifuse-технологии, требуют применения специального программатора, стоимость которого пока еще весьма высока. Од-

нако их отличает высокая надежность, поэтому они являются весьма перспективной базой для специальных применений. Так ПЛИС серии RH1280 имеют следующие характеристики:

- ❑ допустимая доза облучения – 300 000 рад;
- ❑ логическая емкость – 16 000 эквивалентных вентилей;
- ❑ быстродействие – до 135 МГц.

ПЛИС данного типа были применены в марсоходе в системе управления и обработки изображения цифровой видеокамеры робота – марсохода Pathinder и в формирователе кадра для передачи информации на Землю. В настоящее время выпущены радиационно-стойкие ПЛИС и новых семейств.

ПЛИС всех семейств Actel выпускаются в коммерческом и индустриальном диапазоне температур, а также с военной и космической приемкой.

Увеличение эквивалентной логической емкости ПЛИС привело к тому, что в 1998 – 1999 годах началось изменение отношения к программному обеспечению САПР ПЛИС как со стороны разработчиков ПО, так и пользователей. Если до конца 1990-х годов основным средством описания проекта являлся ввод схемы с помощью графических редакторов с использованием библиотек стандартных логических примитивов (логических элементов, простейших комбинационных и последовательностных функциональных узлов, аналогов стандартных ИС малой и средней степени интеграции (74-й серии), то в настоящее время актуальным является использование языков описания аппаратуры (Hardware Description Languages) для реализации алгоритмов на ПЛИС. Причем в современных САПР поддерживаются как стандартизованные языки описания аппаратуры, такие, как VHDL, Verilog HDL, так и языки описания аппаратуры, разработанные компаниями – производителями ПЛИС, специально для использования только в своих САПР и учитывающих архитектурные особенности конкретных семейств ПЛИС. Примером может служить AHDL (Altera Hardware Description Languages), подерживаемый САПР MAX Plus 2 и Quartus компании Altera. Кроме того, многие крупные фирмы – производители программного обеспечения (ПО) САПР интегральных схем активно включились в процесс создания ПО, поддерживающего ПЛИС различных производителей. Это позволяет проводить разработку алгоритмов, пригодных к реализации на ПЛИС не только разных семейств, но и различных производителей, что облегчает переносимость алгоритма и ускоряет процесс разработки. Примером таких систем являются продукты серии FPGA Express фирмы Synopsys, OrCAD Express фирмы OrCAD, продукты фирм VeryBest, Aldec, Cadence Design Systems и многих других.

С ростом логической емкости кристалла ПЛИС стало обычным явлением участие третьих фирм в разработке фирменных пакетов САПР ПЛИС. Примером являются поставляемый фирмой Xilinx пакет ПО Aliance, содержащий в своем составе компилятор FPGA Express фирмы Synopsys, пакет Actel DeskTOP, содержащий средства ввода проекта, моделирования, генерации тестов разработки VeriBest и средства синтеза разработки Synplicity; пакет FPGA Compiler II Altera Edition фирмы Synopsys; а также САПР для ПЛИС фирмы Atmel.

Также характерным в настоящее время является наличие готовых модулей (ядер – cores), мегафункций (megafunctions), предназначенных для решения достаточно сложных задач обработки сигналов. Быстрыми темпами идет разработка готовых функций усилиями третьих фирм. Так, в августе 1995 года была создана программа поддержки партнеров – разработчиков мегафункций (AMPP, ALTERA Megafunction Partners Program). К 1999 году в данной программе участвует 21 независимая фирма-разработчик. Основную массу разработок составляют мегафункции, реализующие стандартные микропроцессоры и микроконтроллеры, устройства обслуживания шинных магистралей (ISA, PCI), сетевые контроллеры и т.д. Типичными предложениями средств ЦОС являются мегафункции, реализующие быстрое преобразование Фурье (БПФ) и фильтры конечной импульсной характеристики (КИХ-фильтры). Фирма Vendor объявила о реализации фильтра бесконечной импульсной характеристики (БИХ-фильтра) и медианного фильтра. Лидером по разработке мегафункций в области ЦОС является фирма Integrated Silicon Systems (ISS). Этой фирмой разработаны библиотеки мегафункций БИХ-фильтров, фильтров обработки изображений, медианных фильтров, а также мегафункции, реализующие некоторые алгоритмы адаптивной обработки сигналов.

В составе САПР ПЛИС фирмы Xilinx имеется генератор логических ядер (CORE Generator). Сгенерированные ядра (LogiCORE) представляют собой функциональные параметризированные блоки системного уровня, предназначенные для применения в цифровой обработке сигналов. Среди ядер фирмы Xilinx разнообразные КИХ-фильтры, построенные на основе распределенной арифметики с возможностью каскадирования, интерполяции и децимации, структуры фильтров без использования умножителей, корреляторы, перемножители, аккумуляторы, сумматоры/вычитатели, делители, БПФ 1024 точки. Кроме того, фирма Xilinx поддерживает программу разработки готовых решений для САПР ПЛИС AllianceCORE.

Несмотря на вышеперечисленные программы, до сих пор на рынке отсутствует ПО для реализации нелинейных, оптимальных и большинства типов адаптивных структур, не реализованы давно известные алгоритмы последовательностной фильтрации. Между тем из бесед с разработчиками на ведущих предприятиях становится ясно, что существует огромная потребность в реализации известных и хорошо обоснованных теоретически алгоритмов, тем более что становится обычным применение импортной элементной базы и в разработках специального назначения.

Разработчики осознают необходимость создания библиотек параметризуемых мегафункций различных функциональных узлов, особенно устройств цифровой обработки сигналов. Определенные шаги в этом направлении предпринимает фирма ALTERA. Так, в августе 1995 года была основана программа поддержки партнеров – разработчиков мегафункций (AMPP, ALTERA Megafunction Partners Program). В 1997 году в данной программе участвовали более 15 независимых фирм-разработчиков. Проанализировав номенклатуру мегафункций, выпущенных в рамках данной программы, можно сказать, что вопросам ЦОС и, в частности, фильтрации уделяется недостаточное внимание. Так, из 18 партнеров AMPP не более четверти представили готовые продукты или заявили о ведущихся разработках в этой области. При этом основную массу разработок составляют мегафункции, воплощающие стандартные микропроцессоры и микроконтроллеры, устройства обслуживания шинных магистралей (ISA, PCI), сетевые контроллеры и т.д. Типичными предложениями средств ЦОС являются мегафункции, реализующие БПФ и КИХ-фильтры. Фирма Vendor объявила о реализации БИХ-фильтра и медианного фильтра. Лидером по разработке мегафункций в области ЦОС является фирма Integrated Silicon Systems (ISS). Этой фирмой разработаны библиотеки мегафункций БИХ-фильтров, фильтров обработки изображений, медианных фильтров, разработаны также мегафункции, реализующие некоторые алгоритмы адаптивной обработки. Видно, что до сих пор отсутствуют предложения в области нелинейных, оптимальных и большинства типов адаптивных структур, не реализованы давно известные алгоритмы последовательностной фильтрации. Между тем из бесед с разработчиками на ведущих предприятиях становится ясно, что существует огромная потребность в реализации известных и хорошо обоснованных теоретически алгоритмов, тем более что применение импортной элементной базы становится обычным делом и в разработках специального назначения.

Разумеется, в рамках одной статьи невозможно охватить все аспекты, связанные с архитектурой ПЛИС и их производителями. За рамками статьи остались изделия ряда фирм, практически недоступные в России. В следующих статьях цикла мы будем рассматривать вопросы, связанные с проектированием устройств ЦОС на современной элементной базе, особенности их схемотехники и конструирования, а также некоторые аспекты применения сециализированного программного обеспечения при разработке устройств.

ЛИТЕРАТУРА

[1] Вицын Н. Современные тенденции развития систем автоматизированного проектирования в области электроники. // Chip News, №1, 1997, с. 12 – 15.

[2] Губанов Д.А., Стешенко В.Б., Храпов В.Ю., Шипулин С.Н. Перспективы реализации алгоритмов цифровой фильтрации на основе ПЛИС фирмы ALTERA. // Chip News, №9 – 10, 1997, с. 26 – 33.

[3] Губанов Д.А., Стешенко В.Б. Методология реализации алгоритмов цифровой фильтрации на основе программируемых логических интегральных схем. // Сборник докладов 1-й Международной конференции "Цифровая обработка сигналов и ее применения" 30.06 – 03.07.1998, М., МЦНТИ, т. 4, с. 9 – 19.

[4] Щербаков М.А., Стешенко В.Б., Губанов Д.А. Цифровая полиноминальная фильтрация: алгоритмы и реализация на ПЛИС. // Инженерная микроэлектроника, №1 (3), март 1999, с.12 – 17.

[5] Губанов Д.А., Стешенко В.Б., Шипулин С.Н. Современные алгоритмы ЦОС: перспективы реализации. // Электроника: наука, технология, бизнес, №1, 1999, с.54 – 57.

[6] Шипулин С.Н., Губанов Д.А., Стешенко В.Б., Храпов В.Ю. Тенденции развития ПЛИС и их применение для цифровой обработки сигналов. // Электронные компоненты, №5, 1999, с. 42 – 45.

[7] Стешенко В. Школа разработки аппаратуры цифровой обработки сигналов на ПЛИС. // Chip News,1999, №8 – 10, 2000, №1,3 – 5.

[8] Стешенко В.Б. Школа схемотехнического проектирования устройств обработки сигналов. // Новые компоненты и технологии, №3 – 6, 2000.

[9] Стешенко В. ПЛИС фирмы ALTERA: проектирование устройств обработки сигналов. М.: Додека, 2000.

[10] Стешенко В.Б. Особенности проектирования аппаратуры цифровой обработки сигналов на ПЛИС с использованием языков описания аппаратуры. // Сборник докладов 2-й Международной конференции "Цифровая обработка сигналов и ее применения" 21.09 – 24.09.1999, М., МЦНТИ, т. 2, с. 307 – 314.