ISSN 0868–5886 НАУЧНОЕ ПРИБОРОСТРОЕНИЕ, 2013, том 23, № 3, с. 114–119

ПРИБОРНО-КОНСТРУКТИВНЫЕ РАЗРАБОТКИ

УДК 621.391.837:681.3

© А. Х. Мурсаев

АППАРАТНАЯ РЕАЛИЗАЦИЯ МЕДИАННОГО ФИЛЬТРА

Предложен аппаратно-ориентированный алгоритм медианной фильтрации и соответствующая структура, обеспечивающие сверхбыстрое определение медианы данных в окне фильтра за счет параллельного исполнения операций сравнения и упорядочения. Предложенный подход обеспечивает экономичную реализацию, существенно ускоряет медианную фильтрацию по сравнению с традиционными решениями и позволяет расширить область применения медианной фильтрации.

Кл. сл.: медианная фильтрация, аппаратная реализация

ВВЕДЕНИЕ

В настоящее время развитие технологии больших интегральных схем позволяет реализовать в составе БИС весьма сложные проблемно-ориентированные блоки для решения конкретных задач обработки данных. Традиционные подходы, основанные на использовании процессорных средств общего назначения, во многих случаях оказываются менее эффективными, чем реализация таких же задач аппаратными средствами. Применение исключительно типовых процессорных средств может привести к избыточным экономическим, энергетическим или массогабаритным показателям [1, 2]. Это прежде всего относится к таким областям применения, как обработка сигналов и изображений в реальном времени, средствам управления производственными процессорами, интеллектуальным измерительным системам и т.д. Аппаратные средства чаще всего используются совместно с процессорным ядром общего назначения, составляя вместе с ним "систему-на-кристалле" (SOC) или "систему-на-плате", и работают как сопроцессоры, "помогающие" центральному процессору системы выполнять части общей задачи, реализация которых в ядре системы по тем или иным соображениям нецелесообразна.

Один из популярных методов предварительной обработки сигналов — медианная фильтрация. Обработка медианным фильтром потока данных, например набора последовательных отсчетов сигналов, эффективно устраняет короткие всплески, включая импульсные помехи в канале передачи, без существенного искажения данных в зонах, в которых всплески отсутствуют. Суть метода состоит в том, что в выбранной последовательности отсчетов (называемой окном наблюдения, или просто окном) ищется медиана, т. е. элемент посредине последовательности упорядоченных по величине отсчетов в окне. Последовательность медианных значений соседних смещенных на один отсчет окон представляет выходную последовательность.

Известно достаточно много алгоритмов, реализующих метод. Обычно в процессе медианной фильтрации значения сигнала в некоторой окрестности точки (т. е. в текущем окне), в которой вычисляется отклик фильтра, сортируются по возрастанию или убыванию. Медиана определяется как значение сигнала соответствующего середине отсортированного ряда. Кроме прямого упорядочения методами последовательных сравнений используют анализ гистограмм распределения и другие приемы [3–5].

Однако большинство из них разработаны для реализации в процессорах традиционной архитектуры с последовательным исполнением операций (команд) и не учитывают особенностей и возможностей аппаратной реализации, в которой многие операции могут исполняться параллельно. Из немногочисленных примеров аппаратных реализаций выделим как наиболее эффективные из известных устройства, описанные в работах [6, 7]. Но первое из них ориентировано на медианную фильтрацию в весьма ограниченном окне, а второе, на наш взгляд, избыточно по объему аппаратуры.

АЛГОРИТМ И БАЗОВАЯ СТРУКТУРА

Предлагаемые в данной статье алгоритм и реализующее его устройство обеспечивают экономичную по аппаратным затратам и сверхпроизводительную реализацию медианной фильтрации непрерывного потока данных. Каждое следующее окно отстает от предыдущего окна на один отсчет.

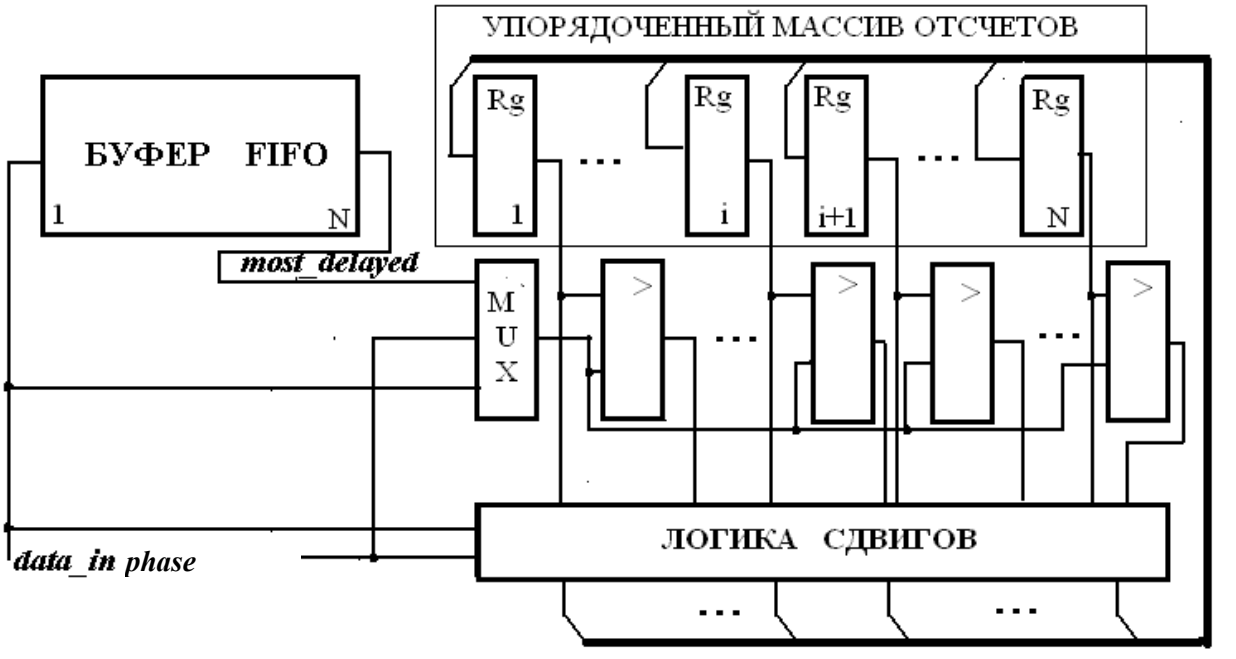

Рис. 1. Вариант структурной организации медианного фильтра (пояснения в тексте)

Очевидно, что при этом не требуется каждый раз выполнять полное упорядочение данных текущего окна. Достаточно лишь немного модифицировать упорядоченный, полученный при обработке предыдущего окна массив, в приведенных программах и далее в тексте называемый `OrA` (***Ordered_Array***). Пусть данные предыдущего окна упорядочены по возрастанию и сохранены в массиве размером в ширину окна (условно примем порядок возрастания "слева направо"). Тогда для формирования упорядоченного массива для нового окна достаточно удалить из предыдущего окна элемент, хранящий отсчет, равный исключаемому элементу окна. При этом содержимое всех элементов в позициях справа от удаляемого переписываются в соседние левые элементы. Отсчет, поступающий в данный момент на вход фильтра, заносится в позицию, хранящую наименьшее большее по сравнению с поступившим отсчетом значение (предыдущее содержимое в этого элемента и содержимое всех элементов справа от него переписываются в соседние правые элементы).

Предлагаемая структура медианного фильтра представлена на рис. 1. Предусмотрено 2 массива данных. Один (`OrA`) в установившемся режиме к концу каждого цикла обработки хранит упорядоченный по возрастанию ряд данных проанализированного окна. Второй хранит тот же набор данных, но упорядоченный по времени поступления, то есть буфер типа FIFO. Наличие этого буфера позволяет легко определить, какой элемент должен быть исключен из упорядоченного массива на очередном шаге — это всегда последний элемент в буфере. Каждый элемент упорядоченного массива соединен со своим компаратором.

Формирование упорядоченного набора данных для следующего окна занимает два шага (в наиболее производительной реализации — 2 такта синхросигнала, линии передачи синхросигнала на рисунке не показаны). В первой фазе (phase=0) на вторые входы всех компараторов через мультиплексор MUX подается наиболее задержанный отсчет из FIFO-буфера, а во второй фазе (phase=1) — новый отсчет со входа.

Описание функционирования упорядочивающего массива представлено фрагментом синтезируемого VHDL-кода (см. Приложение, Код 1). Конкретная схема зависит от используемой схемотехники. Этот блок может быть синтезирован автоматически с помощью любой системы автоматизированного проектирования больших (в том числе программируемых) интегральных схем.

Логика фрагмента соответствует приведенному выше алгоритму с добавлением одной особенности. Вспомогательная переменная `finded` устанавливается в единицу после получение ответа

«не больше» в одном из компараторов и блокирует ответ «больше» от всех последующих компараторов при просмотре «справа налево» по элементам массива (обратим внимание на то, что последовательность просмотра распределена не во времени, а в пространстве). Это потребовалось для учета и при необходимости коррекции ситуаций, когда в исходном массиве OrA есть несколько записей, равных удаляемому значению (сигнал в некотором интервале не изменяется), когда таких записей в OrA нет или массив просто неупорядочен. Два последних случая могут возникнуть при начальном пуске и при сбоях в системе, но массив быстро приходит в рабочее состояние. Медианный фильтр, представленный на рис. 1 был смоделирован и синтезирован (с использованием описания упорядочивающего буфера, приведенного в Приложении, Код 1) в среде проектирования QUARTUS II фирмы ALTERA. В качестве целевой микросхемы задавались микросхемы семейства STRATIX III. Для реализации фильтра 8-разрядной последовательности с длиной окна 7 потребовалось 212 макроячеек, и обеспечивалось функционирование с частотой 164 МГц для микросхем среднего скоростного диапазона. Объем затрат практически линейно зависит от ширины окна и разрядности данных. Допустимая частота не зависит от длины окна и незначительно падает с увеличением разрядности. Последнее объясняется тем, что схема сравнения синтезируются системой QUARTUS с использованием цепей быстрого переноса, характерных для семейства STRATIX III.

ОДНОТАКТНАЯ РЕАЛИЗАЦИЯ ФИЛЬТРА

Дальнейшее повышение производительности в общем случае может осуществляться за счет распараллеливания выполнения операции в наиболее критичных участках исполнения алгоритма. Для описанного выше алгоритма таким критичным является последовательное исполнение двух операций сравнения. За счет некоторого относительно небольшого увеличения оборудования эти операции могут быть разнесены в пространстве и совмещены по времени. Введем две группы параллельных компараторов — в первой группе содержимое всех ячеек массив сравнивается со значением элемента, удаляемого из окна на текущем шаге (обозначим вектор ответов этой группы компараторов как **C1**), а во второй группе — с входным отсчетом, поступающим на вход фильтра в это же время (вектор ответов **C2**). Порядок перестановок и подстановок при переходе к следующему окну иллюстрируется рис. 2. На рисунке отражено содержание упорядочивающего буфера (**OrA**) и схем сравнения перед перестановками. Представлены 2 случая: ***data_in*** > ***most_delayed*** и ***data_in*** < ***most_delayed.*** В случай равенства никаких изменений в **OrA** не производится и пояснений не требуется. В строке **OrA** буквой D отмечены элементы, содержание которых совпадает с удаляемым из окна элементом, а буквой M – элементы массива, содержащие ближайшие бóльшие по отношению к входному отсчету значения. Все прочие элементы массива обозначены как X (не важно). Состояние схем сравнения представлено булевскими векторами **C1** и **C2**, где бит вектора **C1** принимает значение T (TRUE), если соответствующий элемент массива **OrA** превышает удаляемый элемент, и F (FALSE), если не превышает. Подобно алгоритму, описанному выше, предусмотрена блокировка ответов «больше» в разрядах, следующих за первым ответом «меньше» при просмотре «справа налево». Нетрудно видеть, что порядок перестановок зависит от соотношения удаляемого и нового входящего

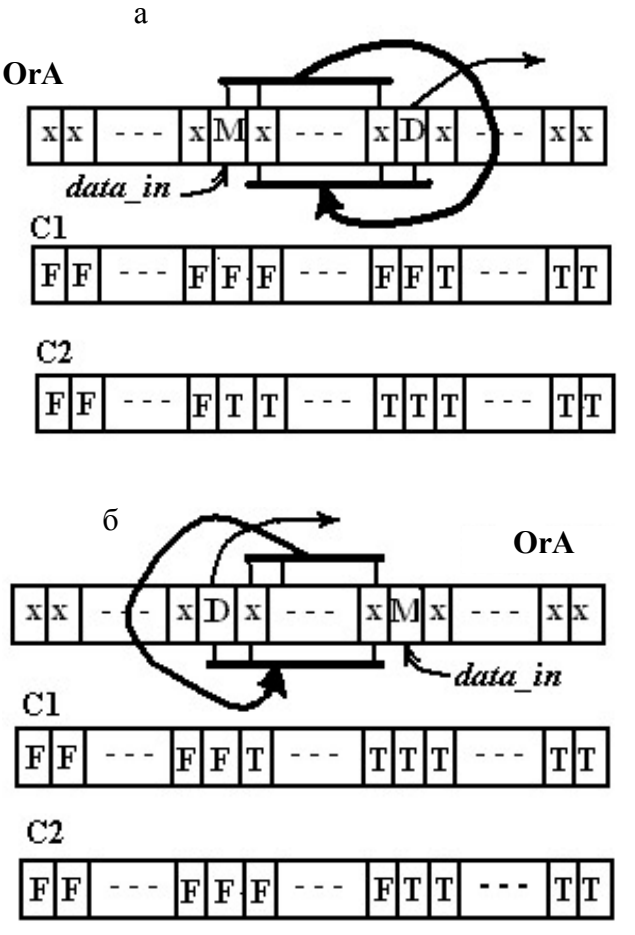

Рис. 2. Порядок перестановок данных в упорядочивающем буфере, реализованном с использованием двух групп компараторов.
а — при ***data_in*** > ***most_delayed***;
б — при ***data_in*** < ***most_delayed***

элементов. Направления перестановок обозначены стрелками и соответствуют одновременному исполнению двух фаз алгоритма, представленного в Приложении, Код 1. Фрагмент VHDL-кода, непосредственно получаемый из рис. 2 и определяющий зависимость выполняемых перестановок и замен от ответов схем сравнения и соотношения удаляемого и входящего сигналов, представлен в Приложении, Код 2.

Медианный фильтр, базирующийся на представленном описании упорядочивающего буфера, а также модулях компаратора и FIFO-буфера из состава библиотеки системы проектирования, был смоделирован, и выполнен его синтез. Как и следовало ожидать, объем оборудования несколько увеличился по сравнению со схемой с одним компаратором (до 256 ячеек ПЛИС семейства STRATIX III для фильтра с окном на 7 восьмиразрядных отсчетов) при сохранении максимальной допустимой частоты. Однако в такой схеме отсчеты могут поступать и соответственно выдаваться на выход в каждом такте, т. е. с удвоенной по сравнению со схемой рис. 1 частотой.

ЗАКЛЮЧЕНИЕ

Описанные реализации не только легко реконфигурируются по длине окна и разрядности данных при реализации в ПЛИС за счет репрограммирования, но и при незначительных дополнениях допускают оперативную (в том числе адаптивную) перестройку параметров в пределах некоторого заранее определенного максимума значений.

Современный уровень развития технологии делает возможной и экономически оправданной реализацию аппаратуры в виде заказных БИС даже при относительно небольших сериях (в пределах до нескольких тысяч экземпляров). Используя приведенные фрагменты VHDL-программ и имеющийся прототип на ПЛИС, можно выполнить разработку заказных схем в кратчайшие сроки и обеспечить производительность до миллиарда отсчетов за секунду, что может удовлетворить требования практически любых современных систем цифровой обработки сигналов.

ПРИЛОЖЕНИЕ

Код 1

```
entity ordered_arr is
   generic (window_length,maxdata:integer);-- длина окна и диапазон значений данных
   port  ( clock,phase: in bit;
   most_delayed,data_in:integer range 0 to maxdata; -- вход и выход FIFO--
   result:out integer range 0 to maxdata);-- центральный элемент массива
end ordered_arr ;

architecture   high_level of ordered_arr  is
         Type my_arr is array (1 to window_length) of integer range 0 to maxdata;
         signal OrA:my_arr; -- регистры упорядывающего массива
         signal compare_input: integer range 0 to maxdata;
begin
   result<=OrA((window_length-1)/2+1);
   compare_input<=most_delayed when phase='0' else data_in; --выход мультиплексора
process(clock)
              variable finded:bit;
   begin    if clock='1' and clock'event then
          finded:='0';
      IF PHASE='0' THEN
           for i in window_length downto 2 loop -- удаление элемента
            if (finded='0') and (Ora(i)>compare_input) then  OrA(i-1)<=Ora(i);
                else  finded:='1';
                end if;
             end loop;
                OrA(window_length)<=maxdata;
      else – phase='1'
             for i in window_length downto 1 loop --вставка нового отсчета
              if i/=1 then
                if (finded='0')  then
```

```
                    if (OrA(i)>compare_input) and  (  OrA(i-1)>compare_input)
                       then   OrA(i)<=OrA(i-1);
                    else OrA(i)<=data_in;  finded:='1';
                    end if;
                end if;--finded
               else --i=1
                 if (finded='0')  then OrA(i)<=data_in;…
                 end if;--finded
            end if;  --i/=1
         end loop;
      end if;  --phase
   end if;  --clock
end process;
 end high_level;
```

Код 2

```
process(clock)
   begin
    if clock='1' and clock'event then
        if data_in=most_delayed then null;
        elsif  data_in>most_delayed then
           for i in window_length-1 downto 1 loop
                 if C1(i+1) and not C2(i+1)  then OrA(i)<=OrA(i+1);
              elsif     not C2(i) and  C2(i+1)  then OrA(i)<=data_in;
             end if;
           end loop;
           if  not C2(window_length) then OrA(window_length)<=data_in;
             end if;
     else --  data_in<most_delayed
            for i in window_length downto 2 loop
                        if not C1(i-1) and  C2(i-1)  and not comp1(i) then
                                                     OrA(i)<=OrA(i-1);
                        elsif C2(i) and   not C2(i-1)  then OrA(i)<=data_in;
                       end if;
           end loop;
             if  C2(1) then OrA(1)<=data_in;
              end if;
     end if;-- d_in=most_delayed
     end if;  --clock
end process;
```

СПИСОК ЛИТЕРАТУРЫ

1. *Kaeslin H.* Digital integrated circuit design from VLSI architectures to CMOS fabrication. N.Y.: Cambridge University Press, 2008. 835 p.
2. *Грушвицкий Р.И., Мурсаев А.Х., Угрюмов Е.П.* Проектирование систем на микросхемах с программируемой структурой. СПб.: БХВ-Петербург, 2006. 736 с.
3. *Гонсалес Р., Вудс Р.* Цифровая обработка изображений. М.: Техносфера, 2005. 1072 с.
4. *Бардин Б.В.* Быстрый алгоритм медианной фильтрации // Научное приборостроение. 2011, № 3. С. 135–139.
5. *Мушкарев С.В.* Реализация ранжирующих и медианных фильтров на процессоре NM6403 (Л1879ВМ1). URL: (www.module.ru/files/papers-cos012005/pdf).
6. *Воробьев Н.* Одномерный цифровой медианный фильтр с трехотсчетным окном // CHIPNEWS. 1999. № 8. URL: (www.chipinfo.ru/literature/chipnews/199908/29.html).
7. *Переверзев А.Л.* Одномерный медианный фильтр с модульной архитектурой. Патент РФ 2362209 c1, дата отсчета срока действия патента 06.12.2007. 4 с.

Санкт-Петербургский государственный электротехнический университет "ЛЭТИ"

Контакты: *Мурсаев Александр Хафизович*,
sashamursaev@yandex.ru

Материал поступил в редакцию 12.02.2013

HARDWARE IMPLEMENTATION OF MEDIAN FILTERING

A. Kh. Mursaev

Saint-Petersburg Electrotechnical University "LETI"

A hardware-oriented algorithm of median filtering and structures to realize it are presented. The solution enables ultra-fast definition of data median inside oserved window due to parallel execution of comparisons and ordering. The approach provides a cost-effective implementation, significantly accelerates the median filtering and expands the scope of the method.

Keywords: median filtering, hardware implementation