Программирование мультикомпьютеров на кристалле семейства SEAforth

Пётр Советов, peter@sovietov.com

1 Введение

Эта статья подготовлена по результатам моей работы в компании IntellaSys [9]. Я благодарю Михаила Монтвелишского, которому обязан многими идеями.

2 Особенности архитектуры SEAforth

Предложенное Э. В. Евреиновым в конце 50-х, начале 60-х гг. комплексное направление однородных вычислительных систем, структур и сред (ОВС) [3] включает в себя архитектуры, подобные рассматриваемой в этой статье.

Ниже изложены три основных принципа ОВС.

1. Глобальный параллелизм. Одновременное выполнение последовательных задач на большом количестве простых компьютеров.

2. Конструктивная однородность элементов и связей между ними. Однотипность компьютеров и соединение их в решётчатые структуры. Такой подход упрощает технологию производства и обеспечивает наращиваемость архитектуры.

3. Переменная логическая структура. Логические связи между взаимодействующими компьютерами изменяются динамически, в соответствии с графом алгоритма [2].

SEAforth [10] можно рассматривать как однородную вычислительную структуру, изготовленную на одном кристалле. Она состоит из асинхронно работающих вычислительных узлов (nodes).

Топология двухмерной решётки, лежащая в основе SEAforth-семейства, легко масштабируется: созданы мультикомпьютеры, имеющие 24, 40 и 144 узла на кристалле (см. рис.1). S144 (переименованный в GA144) был воплощён в кремнии создателем SEAforth-архитектуры, Чарльзом Муром [11], когда последний, покинув IntellaSys, начал работу в собственной компании GreenArrays [12]. В дальнейшем, возможно, будет использоваться и топология тора, которая позволит сократить расстояние между наиболее отдалёнными узлами.

Синхронные (небуферизированные) соединения между соседними узлами организованы в соответствии с теорией взаимодействующих последовательных процессов Ч. Хоара [5]. На время ожидания ответа от соседа узел «впадает в сон», практически

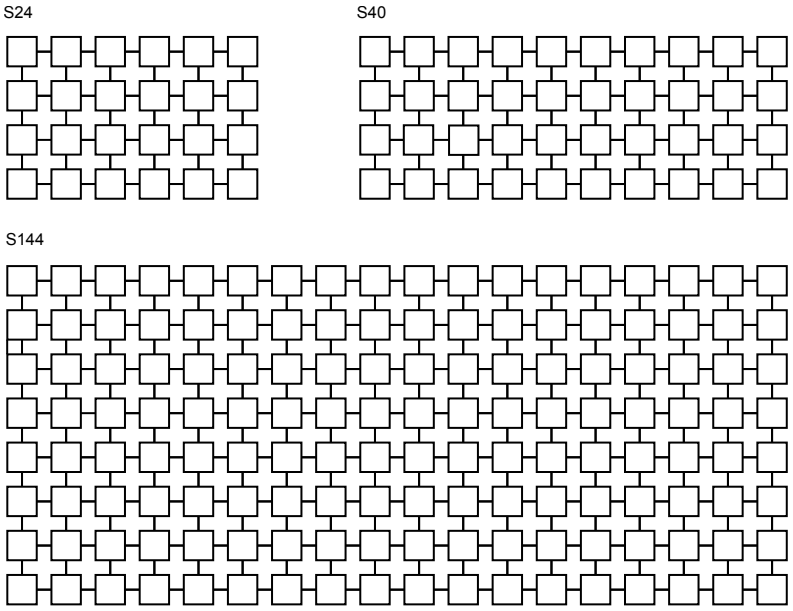

Рис. 1: Мультикомпьютеры S24, S40 и S144

не потребляя энергии в этом режиме. В случае необходимости буферизация может быть добавлена программистом.

Узлом SEAforth-решётки является 18-битный стековый компьютер C18 (см. рис. 2), с локальными ОЗУ (64 слова) и ПЗУ (64 слова) и низкоуровневым диалектом языка Форт [1] в качестве набора команд. Стеки данных (10 слов) и возвратов (9 слов) организованы, как кольцевые буферы.

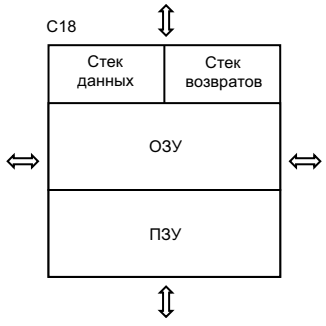

Рис. 2: Компьютер C18

Целочисленное АЛУ C18 выполняет одновременно все поддерживаемые операции при каждом изменении стека, так что очередная инструкция лишь выбирает нужный результат. Команды + (сложение) и +* (шаг умножения с 36-битным результатом) отличаются особым поведением. Они полностью выполняются за два шага АЛУ, что связано с распространением бита переноса. Если предварить одну из этих команд инструкцией, не меняющей состояние стека, то времени для получения результата окажется достаточно.

Очередное выполняемое узлом 18-битное слово представляет собой буфер до четырёх 5-битных инструкций. Выборка следующего слова начинается в тот момент, когда определяется, что остаток команд буфера инструкций не использует шину

адреса. В пределах выполняемого слова возможен особый вид программного цикла (микроцикл), без обновления буфера инструкций.

Для взаимодействия с ближайшими соседями имеются порты. С точки зрения программиста это специальные адреса по которым узлы читают и записывают значения в одном или нескольких одновременно направлениях. Непосредственно на адрес такого порта можно передать управление. В этом режиме счётчик команд не увеличивается, а буфер инструкций заполняется словами от узла-соседа. Имеется также механизм неблокирующего опроса статуса портов, связанных с данным узлом.

Отдельный узел выполняет около 700 миллионов операций в секунду. Пиковое быстродействие мультикомпьютера S40 можно оценить в 28 миллиардов операций в секунду. Это ситуация, когда все узлы полностью загружены выполнением независимых задач. Количество потребляемой мультикомпьютером энергии непосредственно зависит от активности отдельных узлов.

Узлы на внешнем периметре SEAforth-решётки могут функционировать в режиме выделенных периферийных процессоров, благодаря чему отпадает необходимость в механизме обработки прерываний, а время отклика в задачах «реального времени» становится очень малым. Периферийные узлы подключаются к различным контактам и имеют специальные процедуры в ПЗУ для управления внешними интерфейсами. Предусмотрено использование внешних модулей памяти. Имеются узлы со встроенными АЦП и ЦАП.

Программы для SEAforth требуют ручного распределения по узлам решётки. Для программиста такого рода декомпозиция может ассоциироваться с понятиями сопрограммы, объекта или процесса. Возможен графический подход: указание расположения и связей для узлов, выбираемых из библиотеки типовых блоков. Многообещающим представляется сочетание интерпретатора скриптового языка, имеющего доступ к внешней памяти и реализованного на небольшом количестве узлов, с остальной частью решётки, функционирующей в SIMD-режиме.

В целом, радикальная архитектура SEAforth представляет собой вызов программисту, любящему изящные и неординарные решения.

3 Программирование в системе VentureForth

VentureForth [13] является системой с открытым кодом на Форте (стандарт ANS Forth [14]). В неё входят симулятор, ассемблер, загрузчик и другие инструменты. Для работы рекомендуется воспользоваться полноценной Форт-системой, такой как gforth [16]. В дистрибутиве на сайте производителя имеется система SwiftForth [15], но в урезанном виде, с закрытым кодом и без поддержки плавающей запятой.

Программа на VentureForth состоит из команд подключения библиотек, описаний макроузлов и отдельных узлов, команд вызова различных инструментов.

3.1 Макроузлы

Идея макроузла возникла у меня в связи с желанием структурировать и разграничить процесс написания программы и её размещения на SEAforth-решётке. Слово **macro:** создаёт макрокоманду, результатом выполнения которой является специализированный для данного узла код. Благодаря тому, что макроузел является обычным

словом Форта достигается гибкость в обработке входных параметров и ассемблировании результата.

Ниже представлен пример «трубопровода» (см. рис. 3) на трёх узлах (1-2-3). Предполагается, что данные поступают с нулевого узла и принимаются без изменений на четвёртом. Отдельно определён макроузел **pipe** («труба»), далее его код размещён на соответствующих узлах решётки, с указанием конкретных входов и выходов.

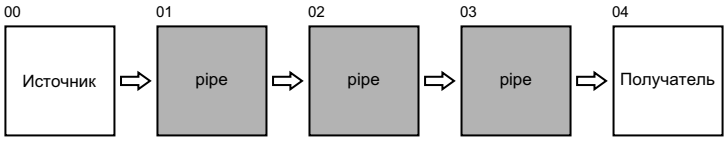

Рис. 3: «Трубопровод»

```
\ определение макроузла
macro: pipe, ( $IN $OUT)

>np equ $OUT >np equ $IN \ параметры макроузла
\ слово >np переводит номер узла в адрес соотв. порта

: start
   here =p \ точка входа
   $IN # a! $OUT # b! \ инициализация регистров

: forever @a !b forever -; \ цикл переброски данных

macro;

\ размещение на узлах решётки
01 {node 00 02 pipe, node}
02 {node 01 03 pipe, node}
03 {node 02 04 pipe, node}
```

3.2 Загрузчик

Загрузочным называется узел, через который программа попадает в мультикомпьютер, распространяясь далее по решётке. Путь загрузки представляет собой последовательность непосредственно связанных узлов, начиная с загрузочного, без повторений. Загрузчик формирует поток на основе содержимого узлов и заданного пути загрузки.

Загрузочный поток содержит специальный заголовок, считываемый процедурой из ПЗУ, и выполняет следующие действия.

1. На портах транзитных узлов располагаются «помпы»: микроциклы, «перекачивающие» информацию для последнего в пути загрузки узла.

2. ОЗУ последнего узла заполняется соответствующим содержимым.

3. На порту последнего узла выполняется код инициализации. В простейшем случае это «прыжок» на точку входа.

4. Путь загрузки уменьшается на один элемент с конца. Пока путь не пуст, повторяются действия, начиная с п. 2.

В силу специфики работы процедуры из ПЗУ загрузочный узел инициализируется только указанием адреса точки входа.

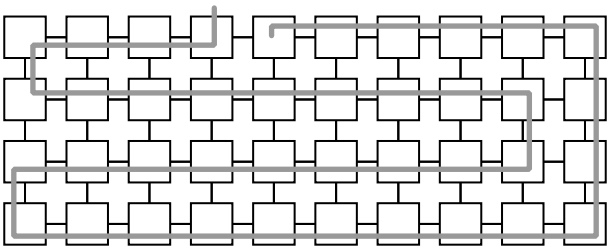

Рис. 4: Порядок обхода всех узлов решётки для **S40>33**

Загрузчик из документации VentureForth [17] (portload.f) устарел, содержит громоздкую схему инициализации и был фактически заменён в реальных программных проектах моей «альтернативной» версией (loader.f), о которой и пойдёт далее речь.

Слово **>stream** ожидает на входе путь загрузки с общим количеством задействованных узлов и формирует в ответ требуемый поток, который возможно использовать далее для передачи на загрузочный узел через соответствующий внешний интерфейс.

Пример.

```
\ формирование загрузочного потока для узлов 33, 34 и 35
33 34 35 3 >stream
```

На практике удобно пользоваться универсальными потоками, такими, например, как **s40>19**, **s40>32**, **s40>33** (см. рис. 4). В их именах отражён тип решётки (S40) и номер загрузочного узла (19, 32 или 33).

Явно указать код инициализации можно с помощью конструкции **{run ... run}**. Если код инициализации не указан, загрузчик задействует вместо него «прыжок» на точку входа (**here =p**) данного узла.

Ниже показан пример уже знакомого читателю макроузла **pipe**, который в данном случае инициализирует регистры не в ОЗУ, а прямо на порту.

```
macro: pipe, ( $IN $OUT)

>np =b >np =a \ запись параметров в переменные симулятора

here =p
```

```
: forever @a !b forever -;

\ код инициализации
{run
a @ # a! \ инициализация регистров
b @ # b!
| forever -; \ переход на forever
run}

macro;
```

3.3 Программирование отдельного узла

Код для компьютера C18 легко обозрим и во многих случаях целиком умещается на странице. Программирование узла представляет собой «ювелирную работу» по созданию миниатюрных определений-процедур. Правильно оценить необходимый для выполнения задачи объем кода способен только опытный SEAforth-программист. На первых порах бывает сложно уложить всю программу в 64 слова. Однако, после тщательной работы над кодом зачастую выясняется, что задача не просто выполнена, но в ОЗУ появилось даже некоторое незанятое место.

Имеет смысл использовать высокоуровневые программные конструкции в духе Форта, где это не противоречит требованиям по быстродействию.

Примером подобного подхода являются переменные, работа с которыми не затрагивает состояние регистров.

```
\ операции над переменной x
: x ( - x) @p+ drop @p+ ; \ чтение
: x! ( x) !p+ ; \ запись
0 , \ значение x
```

Существуют вариации на ту же тему: переменные, встраиваемые в код, переменные двойной длины и т.п.

Слова-объекты, подобные конструкции Форта CREATE ... DOES> могут сделать программу более ясной и зачастую существенно экономят память. Объединение общего кода и различных наборов данных происходит в таких словах при помощи стека возвратов, значение с которого трактуется, как адрес начала данных. Также возможно использование регистра A и соответствующих команд автоинкремента, если все наборы данных расположены друг за другом (обычно начиная с нулевого адреса) и обрабатываются последовательно. Полезными примерами слов-объектов, использующих стек возвратов, являются **poly** и **taps** из библиотеки ПЗУ.

Для оптимизации небольших линейных участков кода я написал утилиту invent, которая относится к классу супероптимизаторов [7]. Она позволяет по заданному коду на Форте автоматически построить его аналог для C18, минимальный по размеру.

3.4 Программирование взаимодействующих узлов

Факторизация слов это то, с чем сталкивается каждый программист на Форте. В архитектуре SEAforth добавляется уровень выше: факторизация узлов. Программа, распределённая по решётке, свободна от существенного влияния «побочного эффекта», узлы функционируют независимо, обмениваясь между собой сообщениями по заданным программистом протоколам. Крупные программные компоненты и блоки памяти организуются на группах узлов. Количество узлов, отданных под реализацию некоторого программного модуля, является во многих случаях ценным ресурсом и может быть сокращено за счёт понижения быстродействия данного модуля и дополнительных усилий по «сжатию» кода. Для SEAforth-архитектуры характерен особый вид оптимизации смысл которой состоит в программировании взаимодействующих узлов наилучшим с точки зрения энергоэффективности образом.

Организация мультикомпьютерных вычислений сводится к двум базовым подходам: конвейеризации и распараллеливанию [2]. Некоторые полезные алгоритмы в этой области можно почерпнуть в книге [4].

Конвейерная обработка цепью непосредственно связанных узлов встречается на SEAforth наиболее часто. Вычислительная нагрузка на ступени конвейера и фазы деятельности ступеней должны быть выравнены. Цепь узлов можно заставить работать параллельно, предварительно распространив нужные данные на узлы всех соседей. Слишком протяжённая цепь обработки вызывает накладные расходы на передачу сообщений наиболее отдалённым адресатам. Ситуацию можно улучшить, используя мультиконвейерную обработку: разбив данную цепь на несколько параллельно функционирующих коротких цепочек. Следует помнить, что передача сообщений занимает некоторое время даже при использовании выделенных узлов в качестве коммутационных элементов.

Благодаря возможности обмена программным кодом между узлами возникла событийно управляемая обработка. В её случае узлы, организующие некоторый вычислительный процесс, не содержат цикла взаимодействия по какому-либо протоколу. Единственное, что они делают, это «прыгают» на один из портов, ожидая прихода управляющей программы-события, которая вызовет ту или иную процедуру из памяти узла или изменит значение определённых ячеек ОЗУ, стеков или регистров.

Для переброски программы через заданное количество узлов удалённому адресату я придумал специальную процедуру. Она может действовать подобно загрузчику, генерируя код «помп» для транзитных узлов. В этом случае перемещается только сама программа, а все транзитные узлы заняты до тех пор, пока идёт передача.

В другом варианте код отправляющей процедуры располагается на каждом из транзитных узлов, а программа оформляется в виде пакета, в котором содержатся следующие сведения.

1. Номер узла назначения.
2. Размер программы.
3. Слова программы, поступающие на порт узла назначения.

Как только пакет был передан через узел, последний может заняться другими делами. Этот вариант отправляющей процедуры известен в библиотеке ПЗУ под именем **relay** (к данной реализации я не имею непосредственного отношения).

Благодаря событийно управляемой обработке возможна реконфигурация всей решётки или её части, то есть замена одной распределённой программы на другую. Обычно это делается с помощью дополнительных загрузочных потоков.

В ожидании на мультипорте появляется необходимость, если неизвестно, с какого направления придёт событие. Правильная работа загрузчика основывается на том факте, что изначально все узлы передают управление на мультипорты. В процессе работы загрузчика происходит «фокусировка»: управление переходит с мультипорта на конкретное направление. Мультипорты также используются для совмещения операций чтения и записи в одном адресе.

Ниже представлен вариант реализации макроузла **pipe** с помощью мультипорта.

```
macro: pipe, ( $IN $OUT)

>np swap >np port+ =b \ вычисление адреса мультипорта

here =p

: forever @b !b forever -;

{run
b @ # b!
| forever -;
run}

macro;
```

С помощью неблокирующего опроса статуса чтения/записи портов (регистр iocs) возможно организовать буферизацию обмена между некоторыми узлами или както иначе «развязать» взаимодействующие узлы в плане очерёдности выполняемых операций.

3.5 Инструменты на основе VentureForth

Симулятор VentureForth является расширяемым. В любой момент его работы возможны чтение и модификация регистров, стеков и локальной памяти узлов и прочих внутренних данных. Выполнение слова **step** представляет собой очередной такт работы симулятора. На основе данного слова я сделал несколько полезных инструментов, таких как симуляция загрузочного потока (**xload**), точки останова и графический профайлер.

Асинхронные точки останова создаются с помощью слова **bp** с именем заранее определённого слова на Форте, которое будет вызываться каждый раз, когда управление на данном узле перейдёт на адрес, где была объявлена точка останова. Слово-обработчик на Форте может, например, просматривать и изменять переменные состояния симулятора, читать и записывать данные в файлы. Наиболее часто этот механизм используется не для отладки, а при моделировании ввода-вывода и сбора разного рода статистики.

Графический профайлер вызывается с помощью слова **heatmap** («тепловая карта») с указанием необходимого для сбора статистики числа тактов симулятора. В качестве результата с помощью библиотеки SvgScript [18] создаётся графический файл в формате SVG, где изображена SEAforth-решётка, показаны актуальные связи между её элементами и цветом отражена степень вычислительной нагрузки отдельных узлов.

Имеется возможность отображения общей вычислительной загруженности решётки в процентах. Эта величина может использоваться при расчёте энергопотребления мультикомпьютером.

Примеры работы графического профайлера можно увидеть далее в статье.

3.6 Процесс разработки программ

Нижеследующие стадии SEAforth-разработки представляются наиболее существенными.

1. Создание прототипа на Форте. Для этих целей полезно иметь библиотеку поддержки SEAforth-арифметики с фиксированной запятой.
2. Декомпозиция и планирование протоколов взаимодействия узлов.
3. Создание программ для отдельных узлов, обособленное и совместное их тестирование на симуляторе. Создание макроузлов.
4. Тестирование программного проекта. Сбор различной статистики по результатам моделирования на симуляторе.
5. Сохранение проекта в одном из загрузочных форматов и тестирование на реальном мультикомпьютере.

4 Примеры программных проектов

Ниже представлен обзор трёх моих демонстрационных программ по теме звукового синтеза и описание проекта слухового аппарата, в разработке которого я принимал непосредственное участие. Все эти программы успешно функционируют на реальных мультикомпьютерах.

4.1 Musicbox

Данная программа проигрывает различные мелодии. Мелодии эти «сочиняет» компьютер методом фрактальной алгоритмической композиции [6], а звук воспроизводится с помощью полифонического синтезатора классической шестиструнной гитары на основе физического моделирования [19]. Результат в виде звуковых семплов поступает на встроенный ЦАП.

Здесь, как и в других моих проектах, главный файл содержит только команды по размещению макроузлов на решётке мультикомпьютера. Описания же самих макроузлов находятся в отдельном файле.

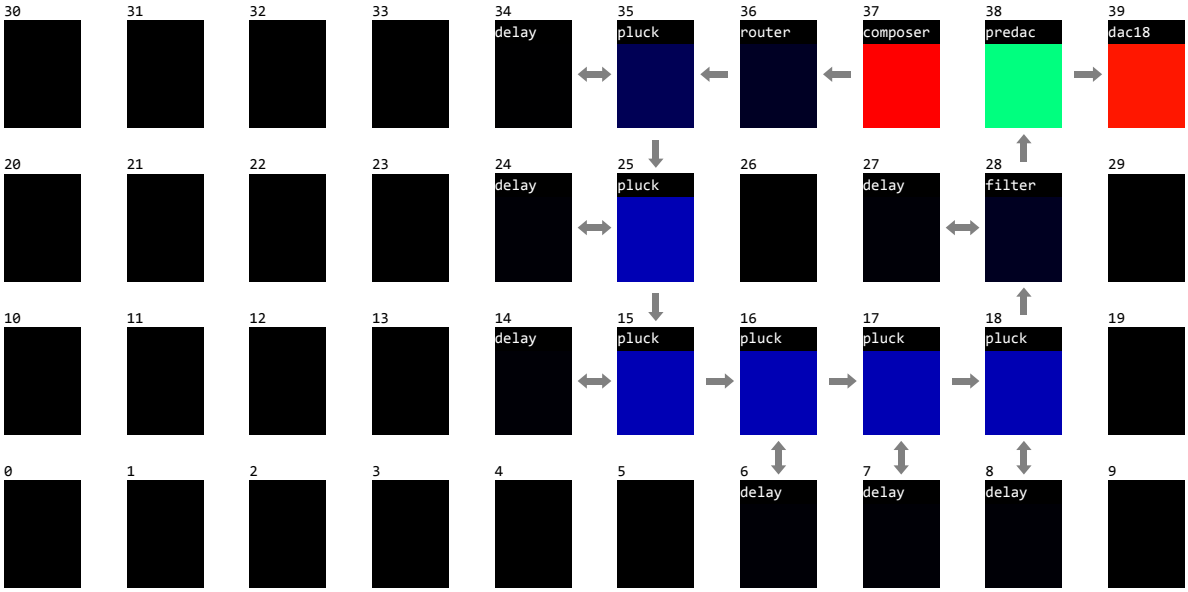

Рис. 5: Musicbox

С помощью инструмента **heatmap** построена «тепловая карта» проекта. Результат можно увидеть на рис. 5. Цвета в диапазоне от чёрного до красного отражают степень вычислительной нагрузки узлов.

Макроузел **composer** производит поток нот, которые поступают на **router**. Далее на шесть «струн» синтезатора (связка **pluck** и **delay**) приходят программные события: широковещательные сообщения сумматора и сообщения для индивидуальных голосов (доставляемые с помощью **relay**) об изменении параметров той или иной «струны».

Макроузел **delay** хранит 128 девятибитных значений на которых основана линия задержки дробной длины с интерполяцией. Чтение и запись элементов **delay** организованы макроузлом **pluck** в виде сопрограммы (команда **;:**).

Остаётся добавить, что код проекта Musicbox входит в дистрибутив VentureForth в версиях для S24 и S40.

4.2 SEAtar

SEAtar — мультитембральный полифонический музыкальный синтезатор. Игра на нём осуществляется с помощью MIDI-контроллера, такого, например, как MIDI-клавиатура. В настоящий момент поддерживаются следующие MIDI-события: «note on» (нажать ноту), «note off» (отпустить ноту), «pitch bend» (изменить высоту тона) и «program change» (выбрать тембр). В звуковых пресетах используются различные виды синтеза: аддитивный, FM-синтез, аналоговое и физическое моделирование.

Загрузка очередного пресета из флеш-памяти во время работы синтезатора (то есть реконфигурация некоторого множества узлов) возможна благодаря событийно управляемой обработке.

SEAtar состоит из «каркаса» и работающих под его управлением программ, составляющих пресеты и занимающих оставшуюся часть решётки мультикомпьютера

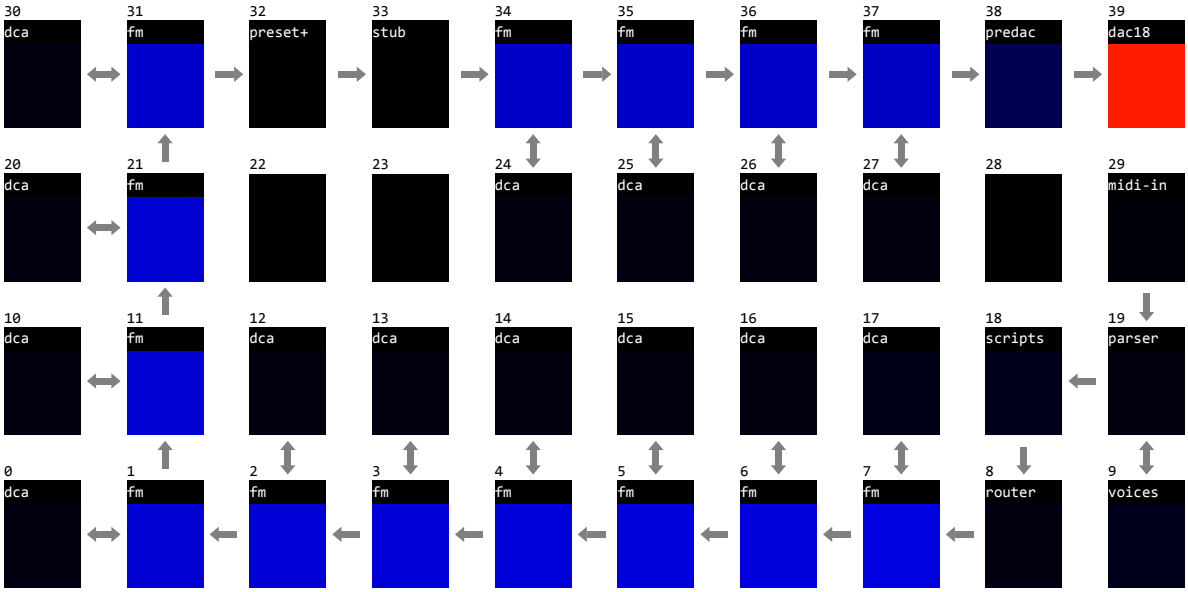

Рис. 6: SEAtar (пресет «epiano»)

(см. рис. 6). В «каркасе» имеются следующие макроузлы: MIDI-драйвер (**midi-in** и **parser**), ЦАП-драйвер (**predac** и **dac18**), таблицы нот и полифонии (**voices**), маршрутизатор сообщений (**router**), загрузчик пресетов (**preset+**).

В макроузле **scripts** хранятся специфичные для пресетов определения программных событий. Эти события в виде **relay**-пакетов отправляются на «голоса» синтезатора в ответ на приходящие MIDI-сообщения.

4.3 FOF

Данный синтезатор голоса в демонстрационном варианте запрограммирован на воспроизведение фразы «seaforth», которая повторяется с различной интонацией. Алгоритм основан на формантном синтезе FOF [20], одном из лучших для имитации пения. Используется быстродействующий вариант данного алгоритма, который был применён в музыкальном синтезаторе FS1R фирмы Yamaha.

Имеется два типа генераторов формант (см. рис. 7): **formant** (FOF-синтез) и **noise** (фильтрованный шум). Данные для этих генераторов поступают из таблицы формант (**formants**) и формантного секвенсера (**events** и **sequencer**).

4.4 Foxtrot

Программная часть слухового аппарата [8] состоит из следующих основных элементов (см. рис. 8).

1. ВЧ фильтрация микрофонных сигналов (**hpmic**), система управления направленностью с подавлением шума (**fdmi**, **rdmi**, **dly**, **sub**, **avg**).

2. Банк из восьми БИХ-фильтров (**sos**, **band1-band8**) с асимметричной АЧХ.

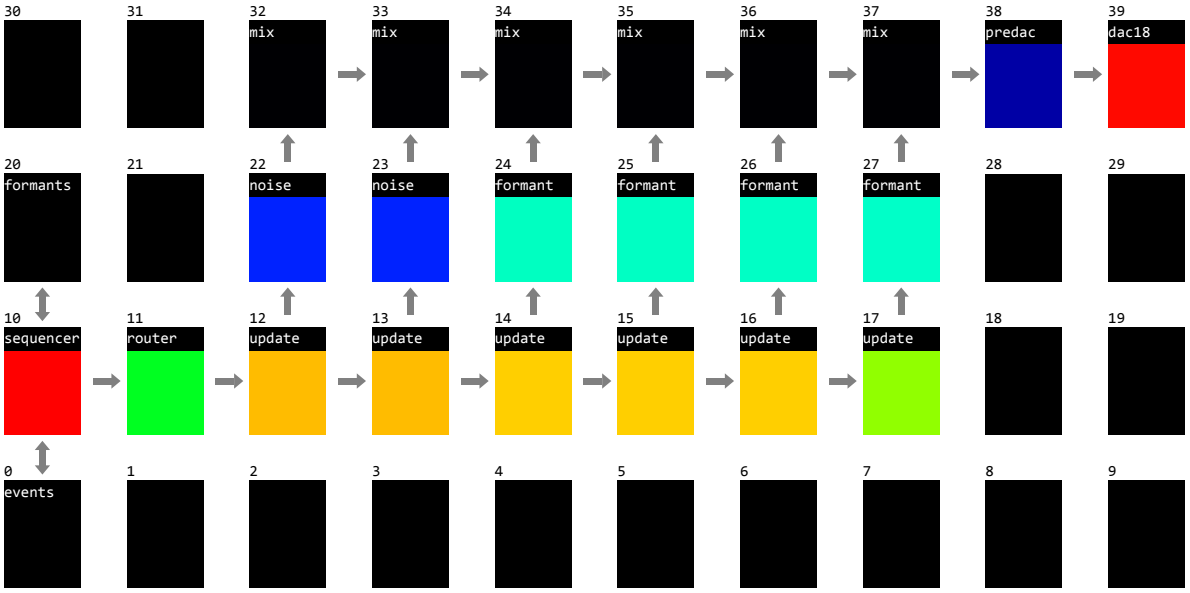

Рис. 7: FOF

3. Многополосный компрессор на основе преобразования Гильберта (**rh**, **re**, **ih**, **im**, **log2**, **exp2**, **div**), сумматор (**sigma**).

Во встроенной в слуховой аппарат флеш-памяти хранятся индивидуальные параметры, такие, как уровень компрессии полос. Эти параметры пользователь может подобрать для себя самостоятельно, с помощью специальной программы на PC.

Проект основывается на исходной Matlab-модели. Также использовалась реализация на языке Си для устройства-прототипа на TMS320C67x DSP фирмы Texas Instruments. В версии для S40 вычисления производятся с 36-битной фиксированной запятой. Для этого применяется режим «расширенной арифметики» с включённым переносом.

Реализация банка фильтров заслуживает отдельного упоминания. На каждую частотную полосу задействовано по два узла. Фильтры работают в параллельном режиме с предварительным распространением очередного звукового семпла по всем полосам. Макроузел **sos** используется для построения фильтров, имеющих до пяти секций второго порядка в первой прямой форме, удобной для вычислений с фиксированной запятой. Управление фильтром осуществляется с помощью интерфейсных слов, передающих коэффициенты вместе с указанием их типа. Полосы содержат до 16 коэффициентов на циклически прокручивающихся стеках. С помощью **sos** построена также ВЧ фильтрация микрофонных сигналов.

Данный проект разрабатывался географически распределёнными участниками. Крупные подсистемы оформлялись в виде библиотек макроузлов. Затем они тестировались на симуляторе обособленно и в различных сочетаниях друг с другом. В результате, на прототипах готового устройства система немедленно заработала в ожидаемом режиме.

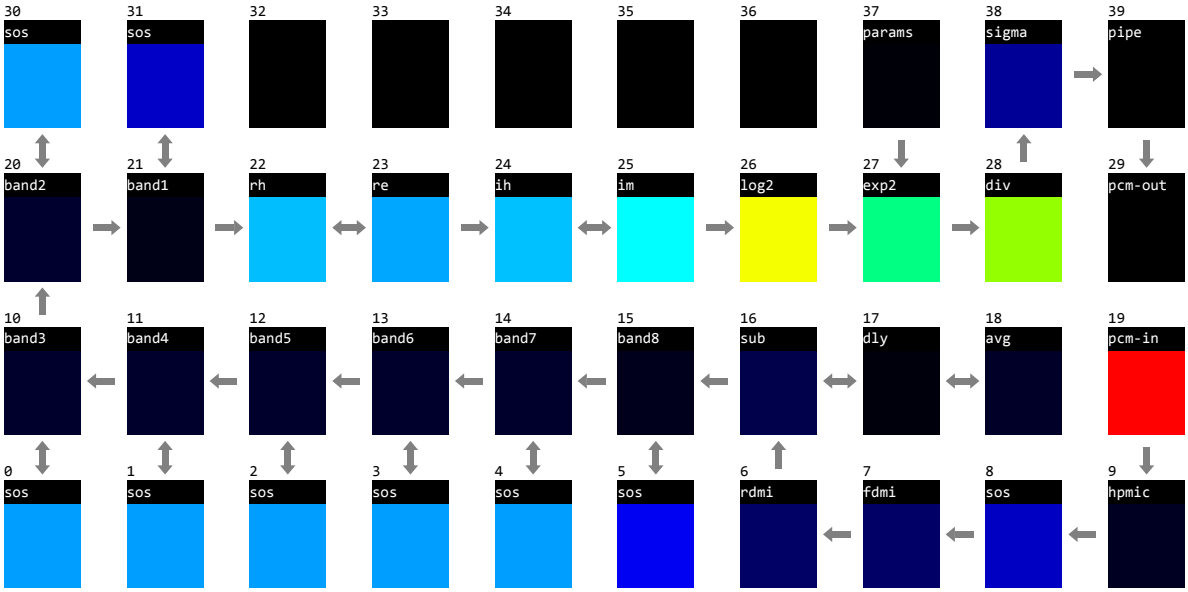

Рис. 8: Foxtrot

5 Заключение

Выражу надежду, что теперь читатель рассматривает SEAforth-архитектуру не как причудливое изобретение без полезных применений. На её основе возможны (и уже созданы) быстродействующие и энергоэффективные решения. Кроме того, эти мультикомпьютеры действительно интересно программировать.

Список литературы

[1] *Броуди, Л.* Начальный курс программирования на языке ФОРТ. — Финансы и статистика, 1990.

[2] *Воеводин В. В., Воеводин Вл. В.*. Параллельные вычисления. — БХВ-Петербург, 2002.

[3] *Евреинов Э. В.* Однородные вычислительные системы, структуры и среды. — Радио и связь, 1981.

[4] *Миллер Р., Боксер Л.* Последовательные и параллельные алгоритмы: Общий подход. — БИНОМ. Лаборатория знаний, 2006.

[5] *Хоар Ч.* Взаимодействующие последовательные процессы. — Мир, 1989.

[6] *Kindermann, Lars.* MusiNum — The Music in the Numbers. `http://reglos.de/musinum`

[7] *Massalin, Henry.* Superoptimizer -- A Look at the Smallest Program. — Proc ACM ASPLOS'87, Sigplan Notices 22,10 (Oct 1987), 122-126.

[8] *Montvelishsky, Michael.* Hearing Aid with SEAforth.
http://seaforth.googlegroups.com/web/Preso-HLS.pdf

[9] Веб-сайт компании IntellaSys.
http://intellasys.net

[10] SEAforth® 40C18 Device Data Sheet.
http://www.intellasys.net/templates/trial/content/S40C18_DataSheet.pdf

[11] Chuck Moore's colorForth, OKAD and S40 Forth Multicomputer chip.
http://colorforth.com

[12] Green Arrays, Inc.
http://greenarraychips.com

[13] VentureForth® S40C18.
http://207.47.34.108/download/index.html

[14] DPANS94.
http://www.taygeta.com/forth/dpans.html

[15] SwiftForth™ Overview.
http://www.forth.com/swiftforth/index.html

[16] Gforth.
http://www.jwdt.com/~paysan/gforth.html

[17] VentureForth® Programmers Guide.
http://www.intellasys.net/templates/trial/content/VF_ProgrammersGuide.pdf

[18] PostScript-подобный язык с выводом в формате SVG. Исходный текст на ANS Forth и некоторые примеры использования.
http://peter.sovietov.com/app/svgscript.zip

[19] The Karplus-Strong Algorithm.
https://ccrma.stanford.edu/~jos/pasp/Karplus_Strong_Algorithm.html

[20] FOF synthesis.
https://ccrma.stanford.edu/~serafin/320/lab3/FOF_synthesis.html