УДК 621.3.049.77:004.312

ОСОБЕННОСТИ РЕАЛИЗАЦИИ НА ПЛИС ОСНОВНЫХ БЛОКОВ РЕКУРРЕНТНОГО ОБРАБОТЧИКА СИГНАЛОВ

В.С. Петрухин, Ю.А. Степченков, В.Н. Волчек, А.А. Прокофьев, Р.А. Зеленов

Рассматриваются особенности проектирования сложных функциональных блоков рекуррентного обработчика сигналов с использованием языка VHDL. На конкретных примерах показано, как в зависимости от выбранных критериев реализации используются определенные конструкции языка VHDL.

1. Введение

В условиях мелкосерийного производства использование языков описания аппаратуры (Hardware Description Language, HDL) позволяет максимально сократить сроки разработки устройств на основе программируемых логических интегральных схем (ПЛИС) и минимизировать финансовые затраты. Но, как известно, за универсальность разработчику приходится платить. Необходимо подстраиваться под используемый базис. В ряде случаев устройство, легко реализуемое по заказной технологии, довольно сложно оптимально «положить» в ПЛИС.

Поэтому задача оптимальной реализации любых устройств на ПЛИС становится наиболее актуальной, особенно с использованием широко распространенного языка, VHDL (Very high speed integrated circuits HDL). Следует отметить, что изначально язык VHDL разрабатывался исключительно как язык описания аппаратуры для моделирования, а не для синтеза [1], и поэтому многие конструкции языка поддерживаются системой моделирования и не поддерживаются компилятором. При этом возникают определенные проблемы: логическую функцию можно описать с помощью разных конструкций VHDL, и реализация этой функции в кристалле может быть в каждом случае различной. Ставится задача выбрать конструкции VHDL, позволяющие эффективно реализовать устройство при синтезе конкретной схемы.

Существует несколько моделей проектирования схем на языке VHDL: поведенческая, структурная и потоковая. Поведенческая модель описывается последовательными программными операторами, как в любом современном языке программирования высокого уровня. В структурном описании архитектура устройства представляется в виде иерархии связанных компонентов. Потоковое описание представляет собой множество параллельных регистровых операций, каждая из которых управляется вентильными сигналами. Перечисленные модели проектирования представляет собой разный уровень абстракции модели устройства. На практике при разработке относительно крупных проектов, как правило, используется смешанное описание модели проектирования, включающее в себя особенности всех моделей. Более подробно с каждым стилем проектирования можно ознакомиться в [2].

Рекуррентный обработчик сигналов (РОС) имеет нетрадиционную архитектуру и требует реализации ряда блоков, нетипичных для традиционной архитектуры. Поэтому была поставлена задача дать анализ эффективности использования языка VHDL для синтеза блоков комбинированного варианта РОС [3]. Это связано с тем, что, несмотря на определенные заданные стандарты синтеза, каждый компилятор синтезирует схему по-своему, да и сам процесс синтеза имеет огромное количество различных опций и настроек, таких, как оптимизация по площади кристалла, временным характеристикам, трассировке связей и т.д. Наиболее полное описание базовых особенностей использования конструкций языка VHDL при синтезе дано в работах [4; 5].

2. Особенности реализации блока MAC

Блок MAC (Multiplier with ACcumulation) — один из самых сложных аппаратных блоков РОС. Тот факт, что этот блок присутствует в каждой секции РОС, требует особенно тщательного подхода к его реализации. Обработка сигналов в режиме реального времени, большая разрядность внутреннего представления данных, сложные суперскалярные команды: BUTT, LSP, ED, LMS — все эти свойства в условиях ограничений ПЛИС и языка VHDL вызывают определенные трудности для оптимальной аппаратной реализации в заданном базисе. Таким образом, в ре-

зультате проведенного исследования были выявлены наиболее оптимальные конструкции языка VHDL для дальнейшей реализации РОС на ПЛИС. Ниже будут перечислены конкретные проблемы, с которыми пришлось столкнуться, а также варианты решения, возможные и оптимальные в данном случае.

Блок MAC представляет собой вычислительное устройство, ориентированное на сигнальную обработку. При проектировании подобных устройств основное внимание уделяется организации быстрой и высокоточной операции умножения с накоплением. Современные ПЛИС имеют в своем составе достаточно большое количество встроенных аппаратных умножителей и сумматоров для ее реализации. В ПЛИС Cyclone III, на базе которой разрабатывается РОС, присутствуют аппаратные умножители разрядностью 9 битов.

При разработке ряда команд MAC'а учитывалась следующая особенность: каждой операции умножения или сложения в тексте VHDL-программы компилятор на RTL-уровне ставит в соответствие аппаратный умножитель или сумматор; иначе говоря, сколько в коде будет операций «*» или «+», столько соответствующих устройств и будет задействовано на RTL-уровне. Однако стоит отметить тот факт, что представления схемы на RTL-уровне и схемы в кристалле иногда могут отличаться, так как RTL-компоненты реализуются с помощью четырехвходовых таблиц соответствия (LUT, Look-Up Tables), аппаратных умножителей, сумматоров и т. д. Если при моделировании в каждой ветви условных операторов *if* и *case* можно указывать команды умножения или сложения, то при синтезе такие конструкции могут привести к нерациональному использованию площади кристалла или к структурному несоответствию схемы на RTL-уровне и конечной схемы на уровне кристалла.

Язык VHDL позволяет проектировать схемы на разных уровнях абстракции: чем абстракция выше, тем меньше размер кода, т. е. схема проще и быстрее, но при этом прозрачность работы и управляемость компилятором становится ниже. Для сложных, тем более нетрадиционных схем, таких, как РОС, рекомендуется описание схемы на низком уровне абстракции. Это приводит к использованию переменных языка VHDL. К тому же переменные — это единственная возможность сохранить результат операции и прочитать его в рамках одного такта. При реализации блока MAC переменные используются для предварительной

подготовки операндов перед выполнением команды. Для наглядности рассмотрим пример использования подобной конструкции.

Если есть задача — провести какую-либо арифметическую операцию над различными источниками, и выбор этих источников зависит от поступающего на вход кода операции, то можно воспользоваться предварительной подготовкой операндов.

Пример 1. *VHDL код*:

```
case epode is
    when '0'=>
        op1:=source1;
        op2:=source2;
    when '1'=>
        op1:=source3;
        op2:=source4;
    when others=>
        op1:=(others=>'0');
        op2:=(others=>'0');
end case;
    result<=op1*op2;
```

Предварительная подготовка операндов позволяет задействовать меньшее количество арифметических операций в тексте VHDL-программы, при этом подобное описание не усложняет схему, что дает возможность развивать устройство без каких-либо трудностей в понимании работы модели. Текст программы достаточно наглядный, а описание позволяет максимально точно оценить схему, которая будет получена в результате синтеза.

Альтернативный вариант не требует предварительной подготовки операндов. В каждой ветке оператора выбора *case* происходит перемножение нужных операндов.

Пример 2. *VHDL код*:

```
case opcode is
    when '0'=>
        result<=source1*source2;
    when '1'=>
        result<=source3*source4;
    when others=>
        result<=(others=>'0');
end case;
```

Подобный стиль описания свойственен классическим языкам программирования. Он обладает большей наглядностью с позиции алгоритма, но при синтезе вызывает некоторые сложности.

На рис. 1 показана схема примера с предварительной подготовкой операндов на RTL-уровне (пример 1).

На рис. 1 видно, что в схеме задействовано оптимальное количество логических элементов — два мультиплексора и один аппаратный умножитель.

На рис. 2 показана RTL-схема без предварительной подготовки операндов (пример 2).

Очевидно, что во втором случае в RTL-схеме — два умножителя для каждой ветки оператора *case*. Таким образом, вариант описания с предварительной подготовкой операндов более оптимален. Стоит отметить, что в данных случаях в действительности в ПЛИС задействуется один встроенный 9-битный умножитель. Как было отмечено выше, реальные затраты не соответствуют RTL-схеме. Однако возможны ситуации, когда нерациональная реализация RTL-схемы приведет к нерациональному использованию площади кристалла. Поэтому с целью большей наглядности и лучшей управляемости процессом компиляции при разработке MAC использовалась конструкция, аналогичная примеру 1 — с предварительной подготовкой операндов.

Одни из самых часто используемых операторов при проектировании схем на языке VHDL — условные операторы присваивания значений сигналам — операторы *if* и *case*. Однако для эффективного построения схем необходимо учитывать некоторые особенности использования данных операторов. Существенно, каким сигналам присваиваются значения в альтернативных ветвях конструкций. Присваивание значений одинаковым наборам сигналов в каждой из ветвей позволяет избегать появления лишних элементов памяти в схеме. Вложение операторов *if* ведет к увеличению аппаратных затрат и ухудшению временны́х характеристик схем [4].

Необходимо понять возможности использования операторов *case* и *if* при синтезе схемы: для оператора *case* задействуется только один мультиплексор, в то время как в операторе *if* для каждой секции *elsif* будет задействовано по одному мультиплексору дополнительно (как и для вложенных операторов *if*), что приводит к нерациональному использованию логических элементов. Поэтому, где это возможно, предпочтительнее использовать оператор *case*, что и отображено в VHDL модели РОС.

Как правило, на схему подаются тактовый сигнал и сигнал асинхронного сброса. В теле параллельного оператора *process*

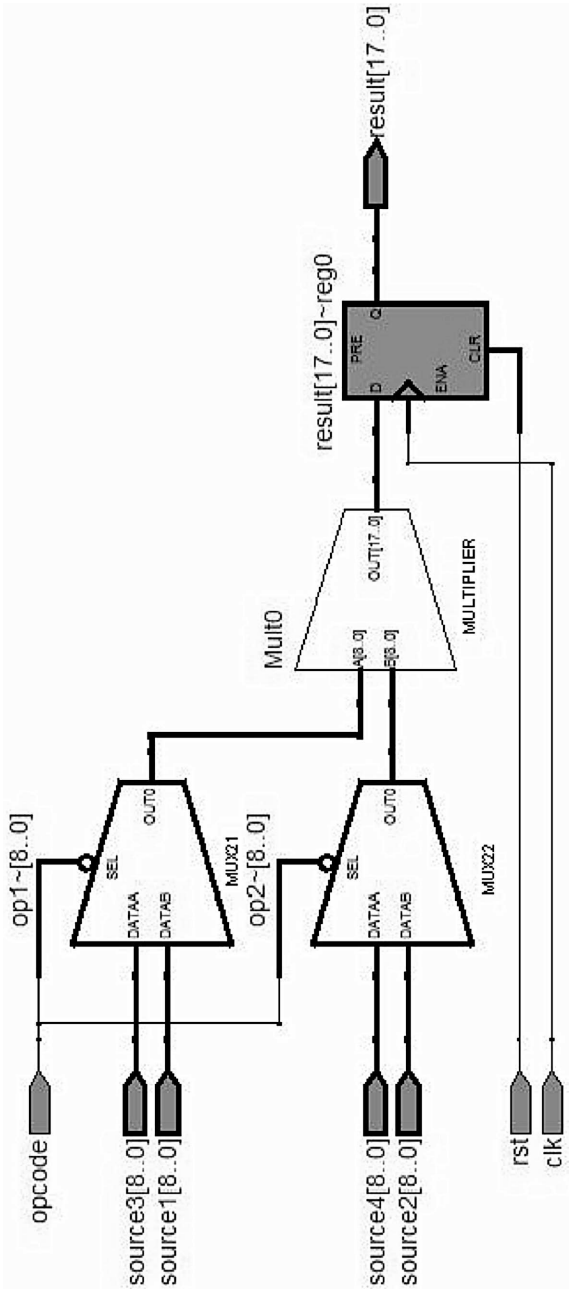

Рис. 1
RTL-схема для VHDL-модели с предварительной подготовкой операндов (пример 1)

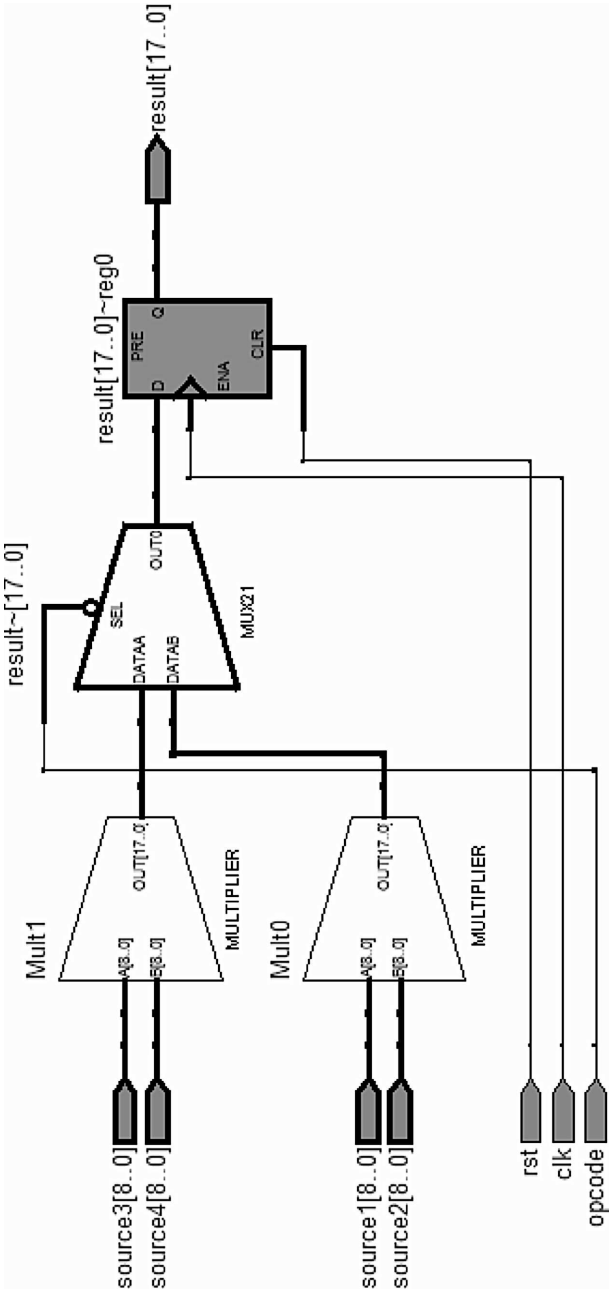

Рис. 2. RTL-схема для VHDL-модели без предварительной подготовки операндов (пример 2)

с помощью конструкции *if-elsif* задается сначала асинхронный сброс, а затем тактирование схемы (пример 3).

Пример 3. *VHDL код*:

```
process (clk,rst)
begin
if rst='1' then
    out_1<=(others=>'0');
    out_2<=(others=>'0');
    out_3<=(others=>'0');
    out_4<=(others=>'0');
elsif clk'event and clk='1' then
if opcode="00" then
    out_1<=in_1+in_2;
elsif opcode="01" then
    out_2<=in_1-in_2;
elsif opcode="10" then
    out_3<=in_1+"0001";
elsif opcode="11>> then
    out_4<=in_1-"0011";
end if;
end if;
end process;
```

Если в теле процесса указывать асинхронный сброс и тактовый сигнал, то выходным значениям будут соответствовать регистры, на которые подается тактовый сигнал и сигнал сброса. Если же их не указывать, то в схеме появятся регистры-защелки, так называемые «латчи» (latch), при этом сильно возрастает число задействованных логических элементов.

«Латч» — небольшой комбинационный контур, предназначенный для хранения значения сигнала до того момента, пока не будет назначено новое значение. FPGA-архитектура базируется на регистрах. В FPGA-устройствах «латчи» используют больше логических ресурсов и приводят, в отличие от регистров, к ухудшению характеристик. Временны́е характеристики схемы могут стать непредсказуемыми, даже если проводился её временной анализ средствами САПР.

Поэтому компания Altera настоятельно рекомендует избегать применения «латчей» в разрабатываемых устройствах. Однако проблема состоит в том, что разработчик может и не догадываться, что синтезатор введет в схему регистр-защелку. Для исключения подобных ситуаций необходимо придерживаться следующих рекомендаций по созданию VHDL-описаний:

• задавать в теле процесса асинхронный сброс всех внутренних регистров, а также активацию внутренней логики по тактовому сигналу;

• в условных операторах *if–elsif–else* всегда применять завершающий *else* (чтобы не возникало ситуации, когда не определено поведение внутренней логики при определенных входных наборах) или, если возможно, заменить этот оператор оператором выбора *case*, при использовании которого сам синтезатор не допустит неопределенных ситуаций.

Подаваемые на вход блока MAC операнды могут быть двух типов — знаковые и беззнаковые, для работы с которыми в VHDL необходимо подключать специальные библиотеки. Как правило, это стандартные пакеты —*std_logic_arith* и *Numeric_std*. Эти пакеты содержат перекрывающиеся определения типов и функций, поэтому одновременное их использование в некоторых САПР не поддерживается. В VHDL-модели РОС используется пакет *Numeric_std*, так как он достаточно хорошо изучен и подробно описан в русскоязычной литературе [6].

Знаковый операнд задается типом *signed*, беззнаковый — типом *unsigned*. К сожалению, язык VHDL не поддерживает перегрузку типа операнда в ходе выполнения программы. Тип операнда зафиксирован на все время выполнения программы и задается в момент объявления сигнала или переменной. Но в блоке MAC одни и те же операнды в зависимости от управляющих сигналов могут быть как знаковыми, так и беззнаковыми. Для решения этой проблемы все операнды в модели представлены как знаковые, т. е. имеют тип *signed*. Разрядная сетка увеличена на один бит слева, причем самый старший бит становится знаковым. В случае, если операнды пришли в блок MAC как знаковые, старший добавленный бит дублирует знаковый разряд; если же операнды пришли как беззнаковые, то старший бит всегда в нуле. Тем самым, имея разрядную сетку $n + 1$ и управляя значением двух крайних левых разрядов, можно менять знак знакового операнда и иметь необходимое знаковое или беззнаковое n-разрядное число. Подобным образом описываются все сигналы и переменные, участвующие в вычислительном процессе, в том числе и представленные в блоке MAC аккумуляторы: 16-разрядный аккумулятор в АЛУ и два 40-разрядных аккумулятора блока DSP. Все внутренние регистры и аккумуляторы представлены внутренними сигналами и объявлены внутри тела *architecture*.

Таблица 1

Отчет по использованию ресурсов в блоке MAC

Ресурс	Использование
Общее количество логических элементов:	1,069/119,088 (< 1 %)
• комбинационные, без регистров	934
• только регистровые	0
• комбинационные, с регистрами	135
Логические элементы, используемые в следующих видах LUT:	
• 4-входовые LUT	643
• 3-входовые LUT	200
• <=2-входовые LUT	226
• регистровые только	0
Логические элементы, в следующих режимах:	
• нормальный режим	838
• арифметический режим	231
Общее количество регистров:	135/122,205 (< 1 %)
• специальные логические регистры	135/119,620 (< 1 %)
• I/O регистры	0/2,585 (0 %)
Общее количество LAB	72/7,443 (< 1 %)
Логические элементы, встроенные пользователем	0
Виртуальные контакты	0
I/O контакты:	99/532 (19 %)
• тактовые входы	2/8 (25 %)
• специальные входные контакты	0/9 (0 %)
Глобальные сигналы	2
M9Ks	0/432 (0 %)
Общее количество битов памяти	0/3,981,312 (0 %)

Таблица 1 (*окончание*)

Ресурс	Использование
Общее количество битов в RAM блоках	0/3,981,312 (0%)
Встроенные 9-битные умножители	2/576 (< 1%)
PLLs	0/4 (0%)
Глобальные тактовые сигналы	2/20 (10%)
Блоки управления сопротивлением	0/4 (0%)
Среднее использование связей	0%
Максимальное использование связей	18%
Maximum fan-out node	lpm_mult:Mult0|mult_ir01:auto_generated|result[0]
Maximum fan-out	143
Highest non-global fan-out signal	Selector108~103
Highest non-global fan-out	84
Total fan-out	4290
Average fan-out	3.03

Примечание. В табл. 1 и 4 применяются следующие обозначения:

- **ЛЭ** — логический элемент;
- **LUT** (Look Up Table) — таблица соответствия;
- **LAB** (Logic Area Block) — блок логических элементов, содержит 16 ЛЭ;
- **нормальный режим** работы ЛЭ, используемый для построения логических комбинационных функций;
- **арифметический режим** работы ЛЭ, идеальный для реализации сумматоров, счетчиков, аккумуляторов и компараторов;
- **M9K** — блок памяти, обеспечивающий до 9 Кбит емкости, работающий на частотах до 260 МГц;
- **PLL** — схема, обеспечивающая умножение (деление) некоторой частоты; может быть динамически перенастроена для обеспечения автоподстройки для работы с внешней памятью во время работы всего устройства;
- **Fan-out** — нагрузочный множитель по выходу/коэффициенты разветвления по выходу.

В табл. 1 приведены результаты выполнения компиляции, а в табл. 2 — отчет по затраченным связям блока MAC в САПР Quartus II v.7.2 [7]. В качестве ПЛИС выбрана модель EP3C120F780I7 семейства Cyclone III [8].

Таблица 2

Отчет по использованию связей в блоке MAC

Тип связи	Использование
Связи между блоками (Block interconnects)	1,972/342,891 (< 1%)
C16 связи (interconnects)	170/10,120 (2%)
C4 связи (interconnects)	1,453/209,544 (< 1%)
Быстрые связи соседних блоков (Direct links)	104/342,891 (< 1%)
Глобальные тактовые сигналы (Global clocks)	2/20 (10%)
Локальные связи (Local interconnects)	478/119,088 (< 1%)
R24 связи (interconnects)	113/9,963 (1%)
R4 связи (interconnects)	1,603/289,782 (< 1%)

Примечание. В табл. 2 и 5 применяются следующие обозначения:

- **C16 interconnects** соединяют 16 блоков по вертикали. Обеспечивают быструю связь друг с другом несмежных блоков (LABs);
- **C4 interconnects** захватывают 4 блока вверх или вниз от исходного блока;
- **Direct links** — связи, обеспечивающие быструю передачу сигналов между соседними блоками, без использования ресурсов горизонтальных связей;
- **Local interconnects** соединяют горизонтально и вертикально логические элементы в одном LAB;
- **R4 interconnects** соединяют 4 LABs, или 3 LABs и один M9K блок памяти, или 3 LABs и один встроенный умножитель справа или слева от исходного LAB. Эти ресурсы используются для быстрой горизонтальной связи в пространстве 4-х LABs;
- **R24 interconnects** соединяют 24 LABs по горизонтали.

Тактовая частота блока MAC, полученная в результате проведения временного анализа, получилась равной 50,21 МГц (период равен 19,918 нс).

3. Особенности реализации блока ПАП

Основным элементом ПАП (память адресной проверки) является память, поэтому в первую очередь стоял вопрос о выборе наиболее подходящей организации хранения данных. Элементы самодостаточных данных (ЭСД) состоят из двух частей — функциональной и содержательной. В зависимости от типа операнда в память не всегда записываются обе части ЭСД. Поэтому для хранения данных используются две независимые памяти (для хранения функциональной и для хранения содержательной частей). Здесь формируется общая шина адреса и отдельные сигналы разрешения записи. Под флаг занятости ячейки памяти была на единицу расширена разрядность слова памяти, хранящей содержательную часть ЭСД.

Имеется два варианта организации выходного порта памяти: с буферизацией (тактируемый выход) и без нее (нетактируемый выход) [9]. Было решено отказаться от буферизации выходного сигнала для уменьшения периода выдачи данных на такт (рис. 3 и 4). Синтез выполнялся для ПЛИС EP3C25U256C6 семейства Cyclone III.

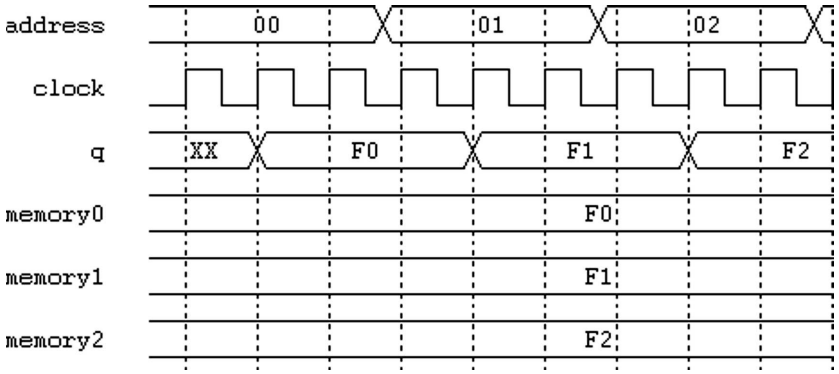

Рис. 3. Чтение из памяти с тактируемым выходом

Первый вариант ПАП был реализован на однопортовой памяти. В этом случае предполагается, что после подачи входных данных необходимо ожидать выходные данные в течение двух тактов, после чего можно подавать следующий ЭСД. Такой вариант реализации ПАП характеризуется минимальными аппаратными затратами и, следовательно, максимальной частотой

работы. Но с учетом того, что ПАП может принимать данные только в одном такте из трех (см. рис. 5), ее частота может стать недостаточной и замедлить работу всего блока РОС. Входной операнд необходимо хранить во внешнем регистре, либо можно организовать этот регистр внутри блока ПАП.

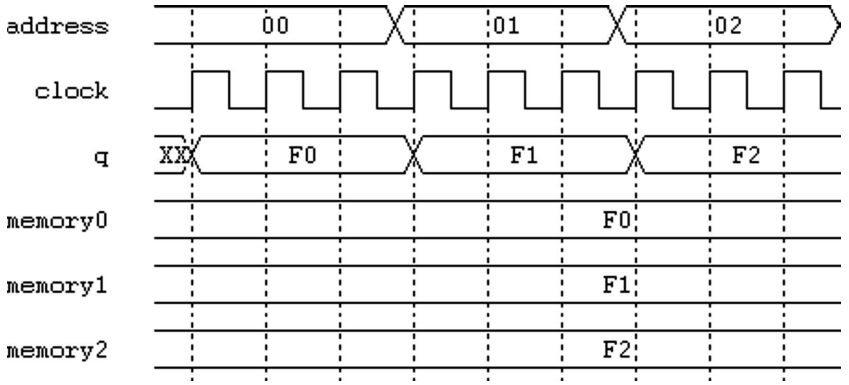

Рис. 4. Чтение из памяти с нетактируемым выходом

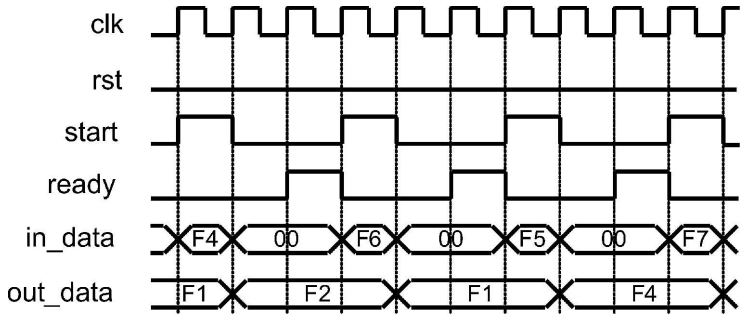

Рис. 5. Временны́е диаграммы ПАП на однопортовой памяти

Временные затраты можно уменьшить реализацией ПАП на двухпортовой памяти, разрешающей подавать данные на каждом такте (конвейерная организация). Но при этом возникают проблемы с одновременным обращением к ячейке памяти для записи и чтения. Поскольку в РОС не предусматриваются коллизии такого типа, были введены дополнительные регистры, предотвращающие одновременное обращение к ячейке памяти для записи и чтения, а также к ячейкам, где еще не зафиксировались новые

данные. Для хранения данных тактов $i-1$ и $i-2$ потребовалось два дополнительных регистра.

Дополнительные регистры работают следующим образом. Каждый элемент данных, который должен быть записан в память, сохраняется также во временном регистре, пока новое значение не зафиксируется в памяти (т. е. два такта). При обращении к ячейке памяти сначала проверяется регистр, хранящий данные, записываемые на такте $i-1$, и если адреса совпадают, то обрабатываются данные не из памяти, а из временного регистра. Затем идет проверка данных, сохраняемых на такте $i-2$, и, аналогично, при совпадении адресов берутся данные из временного регистра. Иначе обрабатываются данные, хранящиеся в памяти.

Использование такой схемы позволяет избежать обращений к ячейкам, где хранятся неактуальные данные, и дает возможность не заботиться о последовательности подаваемых на ПАП данных (рис. 6). На выходе ПАП получается пара сцепленных операндов. Результаты синтеза для обоих вариантов реализации ПАП приведены в табл. 3.

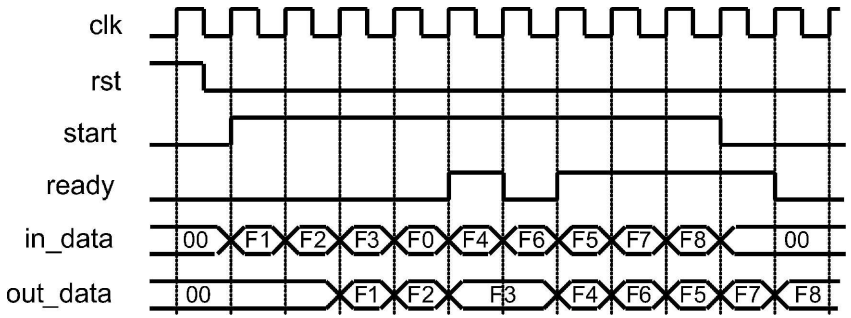

Рис. 6. Временны́е диаграммы ПАП на двухпортовой памяти

Увеличение числа регистров в случае конвейерной ПАП происходит из-за необходимости введения дополнительных временны́х регистров и усложнения логики. В итоге при увеличении аппаратных затрат в 4,5 раза время обработки данных уменьшается в 2,4 раза.

В табл. 4 приведены результаты выполнения компиляции, в табл. 5 — отчет по затраченным связям блока ПАП в САПР Quartus II v.7.2. В качестве ПЛИС выбрана модель EP3C120F780I7 семейства Cyclone III.

Таблица 3

Сводная таблица результатов синтеза ПАП

Параметр	ПАП на основе однопортовой памяти	ПАП на основе двухпортовой памяти
Количество задействованных логических элементов	65	287
Количество используемых регистров	49	249
Количество задействованных битов памяти	1,344	1,344
Частота, МГц	158,23 (период 6,320 нс)	125,85 (период 7,946 нс)
Количество тактов в цикле обработки данных	3	1
Время цикла обработки данных, нс	18,960	7,946

Таблица 4

Отчет по использованию ресурсов в блоке ПАП

Ресурс	Однопортовая память	Двухпортовая память
Общее количество логических элементов:	65/119,088 (< 1 %)	287/119,088 (< 1 %)
• комбинационные, без регистров	16	38
• только регистровые	40	163
• комбинационные, с регистрами	9	86
Логические элементы, используемые в следующих видах LUT:		
• четырехвходовые LUT	18	36
• трехвходовые LUT	4	74
• двухвходовые LUT	3	14
• регистровые только	40	163
Логические элементы в следующих режимах:		
• нормальный режим	25	124
• арифметический режим	0	0

Таблица 4 (*продолжение*)

Ресурс	Однопортовая память	Двухпортовая память
Общее количество регистров:	49/122,205 (< 1 %)	249/122,205 (< 1 %)
• специальные логические регистры	49/119,620 (< 1 %)	249/119,620 (< 1 %)
• I/O-регистры	0/2,585 (0 %)	0/2,585 (0 %)
Общее количество LAB	5/7,443 (< 1 %)	20/7,443 (< 1 %)
Логические элементы, встроенные пользователем	0	0
Виртуальные контакты	0	0
I/O контакты:	93/532 (17 %)	134/532 (25 %)
• тактовые входы	1/8 (13 %)	1/8 (13 %)
• специальные входные контакты	0/9 (0 %)	0/9 (0 %)
Глобальные сигналы	2	2
M9Ks	2/432 (< 1 %)	2/432 (< 1 %)
Общее количество битов памяти	1,344/3,981,312 (< 1 %)	1,344/3,981,312 (< 1 %)
Общее количество битов в RAM блоках	18,432/3,981,312 (< 1 %)	18,432/3,981,312 (< 1 %)
Встроенные девятибитные умножители	0/576 (0 %)	0/576 (0 %)
PLLs	0/4 (0 %)	0/4 (0 %)
Глобальные тактовые сигналы	2/20 (10 %)	2/20 (10 %)
Блоки управления сопротивлением	0/4 (0 %)	0/4 (0 %)
Среднее использование связей, %	0	0
Максимальное использование связей, %	2	3
Maximum fan-out node	clk~inputclkctrl	clk~inputclkctrl
Maximum fan-out	52	251
Highest non-global fan-out signal	process0~0	rst~input

Таблица 4 (*окончание*)

Ресурс	Однопортовая память	Двухпортовая память
Highest non-global fan-out	45	176
Total fan-out	504	1697
Общее количество логических элементов	1.69	2.14

Таблица 5

Отчет по использованию связей в блоке MAC

Тип связи	Однопортовая память	Двухпортовая память
Связи между блоками (Block interconnects)	170/342,891 (< 1 %)	526/342,891 (< 1 %)
C16 связи (interconnects)	106/10,120 (1 %)	126/10,120 (1 %)
C4 связи (interconnects)	306/209,544 (< 1 %)	534/209,544 (< 1 %)
Быстрые связи соседних блоков (Direct links)	1/342,891 (< 1 %)	56/342,891 (< 1 %)
Глобальные тактовые сигналы (Global clocks)	2/20 (10 %)	2/20 (10 %)
Локальные связи (Local interconnects)	19/119,088 (< 1 %)	161/119,088 (< 1 %)
R24 связи (interconnects)	77/9,963 (< 1 %)	118/9,963 (1 %)
R4 связи (interconnects)	228/289,782 (< 1 %)	539/289,782 (< 1 %)

4. Выводы

1. В результате проведенных исследований выявлены конструкции языка VHDL, позволяющие получить оптимальную реализацию при синтезе схемы.

2. Данные о затраченных ресурсах и связях позволяют наглядно оценить аппаратные затраты, которые в дальнейшем приведут к выбору конкретной модели ПЛИС для разработки РОС. В опытном образце планируется реализовать четыре параллельных секции РОС; каждая секция РОС будет включать в себя по

одному блоку MAC и по одному блоку ПАП, а также дополнительную управляющую логику.

3. Приведенные в данной работе приемы и конструкции языка VHDL могут быть использованы и в других проектах, связанных с реализацией цифровых устройств на ПЛИС, в частности для реализации устройств сигнальной обработки.

Список литературы

1. *Емец С.* Verilog — инструмент разработки цифровых электронных схем // Компоненты и технологии. Вып. 2. М., 2002. www.compitech.ru/html.cgi/arhiv/01_02/stat_86.htm/.
2. *Стешенко В.*, *Шишкин Г.*, *Евстифеев А.*, *Седякин Ю.* Школа разработки аппаратуры цифровой обработки сигналов на ПЛИС. Занятие 4. Язык описания аппаратуры VHDL. http://www.chipnews.ru/html.cgi/arhiv/00_01/stat-28.htm/.
3. *Степченков Ю.А.*, *Петрухин В.С.* Особенности гибридного варианта реализации на ПЛИС рекуррентного обработчика сигналов // Наст. сб. С. 118–129.
4. *Суворова Е.А.*, *Шейнин Ю.Е.* Проектирование цифровых систем на VHDL. СПб: БХВ-Петербург, 2003.
5. *Сергиенко А.М.* VHDL для проектирования вычислительных устройств. Киев: ДиаСофт, 2003.
6. *Бибило П.Н.*, *Авдеев Н.А.* VHDL. Эффективное использование при проектировании цифровых систем. М.: Солон-Пресс, 2006.
7. Quartus II Development Software Handbook Vol. 7.2 (Complete Five-Volume Set). http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf.
8. Cyclone III FPGAs: An Unprecedented Combination of Power, Functionality, and Cost. http://www.altera.com/products/devices/cyclone3/cy3-index.jsp.
9. The Cyclone III FPGA Handbook. http://www.altera.com/literature/hb/cyc3/cyclone3_handbook.pdf.

УДК 681.324.192

ВЫБОР ЯЗЫКОВЫХ СРЕДСТВ ПРЕДСТАВЛЕНИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ДЛЯ РЕКУРРЕНТНОГО ОБРАБОТЧИКА СИГНАЛОВ

Ю.А. Степченков, *В.С. Петрухин*, *Д.В. Хилько*

В статье анализируются современные методологии реализации параллельных алгоритмов в различных парадигмах программирования. Рассмотрено несколько конкретных языков программирования и описаны их функциональные возможности. Выработаны критерии, в соответствии с которыми определяется пригодность предложенных языков программирования для реализации параллельных алгоритмов рекуррентным обработчиком сигналов.

1. Введение

Идея параллельных вычислений возникла еще до появления фон-неймановской архитектуры компьютера. Фактически появлению фон-неймановской архитектуры предшествовали различные предложения по реализации архитектур, ориентированных как на последовательные, так и на параллельные вычисления. Но экономически эффективным оказалось лишь предложение фон Неймана.

Однако уже тогда были предложены различные решения и модели архитектур, ориентированных на параллельные вычисления [1], что не осталось незамеченным математиками, программистами и проектировщиками и дало им обильную пищу для размышлений. Пожалуй, именно к тому времени можно отнести возникновение терминов «параллельное программирование», «параллельные алгоритмы».

2. Новая вычислительная парадигма и постановка задачи исследования

В ИПИ РАН ведутся исследования новой вычислительной парадигмы. Была разработана модель рекуррентного обработ-