ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2022 Управление, вычислительная техника и информатика № 59

Tomsk State University Journal of Control and Computer Science

Научная статья
УДК 621.382
doi: 10.17223/19988605/59/12

Использование синтеза высокого уровня для реализации на ПЛИС адаптивных LMS-фильтров, работающих в формате с плавающей запятой

Инна Владимировна Ушенина

Пензенский государственный технологический университет, Пенза, Россия, ivl23@yandex.ru

Аннотация. Рассмотрена возможность использования синтеза высокого уровня для реализации на ПЛИС адаптивных фильтров наименьших квадратов (LMS-фильтров), работающих в формате с плавающей запятой. Разработка адаптивных фильтров не автоматизирована в средах проектирования устройств на ПЛИС и является трудоемким и длительным процессом, а синтез высокого уровня позволяет сэкономить время – и за счет описания фильтра на высоком уровне абстракции, и за счет возможности быстрой смены настроек, влияющих на архитектуру фильтра. В статье представлены исходный код, описывающий алгоритм LMS, настройки и результаты синтеза высокого уровня – два варианта описания LMS-фильтра, готовые к реализации на ПЛИС. Показано, что эффективной мерой сокращения времени выполнения LMS-алгоритма является конвейеризация цикла вычислений.

Ключевые слова: синтез высокого уровня; ПЛИС; LMS-фильтр; формат с плавающей запятой; директивы синтеза

***Для цитирования*:** Ушенина И.В. Использование синтеза высокого уровня для реализации на ПЛИС адаптивных LMS-фильтров, работающих в формате с плавающей запятой // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2022. № 59. С. 108–116. doi: 10.17223/19988605/59/12

Original article
doi: 10.17223/19988605/59/12

FPGA implementation of floating-point LMS adaptive filters using high-level synthesis

Inna V. Ushenina

Penza State Technological University, Penza, Russian Federation, ivl23@yandex.ru

Abstract. This paper considers how to use high-level synthesis (HLS) tools for FPGA implementation of floating-point least mean square (LMS) adaptive filters. The essence of high-level synthesis is to transform a source code written in a programming language like C or C++ to an RTL code which is later used for creating an IP module. High-level synthesis provides a possibility to concentrate on the algorithm and architecture of a scheme being developed, while FPGA resources and their connectivity are determined automatically. This is especially important when developing adaptive filters on FPGAs because the automated FPGA design of adaptive filters is not supported in IDEs just as it is done for FIR filters.

In this paper, the source code of the LMS adaptive algorithm is written in C++ language. In the main function of the source code, input, output, and internal variables are presented as single-precision, floating-point numbers in line with IEEE 754 standard. Samples of the input signal and desired signal are the input variables, while samples of the output signal and error signal are the output variables. During a high-level synthesis procedure, input and output variables of the main function are transformed into input and output ports of the IP module being synthesized.

HLS is carried out under the control of synthesis settings and directives formed separately from the source code. A combination of the source code with different synthesis settings and (or) directives leads to different results, each of which is called a project solution. In this paper, the same source code of the LMS adaptive algorithm is used

to create two solutions of the LMS adaptive filter under the control of two different sets of synthesis settings and directives. Within each solution, the LMS filters were synthesized of the order ranging from 32 to 512 coefficients. For each set of synthesis settings, the minimum clock frequency was set equal to 100 MHz. The other synthesis settings and directives were left default for the first solution, while for the second one, the loop of operations contained in the source code was pipelined. That is, in the first LMS filter solution, the next loop iteration starts after the previous one has ended. In the second solution, the next iteration starts five cycles after the previous one, whereas in both solutions the whole iteration needs twenty clock cycles. In the LMS algorithm, the loop contains the major part of calculations, including filter coefficients updating and convolution between the input signal and the current impulse response of the filter. Therefore, pipelining of the loop has enabled the latency of the LMS algorithm to be significantly reduced, namely by more than 70%.

It has been shown that for both solutions, operations are scheduled into clock cycles similarly, and the basic operations such as multiplication, addition, and RAM reading and writing are performed in the same number of cycles. The order of operations is, in general, the same for both solutions.

Besides timing characteristics, resource utilization has been assessed for both solutions. It has been shown that the loop pipelining affects the resource utilization insignificantly: the FF and LUT utilization increases by not more than 31% and 10%, respectively. The increased utilization of FFs and LUTs in the second solution is caused by the more complicated control logic and by the storage of more intermediate results of calculations. The utilization of DSP blocks and block RAMs is the same for both solutions.

It has also been shown that when increasing the filter order, the utilization of memory grows up in proportion to it, but the utilization of DSP blocks remains unchanged, and the utilization of FFs and LUTs increases very little.

High-level synthesis was carried out in Vivado HLS 2018.3. The LMS filter IP modules are intended to be implemented on Xilinx XC7A200tfbg-1 FPGA.

Keywords: high-level synthesis; FPGA; LMS filter; floating point format; synthesis directives

***For citation*:** Ushenina, I.V. (2022) FPGA implementation of floating-point LMS adaptive filters using high-level synthesis . *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitelnaja tehnika i informatika – Tomsk State University Journal of Control and Computer Science*. 59. pp. 108–116. doi: 10.17223/19988605/59/12

Адаптивные фильтры с алгоритмом наименьших квадратов (LMS) применяются для идентификации систем, эхокомпенсации в телефонных сетях, выравнивания характеристик электрических каналов связи, адаптивного формирования диаграмм направленности в антенных решетках, подавления шумов, отслеживания объектов и др. Алгоритм LMS надежен, сравнительно прост в технической реализации и имеет большое количество модификаций, учитывающих специфику решаемых задач [1–4]. Для эффективной работы LMS-фильтра предпочтителен формат с плавающей запятой из-за более высокой скорости сходимости и значительно меньшей мощности остаточного сигнала ошибки [5, 6].

Для аппаратной реализации адаптивных фильтров, работающих в формате с плавающей запятой, в целом более приспособлены сигнальные процессоры. ПЛИС представляют интерес, когда сигнальные процессоры не могут обеспечить достаточную производительность. Высокая производительность требуется, например, в устройствах отслеживания источника звука в условиях реверберации [7–10], слепой идентификации каналов [11], улучшения качества звукозаписи и звуковоспроизведения [12], где адаптивные фильтры должны быть многоканальными.

Возможности современных ПЛИС (FPGA) позволяют использовать их для решения широкого круга задач: цифровой обработки сигналов [6, 13–18], высокопроизводительных вычислений [19, 20], машинного обучения и реализации нейронных сетей [21] и др. Тем не менее выбор ПЛИС для реализации LMS-фильтра, работающего в формате с плавающей запятой, сопряжен с рядом проблем. Так, встроенными аппаратными блоками для вычислений в формате с плавающей запятой располагают очень немногие ПЛИС. В остальных случаях арифметические устройства нужно собирать из блоков цифровой обработки сигналов, ориентированных на формат с фиксированной запятой, и (или) программируемой логики общего назначения. При этом разработка адаптивных фильтров в средах проектирования устройств на ПЛИС не автоматизирована, т.е. не предусмотрены соответствующие модули интеллектуальной собственности (IP-модули), графический пользовательский интерфейс и т.п.

В литературе в основном описаны варианты реализации на ПЛИС LMS-фильтров, работающих в формате с фиксированной запятой (см. напр.: [6, 13–17]). Описания LMS-фильтров в этих работах выполнены на языке VHDL или Verilog – как вручную [17], так и автоматически, при использовании IP-модулей сумматоров и умножителей [6, 13] или инструмента System Generator, встраиваемого в MATLAB [14–16]. Реализация LMS-фильтров, выполняющих вычисления над числами в формате

с плавающей запятой, описывается в литературе реже. В [6] сравниваются результаты реализации 32-разрядного LMS-фильтра в форматах с фиксированной и плавающей запятой. Для описания фильтра, работающего в формате с фиксированной запятой, в [6] использован VHDL-пакет fixed-point package; для реализации фильтра, работающего в формате с плавающей запятой, – IP-модули сумматора и умножителя. В [18] описан на языке Verilog и реализован на ПЛИС вычислительный модуль, работающий в формате с плавающей запятой и включающий в себя LMS-фильтр.

В настоящей работе рассмотрена возможность использования синтеза высокого уровня для реализации на ПЛИС LMS-фильтров, работающих в формате с плавающей запятой. Суть синтеза высокого уровня (High-Level Synthesis; HLS) заключается в преобразовании исходного кода, написанного на языке программирования (например, C или C++), в описание устройства на уровне регистровых передач, т.е. на языке VHDL или Verilog. На языке программирования описывается только алгоритм, без уточнения деталей его реализации. Один и тот же исходный код можно реализовать с различным набором настроек и директив и получить различные результаты, называемые решениями HLS-проекта [22]. То есть можно в короткий срок получить описания устройств, работающих по одному и тому же алгоритму, но имеющих различную архитектуру, производительность и т.д. Для встраивания того или иного решения в проект, реализуемый на ПЛИС, требуется преобразовать это решение в IP-модуль.

В данной работе из исходного кода на языке C++ получены и проанализированы два решения LMS-фильтра: с последовательным выполнением всех операций, предусмотренных алгоритмом LMS, и с конвейеризацией цикла операций, содержащегося в алгоритме LMS. В цикле выполняется основная часть операций алгоритма LMS, поэтому конвейеризация позволила существенно сократить время его выполнения (латентность). В рамках каждого решения получены результаты синтеза описаний фильтров с порядком от 32 до 512 при заданной тактовой частоте 100 МГц. Полученные решения оценены по критериям ресурсоемкости, латентности и распределения операций алгоритма во времени. Для синтеза высокого уровня использован синтезатор Vivado HLS 2018.3 от фирмы Xilinx [22]. Полученные IP-модули LMS-фильтра предназначены для ПЛИС XC7A200tfbg-1 от Xilinx.

1. LMS-фильтр и описание его работы на языке C++

1.1. LMS-фильтр

На каждом шаге дискретизации LMS-фильтр принимает по одному отсчету входного сигнала – $x[n]$, и требуемого сигнала – $d[n]$, и формирует отсчеты выходного сигнала и сигнала ошибки – $y[n]$ и $e[n]$ соответственно [1].

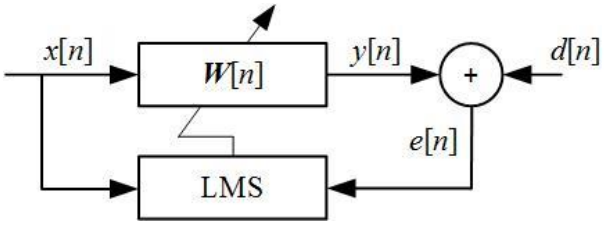

Рис. 1. Структурная схема LMS-фильтра
Fig. 1. LMS filter block diagram

Как показано на рис. 1, LMS-фильтр содержит КИХ-фильтр с коэффициентами $\boldsymbol{W}[n]$, обрабатывающий отсчеты входного сигнала согласно формуле (1), и блок адаптации LMS, где происходит перерасчет коэффициентов КИХ-фильтра по формуле (2):

$$y[n] = \sum_{i=0}^{N-1} w[i] \cdot x[n-i], \tag{1}$$

$$\boldsymbol{W}[n+1] = \boldsymbol{W}[n] + \mu e[n]\boldsymbol{X}[n]. \tag{2}$$

В (1) n – номер шага дискретизации, N – порядок фильтра, $w[i]$ – коэффициенты фильтра, образующие вектор $\boldsymbol{W}[n]$, $x[n-i]$ – отсчеты входного сигнала, образующие вектор $\boldsymbol{X}[n]$. В (2) $\boldsymbol{W}[n+1]$ и $\boldsymbol{W}[n]$ – векторы коэффициентов фильтра, действующие на $[n+1]$-м и n-м шагах дискретизации, $e[n]$ – сигнал ошибки, т.е. разность $d[n]$ и $y[n]$, $\boldsymbol{X}[n]$ – вектор отсчетов входного сигнала, μ – параметр, определяющий скорость сходимости алгоритма и выбираемый с учетом максимального собственного значения корреляционной матрицы входного сигнала λ_{max} (3):

$$0 < \mu < \frac{1}{\lambda_{max}}. \qquad (3)$$

1.2. Исходный код

Для выполнения синтеза высокого уровня одна из функций исходного кода должна быть объявлена главной, чтобы ее входные и выходные переменные были преобразованы во входные и выходные порты устройства. Разработанный здесь исходный код содержит всего одну функцию, входными переменными которой являются отсчет входного сигнала *x* и отсчет требуемого сигнала *d*, а выходными переменными – отсчет выходного сигнала *y* и отсчет сигнала ошибки *e*.

Внутри функции также объявлены:

1) целочисленная переменная *i*, служащая счетчиком в цикле операций;

2) массив отсчетов входного сигнала *x_array*;

3) массив коэффициентов фильтра *w_array*;

4) переменная *curr_w* для временного хранения коэффициента, с которым функция работает на *i*-й итерации цикла, обновляя этот коэффициент и умножая его на отсчет входного сигнала;

5) переменная *curr_x* для временного хранения отсчета входного сигнала, с которым функция работает на *i*-й итерации цикла, умножая его на коэффициент фильтра;

6) переменная *mue* для хранения произведения отсчета сигнала ошибки и шага сходимости;

7) переменная *acc* для временного хранения суммы произведений отсчетов входного сигнала на коэффициенты фильтра, накопленной за уже выполненные итерации цикла.

За исключением *i*, все перечисленные выше переменные и элементы массивов имеют тип float, что соответствует стандарту IEEE 754 для чисел одинарной точности.

После объявления массивов и переменных функция переходит к выполнению цикла с количеством итераций, равным порядку фильтра *N* (рис. 2). С каждой итерацией переменная *i* меняется от $N-1$ до 0.

```
1   loop: for (i=N-1; i>=0; i--)
2    {if (i == 0)
3      {curr_x = x; x_array[0] = x;}
4     else {curr_x = x_array[i-1];
5     if (i != (N-1))
6   {x_array[i] = x_array[i-1];}}
7   curr_w = w_array[i];
8   curr_w += mue*curr_x;
9        w_array[i]=curr_w;
10  acc += curr_w*curr_x; }
```

Рис. 2. Цикл в составе исходного кода
Fig.2 Loop in the source code

На каждой итерации цикла сначала выполняется перемещение одного из отсчетов входного сигнала в массиве *x_array*. После выполнения *N* итераций самый старый отсчет отбрасывается, остальные отсчеты сдвигаются в массиве с приращением индекса на единицу, а на освободившееся место *x_array*[0] помещается новый отсчет входного сигнала. Переменной *curr_x* на каждой итерации приравнивается один из отсчетов входного сигнала.

Затем из массива *w_array* в переменную *curr_w* считывается коэффициент с индексом *i*, и выполняется его перерасчет с использованием переменных *curr_x* и *mue*. После перерасчета коэффициент *w_array*[*i*] обновляется.

Последней операцией в цикле является вычисление текущей суммы произведений *acc*, т.е. добавление к текущему значению *acc* произведения переменных *curr_x* и *curr_w*. Таким образом, на каждом шаге дискретизации коэффициенты фильтров будут использоваться сразу после обновления.

После выхода из цикла отсчет выходного сигнала *y* приравнивается к переменной *acc*; вычисляются отсчет сигнала ошибки *e* и его произведение с шагом сходимости *mue*.

2. Методы исследования

Разработка исходного кода, указание настроек и директив, а также собственно синтез высокого уровня выполнены в среде Vivado HLS 2018.3. В рамках каждого из решений синтез высокого уровня был проведен при варьировании порядка фильтра *N* от 32 до 512 коэффициентов. Значение *N* указывалось в исходном коде. В настройках синтеза для обоих решений были указаны минимально допустимая тактовая частота 100 МГц и целевой кристалл – XC7A200tfbg-1.

Во втором решении для конвейеризации цикла (см. рис. 2) на него была наложена директива “pipeline” с указанием интервала инициации (ii), т.е. количества тактов с момента начала итерации цикла, спустя которое запускается следующая итерация [22]. Был указан минимально возможный для данного исходного кода интервал инициации – 5 тактов.

После выполнения синтеза высокого уровня средой Vivado HLS 2018.3 формировались отчеты, из которых для каждого решения и каждого значения *N* получены:

1) максимальная тактовая частота фильтра (*Fmax*);

2) латентность функции, т.е. количество тактов, требуемое на ее выполнение (*FL*);

3) латентность одной итерации цикла, т.е. количество тактов, требуемое на ее выполнение (*IL*);

4) график распределения операций функции по тактам;

5) ресурсоемкость по таким видам ресурсов, как D-триггеры (FF), табличные преобразователи (LUT), блоки цифровой обработки сигналов (DSP) и блоки ОЗУ (BRAM).

3. Результаты синтеза высокого уровня и их обсуждение

3.1. Оценка ресурсоемкости LMS-фильтров

В табл. 1 и 2 приведены результаты оценки максимальной тактовой частоты, ресурсоемкости и латентности для двух решений LMS-фильтра. Требование к минимальной тактовой частоте выполнено для обоих решений при всех значениях *N*.

Ресурсоемкость фильтров с увеличением *N* меняется незначительно как при конвейеризации цикла, так и при ее отсутствии. Так, в обоих случаях увеличение *N* не приводит к увеличению расхода блоков DSP. Из отчетов о синтезе высокого уровня следует, что два из пяти блоков DSP используются для реализации сумматора, еще три – для реализации умножителя. Повышение *N*, разумеется, приводит к увеличению объема памяти, требуемого для хранения массивов *x_array* и *w_array*. При $N = 32$ для хранения каждого из массивов требуется по 1 024 бита, при $N = 64$ – по 2 048 бит, и т.д. Тем не менее при $N \leq 512$ для хранения массивов требуется не более двух блоков ОЗУ. Незначительное увеличение расхода табличных преобразователей с повышением *N* для обоих решений связано с декрементом счетчика цикла *i* и сравнением *i* с нулем и величиной $N - 1$, поскольку с увеличением *N* эти операции требуют большей разрядности сумматора и компараторов. Увеличение расхода D-триггеров для обоих решений связано с увеличением разрядности шины адреса блока памяти коэффициентов фильтра, а также с увеличением разрядности *i*. В остальном структура и ресурсоемкость фильтров, полученных в рамках обоих решений, не зависят от *N*.

Таблица 1

Результаты синтеза LMS-фильтра при отсутствии директив (решение 1)

N	*Fmax*, МГц	LUT	FF	DSP	BRAM	*FL/IL*
32	121	693	683	5	2	649/20
64	121	693	686	5	2	1 289/20
128	121	693	689	5	2	2 569/20
256	121	698	692	5	2	5 129/20
512	121	699	695	5	2	10 249/20

Таблица 2

Результаты синтеза LMS-фильтра при конвейеризации цикла (решение 2)

N	*Fmax*, МГц	LUT	FF	DSP	BRAM	*FL/IL*
32	121	764	898	5	2	184/20
64	121	764	901	5	2	344/20
128	121	764	904	5	2	664/20
256	121	769	907	5	2	1 304/20
512	121	770	910	5	2	2 584/20

Конвейеризация цикла приводит к повышению расхода триггеров не более чем на 31%, а табличных преобразователей – не более чем на 10%. Повышение ресурсоемкости связано с тем, что конвейеризация требует более сложного управляющего автомата и большего количества ресурсов для формирования входных сигналов сумматора, умножителя и блоков памяти, а также для сохранения промежуточных результатов фильтрации.

3.2. Распределение операций по тактам

На рис. 3 и 4 приведены графики распределения операций по тактам, сгенерированные средой Vivado HLS 2018.3 при отсутствии (см. рис. 3) и наличии (см. рис. 4) конвейеризации цикла. Номера тактов, начиная с нуля, приведены в верхней части рисунков. Такты с 1-го по 20-й, приходящиеся на циклы (loop), сгруппированы по выполняемым операциям. Операции, выполняемые в каждой группе тактов, обозначены цифрами, которые соответствуют номерам строк во фрагменте исходного кода на рис. 2. При конвейеризации цикла очередная итерация запускается спустя 5 тактов предыдущей итерации; каждый старт новой итерации обозначен на рис. 4 вертикальной жирной чертой. Тактам, следующим после выхода из цикла, условно присвоены номера, начиная с 21.

Распределение операций по тактам выполнено для обоих решений похожим образом. До вхождения в цикл на такте 0 осуществляются прием *x* и *d* и их размещение в памяти, а также считывание переменной *mue*, вычисленной на предыдущем шаге дискретизации.

В цикле в течение тактов 1 и 2 происходят перемещение одного из отсчетов входного сигнала в массиве *x_array* и перезапись переменной *curr_x*. При этом выполняются вспомогательные операции с переменной *i*: сравнение с 0 и *N* – 1, а в первом решении – также и декремент. При конвейеризированном цикле в тактах 1 и 2 выполняется также считывание *i*-го коэффициента фильтра из массива *w_array* в переменную *curr_w*.

С 3-го по 6-й такты выполняется умножение переменной *curr_x* на переменную *mue*. В первом решении на 5-м и 6-м тактах происходит считывание *i*-го коэффициента фильтра из массива *w_array* в переменную *curr_w*. Во втором решении на 5-м такте выполняется декремент счетчика *i*.

С 7-го по 11-й такты выполняется сложение переменной *curr_w* и полученного на 6-м такте произведения *curr_x***mue*; результат сохраняется в переменную *curr_w*. На 12-м такте *i*-й элемент массива *w_array* приравнивается к переменной *curr_w*. С 12-го по 15-й такт вычисляется произведение переменных *curr_x* и *curr_w*.

С 16-го по 20-й такт выполняется сложение только что вычисленного произведения *curr_x* и *curr_w* и накопленной к данному моменту суммы произведений коэффициентов фильтра на отсчеты входного сигнала, хранящейся в переменной *acc*.

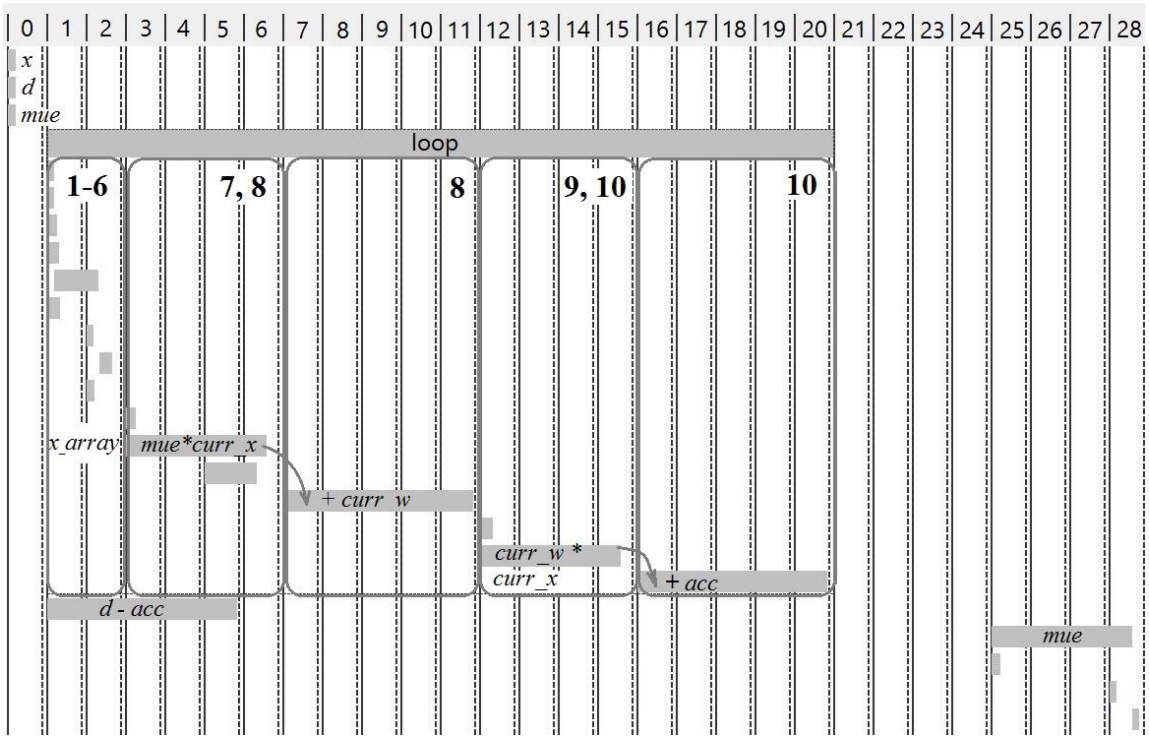

Рис. 3. Распределение операций по тактам при последовательном выполнении итераций цикла (решение 1)
Fig. 3. Scheduling the operations into clock cycles under condition of the loop iterations are performed sequentially (solution 1)

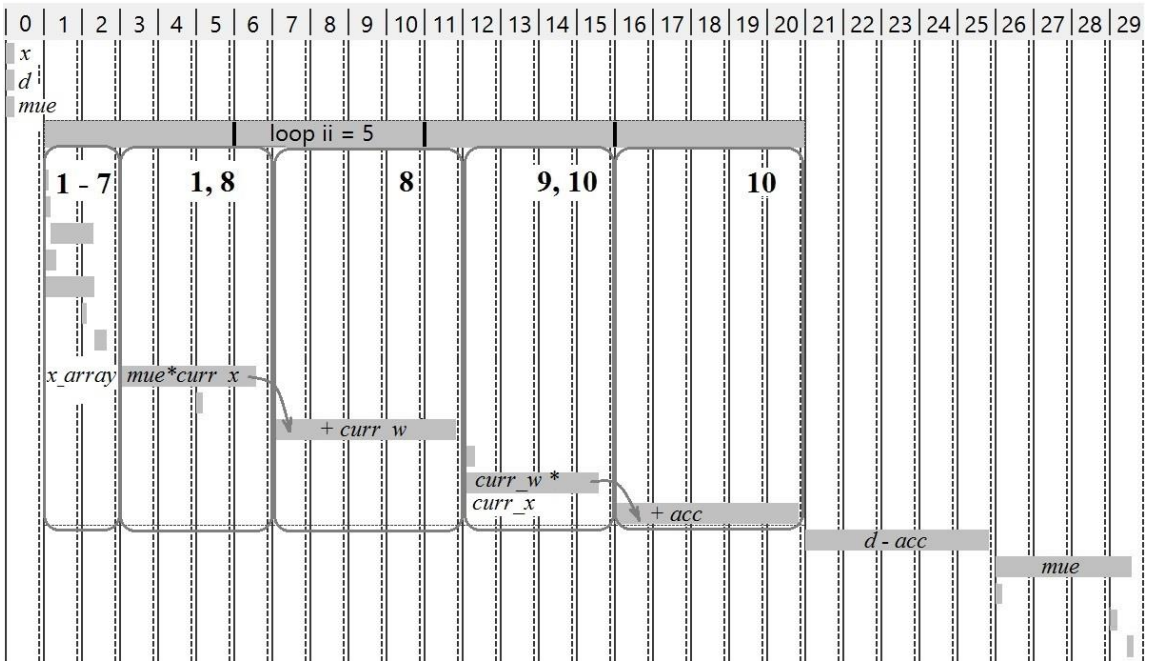

Рис. 4. Распределение операций по тактам при конвейеризации цикла (решение 2)
Fig. 4. Scheduling the operations into clock cycles under condition of the loop is pipelined (solution 2)

После выхода из цикла выходной переменной *y* присваивается значение переменной *acc*. В течение пяти тактов рассчитывается *e* как разность между *d* и *acc*. Заметим, что на рис. 3 такты 21–24 не заполнены. Это связано с работой управляющего автомата в решении 1: после каждой итерации цикла автомат переходит к проверке *i*, что соответствует такту 1. Если $i < 0$, т.е. все итерации выполнены, автомат переходит к состояниям, соответствующим расчету *e*. Этим состояниям на рис. 3 фактически соответствуют такты 21–24, хотя они показаны под тактами 2–5.

Еще четыре такта требуется на расчет переменной *mue*, которая будет использоваться на следующем шаге дискретизации. Одновременно с расчетом *mue* выходные переменные *e* и *y* выдаются на одноименные порты фильтра.

Из табл. 1 и 2, рис. 3 и 4 следует, что при *N* от 32 до 512 за счет конвейеризации цикла удается сократить латентность функции более чем на 70%.

Заключение

С помощью инструментов синтеза высокого уровня удалось сформировать и оценить два решения LMS-фильтра с порядком от 32 до 512, отличающихся организацией предусмотренного алгоритмом LMS цикла операций. Конвейеризация цикла оказалась эффективной мерой сокращения времени выполнения LMS-алгоритма. На ресурсоемкость фильтра конвейеризация цикла влияет незначительно.

Список источников

1. Уидроу Б., Стирнз С. Адаптивная обработка сигналов : пер. с англ. М. : Радио и связь, 1989. 440 с.
2. Farhang-Boroujeny B. Adaptive Filters: Theory and Applications. New York : John Wiley & Sons, 2013. 802 p.
3. Elliott S. Signal Processing for Active Control. London : Academic Press, 2001. 517 p.
4. Glentis G.O., Berberidis K., Theodoridis S. Efficient least squares adaptive algorithms for FIR transversal filtering // IEEE Signal Processing Magazine. 1999. V. 16 (4). P. 13–41.
5. Caraiscos C., Liu B. A roundoff error analysis of the LMS adaptive algorithm // IEEE Transactions on Acoustics, Speech, and Signal Processing. 1984. V. 32 (1). P. 34–41.
6. Shashikala P., Renjith Kumar T.G., Subramani H. An FPGA implementation of the LMS adaptive filter for active vibration control // International Journal of Research in Engineering and Technology. 2013. V. 2 (10). P. 1–10.
7. Wang L., Zhang Z., Kai A. Hands-free speaker identification based on spectral subtraction using a multi-channel least mean square approach // Acoustics, Speech and Signal Processing : Proc. of the IEEE International Conference. 2013. P. 7224–7228.
8. Wang L., Kitaoka N., Nakagawa S. Distant-talking speech recognition based on spectral subtraction by multi-channel LMS algorithm // IEICE Transactions on Information and Systems. 2011. V. 94 (3). P. 659–667.
9. Antonacci F., Lonoce D., Motta M., Sarti A., Tubaro S. Efficient source localization and tracking in reverberant environments using microphone arrays // Acoustics, Speech, and Signal Processing : Proc. of the IEEE International Conference. 2005. V. 4. P. iv/1061–iv/1064.
10. Benesty J., Huang Y. Adaptive Signal Processing: Applications to Real-World Problems. Springer Science & Business Media, 2013. 365 p.
11. Huang Y.A., Benesty J. Adaptive multi-channel least mean square and Newton algorithms for blind channel identification // Signal Processing. 2002. V. 82 (8). P. 1127–1138.
12. Nelson P.A., Orduna-Bustamante F., Hamada H. Multi-channel signal processing techniques in the reproduction of sound // Journal of the Audio Engineering Society. 1992. V. 44 (11). P. 973–989.
13. Yaqin W., Xuebin L., Bingliang H. Implementation of a LMS filter on FPGA employing extremeDSP and smart IP-core design // Electronic Measurement & Instruments : Proc. of the 10th International Conference. 2011. P. 341–345.
14. Bahoura M., Ezzaidi H. FPGA-implementation of parallel and sequential architectures for adaptive noise cancelation // Circuits, Systems, and Signal Processing. 2011. V. 30 (6). P. 1521–1548.
15. Liu Y., Ge S., Xing J., Cui Z., Meng J. FPGA implementation of high-throughput, low-latency complex LMS algorithm // Signal, Information and Data Processing : Proc. of the IEEE International Conference. 2019. P. 1–4.
16. Liwei C., Zhiliang T., Lidong C. LMS algorithm fixed-point modeling designated based on System Generator modules // Environmental Electromagnetics : Proc. of the 7th Asia-Pacific Conference. 2015. P. 152–156.
17. Ушенина И.В. Реализация алгоритмов цифровой адаптивной фильтрации на ПЛИС // XXI век: итоги прошлого и проблемы настоящего плюс. 2012. № 5 (09). С. 134–138.
18. Vasudeva B., Deora P., Pradhan P.M., Dasgupta S. Efficient implementation of LMS adaptive filter-based FECG extraction on an FPGA // Healthcare Technology Letters. 2020. V. 7 (5). P. 125–131.
19. Martyshkin A.I., Salnikov I.I. Hardware memory buffer module for multiprocessor system // Ad Alta: Journal of Interdisciplinary Research. 2018. V. 8 (1). P. 304–308.
20. Martyshkin A.I., Salnikov I.I., Pashchenko D.V., Trokoz D.A. Associative co-processor on the basis of programmable logical integrated circuits for special purpose computer systems // Proc. of the 2018 Global Smart Industry Conference. 2018. P. 1–5.
21. Mittal S. A survey of FPGA-based accelerators for convolutional neural networks // Neural computing and applications. 2020. V. 32 (4). P. 1109–1139.
22. Vivado Design Suite User Guide. High-Level Synthesis. URL: https://www.xilinx.com/support/documentation /sw_manuals/ xilinx2020_1/ug902-vivado-high-level-synthesis.pdf (accessed: 20.12.2021).

References

1. Widrow, B. & Stearns, S.D. (1985) *Adaptivnaya obrabotka signalov* [Adaptive Signal Processing]. Translated from English. Moscow: Radio i svyaz'.
2. Farhang-Boroujeny, B. (2013) *Adaptive Filters: Theory and Applications*. New York: John Wiley & Sons.
3. Elliott, S. (2000) *Signal Processing for Active Control*. London: Academic Press.
4. Glentis, G.O., Berberidis, K. & Theodoridis, S. (1999) Efficient least squares adaptive algorithms for FIR transversal filtering. *IEEE Signal Processing Magazine*. 16(4). pp. 13–41. DOI: 10.1109/79.774932
5. Caraiscos, C. & Liu, B. (1984) A roundoff error analysis of the LMS adaptive algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*. 32(1). pp. 34–41. DOI: 10.1109/TASSP.1984.1164286
6. Shashikala, P., Renjith Kumar, T. & Subramani, H. (2013) An FPGA implementation of the LMS adaptive filter for active vibration control. *International Journal of Research in Engineering and Technology*. 2(10). pp. 1–10.
7. Wang, L., Zhang, Z. & Kai, A. (2013) Hands-free speaker identification based on spectral subtraction using a multi-channel least mean square approach. *Acoustics, Speech and Signal Processing: Proceedings of the IEEE International Conference*. pp. 7224–7228.
8. Wang, L., Kitaoka, N. & Nakagawa, S. (2011) Distant-talking speech recognition based on spectral subtraction by multi-channel LMS algorithm. *IEICE Transactions on Information and Systems*. 94(3). pp. 659–667. DOI: 10.1587/transinf.E94.D.659
9. Antonacci, F., Lonoce, D., Motta, M., Sarti, A. & Tubaro, S. (2005) Efficient source localization and tracking in reverberant environments using microphone arrays. *Acoustics, Speech, and Signal Processing: Proceedings of the IEEE International Conference*. 4. pp. iv/1061-iv/1064.
10. Benesty, J. & Huang, Y. (2013) *Adaptive Signal Processing: Applications to Real-World Problems*. Springer Science & Business Media.
11. Huang, Y.A. & Benesty, J. (2002) Adaptive multi-channel least mean square and Newton algorithms for blind channel identification. *Signal Processing*. 82(8). pp. 1127–1138. DOI: 10.1016/S0165-1684(02)00247-5
12. Nelson, P.A., Orduna-Bustamante, F. & Hamada, H. (1992) Multi-channel signal processing techniques in the reproduction of sound. *Journal of the Audio Engineering Society*. 44(11). pp. 973–989.
13. Yaqin, W., Xuebin, L. & Bingliang, H. (2011) Implementation of a LMS filter on FPGA employing extremeDSP and smart IP-core design. *Electronic Measurement & Instruments: Proceedings of the 10th International Conference*. pp. 341–345.
14. Bahoura, M. & Ezzaidi, H. (2011) FPGA-implementation of parallel and sequential architectures for adaptive noise cancelation. *Circuits, Systems, and Signal Processing*. 30(6). pp. 1521–1548. DOI: 10.1007/s00034-011-9310-0
15. Liu, Y., Ge, S., Xing, J., Cui, Z. & Meng, J. (2019) FPGA implementation of high-throughput, low-latency complex LMS algorithm. *Signal, Information and Data Processing*: *Proceedings of the IEEE International Conference*. pp. 1–4.
16. Liwei, C., Zhiliang, T. & Lidong, C. (2015) LMS algorithm fixed-point modeling designated based on System Generator modules. *Environmental Electromagnetics: Proc. of the 7th Asia-Pacific Conference*. pp. 152–156.
17. Ushenina, I.V. (2012) Realizatsiya algoritmov tsifrovoy adaptivnoy fil'tratsii na PLIS [FPGA-based implementation of digital adaptive filtering algorithms]. *XXI vek: itogi proshlogo i problemy nastoyashchego plyus*. 9(5). pp. 134–138.
18. Vasudeva, B., Deora, P., Pradhan, P.M. & Dasgupta, S. (2020) Efficient implementation of LMS adaptive filter-based FECG extraction on an FPGA. *Healthcare Technology Letters*. 7(5). pp. 125–131. DOI: 10.48550/arXiv.1910.07496
19. Martyshkin, A.I. & Salnikov, I.I. (2018) Hardware memory buffer module for multiprocessor system. *Ad Alta: Journal of Interdisciplinary Research*. 8(1). pp. 304–308.
20. Martyshkin, A.I., Salnikov, I.I., Pashchenko, D.V. & Trokoz, D.A. (2018) Associative co-processor on the basis of programmable logical integrated circuits for special purpose computer systems. *Proceedings of the 2018 Global Smart Industry Conference*. pp. 1–5.
21. Mittal, S. (2020) A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computing and Applications*. 32(4). pp. 1109–1139. DOI: 10.1007/s00521-018-3761-1
22. AMD Xilinx. (2020) *Vivado Design Suite User Guide. High-Level Synthesis*. [Online] Available from: https://www.xilinx.com/support/docu mentation/sw_manuals/xilinx2020_1/ug902-vivado-high-level-synthesis.pdf. (Accessed: 20th December 2021).

***Информация об авторе*:**

Ушенина Инна Владимировна – доцент, кандидат технических наук, доцент кафедры программирования Пензенского государственного технологического университета (Пенза, Россия). E-mail: ivl23@yandex.ru

Автор заявляет об отсутствии конфликта интересов.

Information about the author:

Ushenina Inna V. (Candidate of Technical Sciences, Associate Professor, Penza State Technological University, Penza, Russian Federation). E-mail: ivl23@yandex.ru

The author declares no conflicts of interests.

Поступила в редакцию 23.12.2021; принята к публикации 30.05.2022

Received 23.12.2021; accepted for publication 30.05.2022