РАДИОТЕХНИКА И ЭЛЕКТРОНИКА, 2024, том 69, № 3, с. 243–252

ТЕОРИЯ И МЕТОДЫ ОБРАБОТКИ СИГНАЛОВ

УДК 621.3;004.3

ПОВЫШЕНИЕ БЫСТРОДЕЙСТВИЯ ЦИФРОВЫХ УСТРОЙСТВ ПРИ ИСПОЛЬЗОВАНИИ МЕТОДИКИ ASMD-FSMD С ПОМОЩЬЮ ОПЕРАТОРОВ НЕБЛОКИРУЮЩЕГО НАЗНАЧЕНИЯ

© 2024 г. В. В. Соловьев*, А. С. Климович

Белостокский технологический университет,
ул. Вейска, 45А, Белосток, 15-351, Республика Польша

**E-mail: valsol@mail.ru*

Поступила в редакцию 06.03.2023 г.
После доработки 05.04.2023 г.
Принята к публикации 27.04.2023 г.

Рассмотрена методика ASMD-FSMD проектирования цифровых устройств, которая заключается в построении блок-схемы автомата с трактом обработки данных (algorithmic state machine with datapath – ASMD), описывающей поведение устройства, и в создании кода проекта на языке Verilog в виде конечного автомата с трактом обработки данных (finite state machine with datapath – FSMD). Одним из направлений развития методики ASMD-FSMD является использование особенностей языка описания аппаратуры (hardware description language – HDL). Выдвинута гипотеза: в методике ASMD-FSMD возможно применение нескольких операторов неблокирующего назначения к одной и той же переменной в одном такте синхронизации, что приведет к увеличению быстродействия устройства. Выдвинутая гипотеза исследована при проектировании синхронных умножителей, реализующих классические алгоритмы умножения *c* и *d*. Экспериментальные исследования подтвердили справедливость выдвинутой гипотезы, при этом быстродействие умножителей увеличивается в два-три раза, а стоимость реализации в большинстве случаев уменьшается по сравнению с традиционным подходом.

Ключевые слова: автомат с трактом обработки данных, конечный автомат с трактом обработки данных, язык описания аппаратуры, операторы неблокирующего назначения, синхронные умножители

DOI: 10.31857/S0033849424030059, **EDN:** JVEISW

ВВЕДЕНИЕ

Традиционно проектируемое цифровое устройство принято представлять в виде операционного устройства (datapath) и устройства управления (controller или control unit), которые обычно проектируются отдельно: операционное устройство – в виде совокупности стандартных функциональных узлов (регистров, шин, мультиплексоров и др.), а устройство управления – в виде конечного автомата (finite state machine, FSM).

В работе [1] было предложено объединить вместе операционное и управляющее устройства и представить как конечный автомат с трактом обработки данных (finite state machine with datapath – FSMD). Модель FSMD быстро стала популярной, в работе [2] приведены FSMD для синхронных и асинхронных проектов. Модель FSMD оказалась очень удобной для проверки эквивалентности двух схем, полученных в результате синтеза или различных проектных преобразований [3, 4]. В [5] предложено цифровую систему представлять в виде сети FSMD, которая приводит к реализации аппаратуры, свободной от гонок.

Общая модель FSMD не всегда удобна при проектировании конкретных приложений, поэтому в ряде работ предлагаются расширения FSMD: для представления архитектуры процессора и ASIC [6] (application-specific integrated circuit), для синхронного доступа к памяти [7], для программ обработки массивов [8]. Уменьшение потребляемой мощности FSMD путем декомпозиции рассматривается в [9], а путем стробирования – в [10]. Эффективности FSM и FSMD сравниваются в работе [11].

Благодаря своей наглядности блок-схемы автоматов (algorithmic state machine, ASM) получили

широкое распространение для представления алгоритма функционирования FSM. Впервые ASM были предложены в [12] как альтернатива графам автоматов. В [13] рассматривается реализация ASM на PROM, FPLA и мультиплексорах. В [14] предлагаются методы минимизации числа вершин ASM. В [15] описывается инструмент ABELITE синтеза контроллеров, основанных на ASM.

Традиционно ASM используются для представления алгоритма функционирования устройства управления, т.е. FSM. В [16] предложено ASM использовать как для описания поведения устройства управления, так и для описания операций, выполняемых в операционном устройстве. Такая ASM получила название блок-схема автомата с трактом обработки данных (algorithmic state machine with datapath, ASMD). Диаграммы ASMD в последнее время все чаще применяются в проектах на FPGA: при реализации промышленных систем управления [17], для реализации функции asin с помощью алгоритма CORDIC [18], при аппаратной реализации криптографического алгоритма AES [19], при проектировании универсального асинхронного приема-передатчика UART [20] и др.

Методика ASMD-FSMD проектирования цифровых устройств впервые представлена в [21]. В методике ASMD-FSMD предложено описывать код проекта на языке Verilog непосредственно по схеме ASMD в виде FSMD. Сравнение методики ASMD-FSMD с традиционным подходом в случае использования в качестве устройства управления автоматов типа Мили и Мура приведено в [22]. В [23] рассматривается использование методики ASMD-FSMD для проектирования устройств цифровой обработки сигналов.

Главным достоинством методики ASMD-FSMD является значительное сокращение времени разработки проектов (в пять-семь раз). Кроме того, использование методики ASMD-FSMD позволяет в большинстве случаев уменьшить стоимость реализации и увеличить быстродействие проектов по сравнению с традиционным подходом. Одним из направлений развития методики ASMD-FSMD является использование особенностей языка описания аппаратуры (hardware description language, HDL).

Цель данной работы – увеличить быстродействие цифровых устройств с помощью использования в методике ASMD-FSMD операторов неблокирующего назначения языка Verilog [24].

1. МЕТОДИКА ASMD-FSMD

Разработка цифровых устройств с помощью методики ASMD-FSMD заключается в построении схемы ASMD поведения устройства и создании кода проекта в виде FSMD на языке Verilog.

Схема ASMD состоит из блоков ASMD, каждый из которых описывает поведение FSMD в одном состоянии в течение одного такта синхронизации (рис. 1). Блок ASMD включает одну вершину состояния (прямоугольник) и может иметь несколько условных вершин (ромбов) и вершин выходов по условию (овалов), причем ромбы могут как предшествовать овалам, так и следовать после овалов. Блок ASMD имеет только один вход, который является входом в вершину состояния, и может иметь один или несколько выходов. Входы и выходы вершин соединяются с помощью дуг. Обратные связи запрещены внутри блока ASMD. Циклы алгоритма и ждущие состояния в схеме ASMD реализуются с помощью внешних (по отношению к блоку ASMD) обратных связей.

Вблизи вершины состояния записывается имя состояния FSMD (например, S_1). В случае FSMD типа Мура внутри вершины состояния (прямоугольника) записываются операции, выполняемые в данном состоянии FSMD (outputs). В случае FSMD типа Мили в вершинах выхода по условию (овалах) записываются операции, выполняемые на данном переходе FSMD (conditional outputs). В условных вершинах (ромбах) записываются логические выражения (condi-

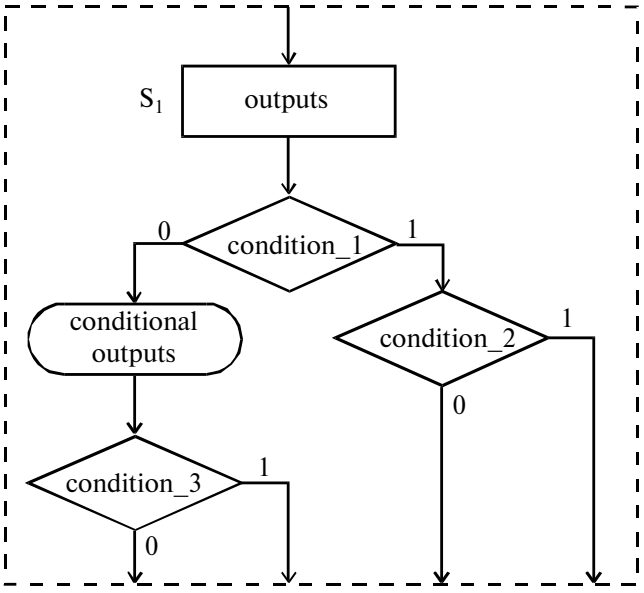


Рис. 1. Блок ASMD.

tion_1, condition_2, ...). Выходы условной вершины обозначаются значениями 0 и 1, которые соответствуют переходам в случае ложного или истинного значения логического выражения. В качестве операций, записываемых в прямоугольниках и овалах, а также в качестве логических выражений могут использоваться любые операции и логические выражения, допустимые в HDL (в нашем случае в языке Verilog).

Схема ASMD представляет собой композицию соединенных между собой блоков ASMD. При этом каждый выход любой вершины ASMD может быть соединен только с одним входом другой вершины, т.е. ветвление алгоритма возможно только в условных вершинах.

Код проекта на языке Verilog строится непосредственно по созданной схеме ASMD. Вначале выполняются необходимые объявления переменных (внутренних регистров устройства) и состояний FSMD. Каждый блок ASMD (состояние FSMD) описывается в виде процедурного блока **begin…end**. Условные вершины описываются с помощью операторов **if**. Присваивание значений переменным описывается с помощью операторов неблокирующего назначения (<=). Операции, выполняемые в прямоугольниках (для FSMD Мура) описываются в начале блока **begin…end**, а операции, выполняемые в овалах (для FSMD Мили) описываются в соответствующих местах операторов **if** (при этом возможно использование операторных скобок **begin…end**).

2. СУТЬ ПРЕДЛАГАЕМОГО ПОДХОДА

Очевидно, что для увеличения быстродействия проекта необходимо минимизировать число состояний FSMD в циклах алгоритма, т.е. в цепях обратной связи ASMD. Пусть с данными, находящимися в регистре (т.е. со значением переменной v), необходимо выполнить несколько последовательных действий: например, сложить содержимое регистра v с содержимым другого регистра w, результат запомнить в исходном регистре ($v = v + w$), сдвинуть содержимое регистра влево или вправо на несколько разрядов ($v = v \ll 1$) и др.

При традиционном подходе для хранения значения переменной необходимо использовать сдвиговый регистр. В этом случае, чтобы загрузить в сдвиговый регистр некоторое значение (результат сложения) и сдвинуть содержимое регистра, необходимо два такта синхронизации: один – для загрузки в регистр внешних данных, а второй – для сдвига содержимого регистра.

Возникает вопрос: а нельзя ли эти действия выполнить за один такт синхронизации. Предпосылкой для такого предположения является то, что в одном блоке ASMD можно использовать несколько вершин выходов по условию (овалов). Кроме того, язык Verilog предоставляет операторы неблокирующего назначения (<=), которые не препятствуют выполнению других операторов в одном и том же процедурном блоке. Отметим также, что для FSMD Мили в одном блоке ASMD можно проверять несколько логических выражений и в зависимости от результатов этих проверок выполнять различные операции с одними и теми же переменными, записывая операции в различных овалах. Другими словами, вопрос в том, можно ли несколько операций с одной и той же переменной выполнять в одном блоке ASMD.

Гипотеза. Использование операторов неблокирующего назначения языка Verilog в методике ASMD-FSMD позволяет выполнить несколько операций с одной и той же переменной за один такт синхронизации, что приводит к увеличению быстродействия устройства, по сравнению с традиционным подходом.

3. ПРИМЕР ИСПОЛЬЗОВАНИЯ МЕТОДИКИ ASMD-FSMD ДЛЯ ПРОЕКТИРОВАНИЯ СИНХРОННОГО УМНОЖИТЕЛЯ

Известны следующие классические методы умножения, по которым строятся синхронные умножители [25]:

алгоритм a: когда проверяется младший разряд множителя, множитель сдвигается вправо, а множимое – влево;

алгоритм b: когда проверяется старший разряд множителя, множитель сдвигается влево, а множимое – вправо;

алгоритм c: когда проверяется старший разряд множителя, множитель и частные произведения сдвигаются влево;

алгоритм d: когда проверяется младший разряд множителя, множитель и частные произведения сдвигаются вправо.

Для проверки нашей гипотезы рассмотрим алгоритм умножения *c*. Операционное устройство и конечный автомат в случае традиционного подхода показаны на рис. 2 и 3 соответственно.

Значения чисел A и B поступают на вход операционного устройства. На выходах операци-

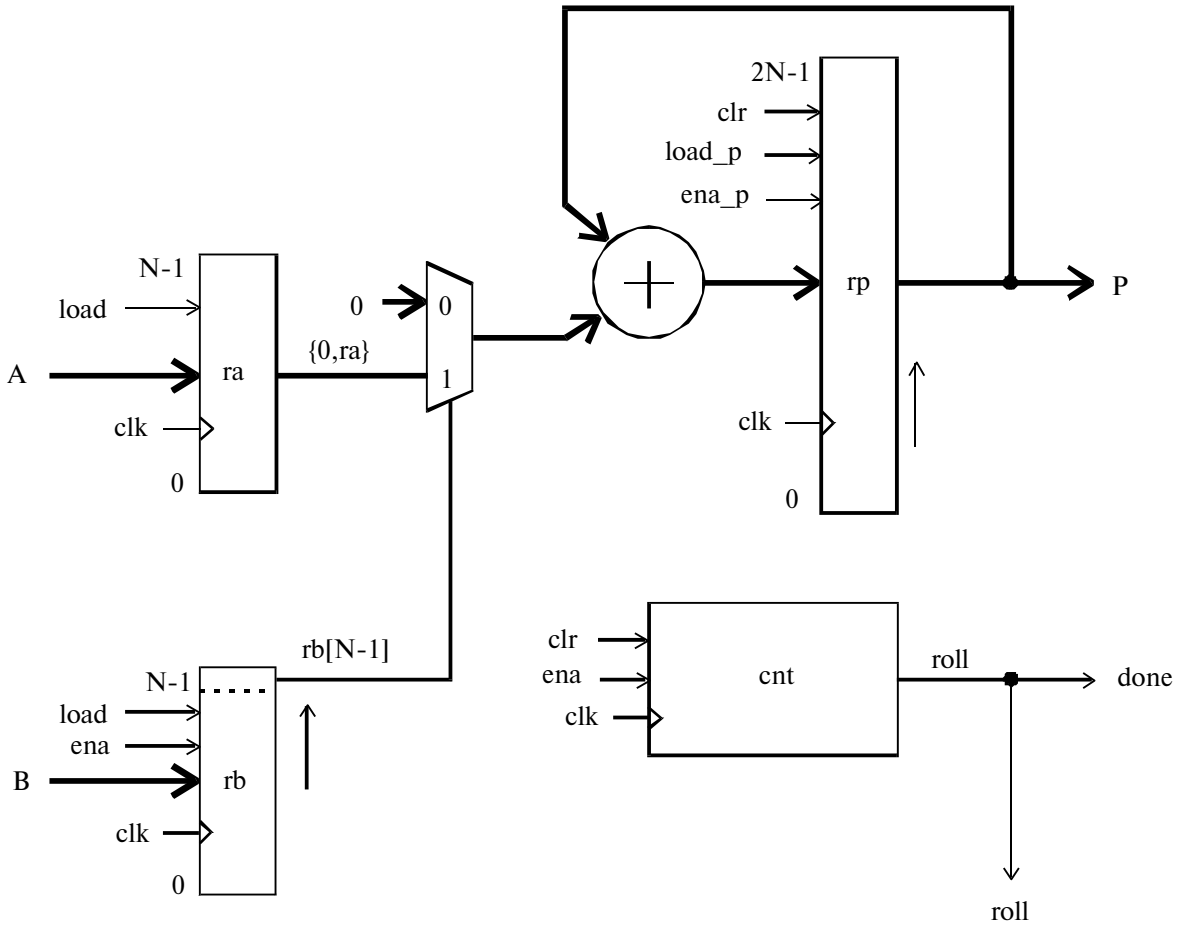


Рис. 2. Операционное устройство синхронного умножителя *c*.

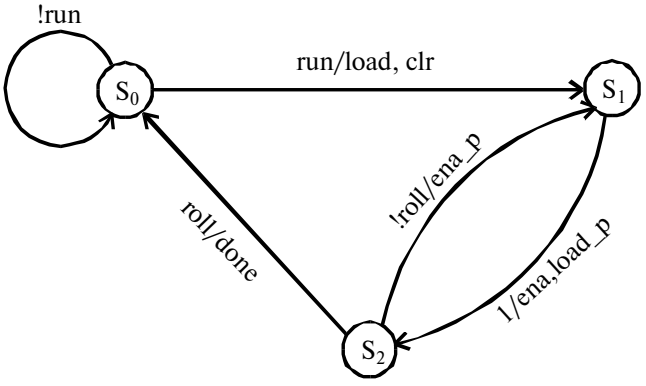


Рис. 3. Устройство управления синхронного умножителя *c* в виде графа конечного автомата типа Мили.

онного устройства формируется произведение *P* и сигнал *done*, указывающий на окончание процесса умножения. Значение множимого *A* (см. рис. 2) загружается в регистр ra, значение множителя *B* загружается в сдвиговый регистр rb; произведение *P* формируется в сдвиговом регистре rp. Сумматор выполняет сложение содержимого регистра rp с множимым *A* или с нулем. Мультиплексор служит для выбора второго операнда сумматора. Счетчик cnt формирует внутренний сигнал обратной связи *roll*, совпадающий с сигналом *done*, единичное значение которого указывает на окончание процесса умножения.

Процесс умножения чисел *A* и *B*, начинается с установки сигнала *run*. Конечный автомат FSM (см. рис. 3) на основании значений сигналов *run* и *roll* формирует необходимые значения следующих управляющих сигналов: *load* – для загрузки в регистры значений умножаемых чисел *A* и *B*; *load_p* – для загрузки результата сложения в регистр rp; *clr* – для сброса регистра rp и счетчика cnt; *ena* – для разрешения операции сдвига содержимого регистра rb и увеличения значения счетчика cnt; *ena_p* – для разрешения операции сдвига содержимого регистра rp. Умножитель также управляется сигналом синхронизации *clk* и сигналом сброса *reset*.

В состоянии S_0 конечный автомат ожидает прихода сигнала *run*, который запускает процесс умножения путем перехода в состояние S_1, при этом формируются сигналы *load* и *clr*.

Для загрузки результатов суммирования в регистр rp и сдвига содержимого регистра rp требуется два такта синхронизации, поэтому одному циклу умножения соответствуют два состояния: S_1 и S_2. При переходе из состояния S_1 в состояние S_2 в регистр rp по сигналу *load_p* загружается результат суммирования, а по сигналу *ena* выполняется сдвиг содержимого регистра rb. При переходе из состояния S_2 в состояние S_1 по сигналу *ena_p* содержимое регистра rp сдвигается влево. Процесс умножения заканчивается (FSM переходит в состояние S_0) после формирования сигнала *roll*.

Таким образом, при традиционном подходе умножение N-битовых чисел A и B с помощью алгоритма c выполняется за $n = 2N + 1$ тактов синхронизации.

Схема ASMD алгоритма умножения c (рис. 4) состоит из двух блоков ASMD, соответствующих состояниям S_0 и S_1 FSMD Мили. В состоянии S_0 ожидается единичное значение сигнала *run*, при его установке выполняется инициализация переменных. Состояние S_1 соответствует циклу умножения. При rb[N–1] = 1 к содержимому регистра rp добавляется значение множимого (в противном случае ничего не добавляется). Затем увеличивается значение счетчика cnt и выполняется сдвиг влево содержимого регистра rb. В этом же блоке ASMD проверяется значение счетчика cnt. Если cnt = N–1, то устанавливается флаг *done* и процесс умножения заканчивается, иначе содержимое регистра rp сдвигается влево и цикл умножения повторяется.

Отметим, что в одном блоке ASMD выполняется увеличение значения и сдвиг содержимого регистра rp, а также увеличение значения счетчика cnt и сравнение этого значения с величиной N–1. Таким образом, при использовании методики ASMD-FSMD умножение N-битовых чисел A и B по алгоритму c выполняется за $n = N + 1$ тактов синхронизации.

Непосредственно по схеме ASMD на рис. 4 можно записать следующий код FSMD на языке Verilog проекта mult_c_Mealy_FSMD синхронного умножителя, реализующего алгоритм умножения c:

```
module mult_c_Mealy_FSMD
       #(parameter N=4)                    // размер операндов
       (input clk,reset,run,               // сигналы управления
       input [N-1:0] a,b,          // множимое a и множитель b
       output reg [2*N-1:0] p,     // произведение p
       output reg done);           // флаг окончания умножения
       reg [2*N-1:0] rp;           // объявление регистров
       reg [N-1:0] ra,rb;
       reg [N:0] cnt;                      // счетчик
       localparam [0:0] s0=0,s1=1; // состояния FSMD
       reg [0:0] state;                    // переменная состояний
       always @(posedge clk, negedge reset)    // описание поведения FSMD
              if(!reset) state <= s0;    // сброс FSMD
              else
                     case (state)
                     s0:    if(run) begin// описание первого блока ASMD
                                   rp = 0; cnt <= 0; done <= 0;
                                   ra <= a;
                                   rb <= b;
                                   state <= s1;
                            end
                            else   state <= s0;
                     s1:    begin              // описание второго блока ASMD
                                   if(rb[N-1])rp = rp + ra;
                                   cnt <= cnt + 1'b1;
                                   rb <= rb << 1;

                                   if (cnt == N-1)     begin
                                          done <= 1'b1;
                                          p <= rp;
                                          state <= s0;
                                   end
                                   else begin
                                          rp = rp << 1;
                                          state <= s1;
                                   end
                            end
                     default: state <= s0;
                     endcase
endmodule
```

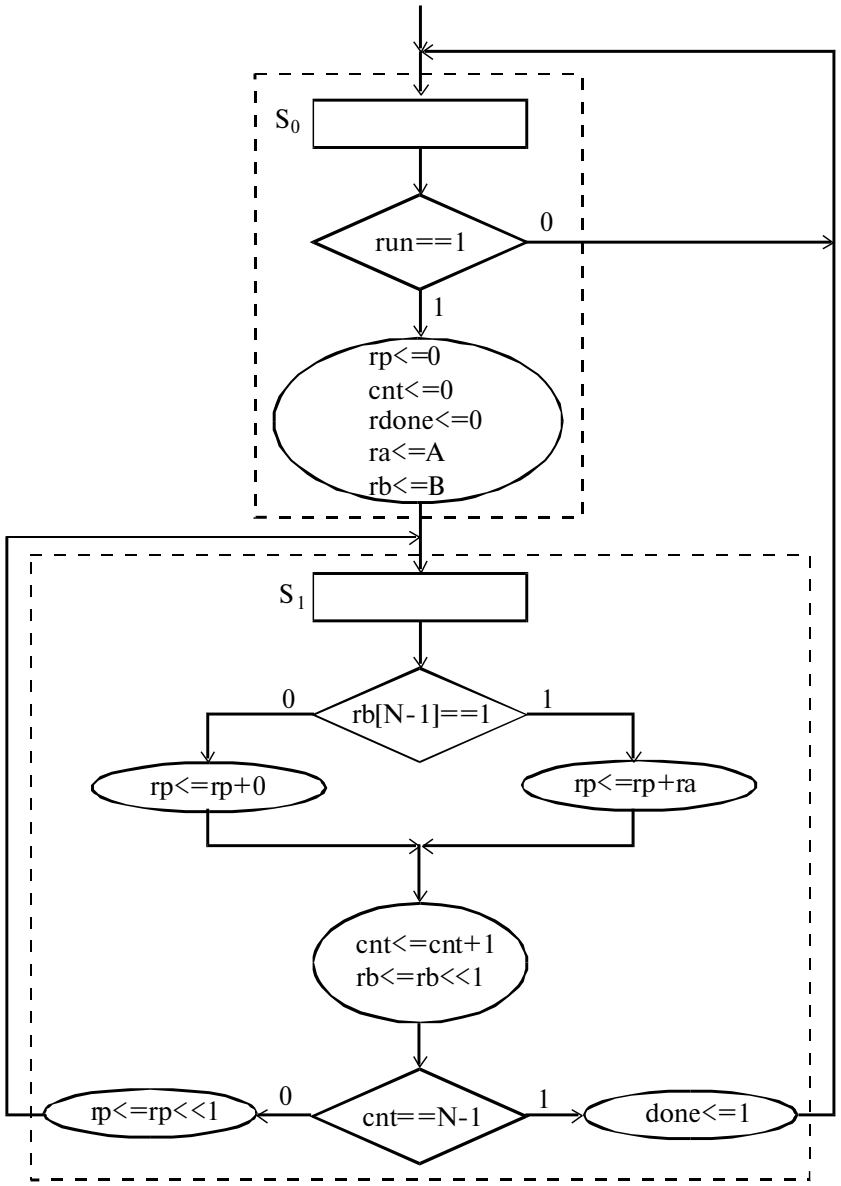


Рис. 4. Схема ASMD алгоритма умножения *c*. Результаты моделирования проекта mult_c_Mealy_FSMD в системе Quartus Prime приведены на рис. 5. Видно, что умножение 4-битовых чисел действительно выполняется за пять тактов синхросигнала *clk*. Для сравнения на рис. 6 приведены результаты моделирования синхронного умножителя, реализующего алгоритм умножения *c*, который был спроектирован по традиционной технологии. Как видно, умножение 4-битовых чисел выполняется за девять тактов синхронизации.

Таким образом, наша гипотеза – использование операторов неблокирующего назначения языка Verilog в методике ASMD-FSMD позволяет выполнить несколько операций с одной и той же переменной за один такт синхронизации и увеличить быстродействие устройства – полностью подтвердилась.

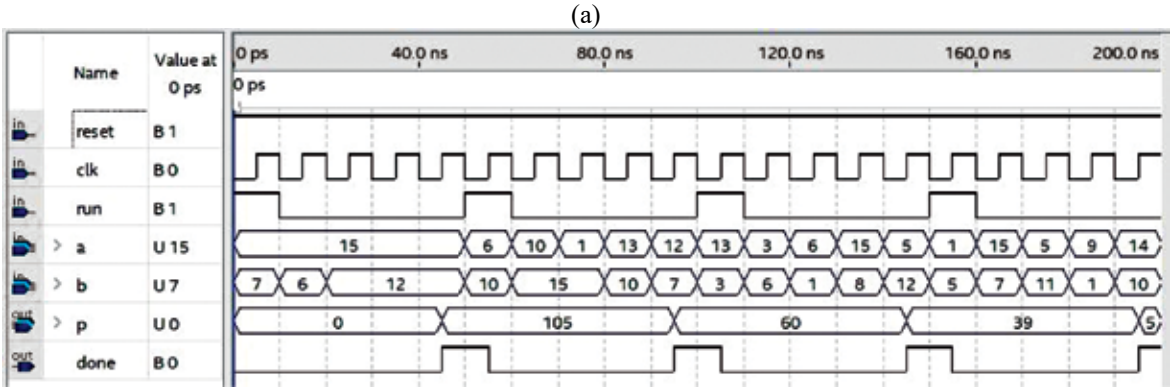
Отметим, что возможность выполнения нескольких операций с одной и той же переменной за один такт синхронизации объясняется особенностями реализации компилятором языка Verilog операторов неблокирующего назначения.

4. ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ

Исследование эффективности методики ASMD-FSMD проводили при реализации на FPGA семейства Cyclone IV E с помощью системы Quartus Prime версии 18.1 алгоритмов умножения *c* и *d*. При описании алгоритма *d* схемой ASMD в одном блоке ASMD выполняется добавление к содержимому регистра rp значения регистра ra, а также сдвиг вправо регистра rp.

Алгоритмы умножения *c* и *d* были реализованы с помощью традиционного подхода в виде

(а)



(б)

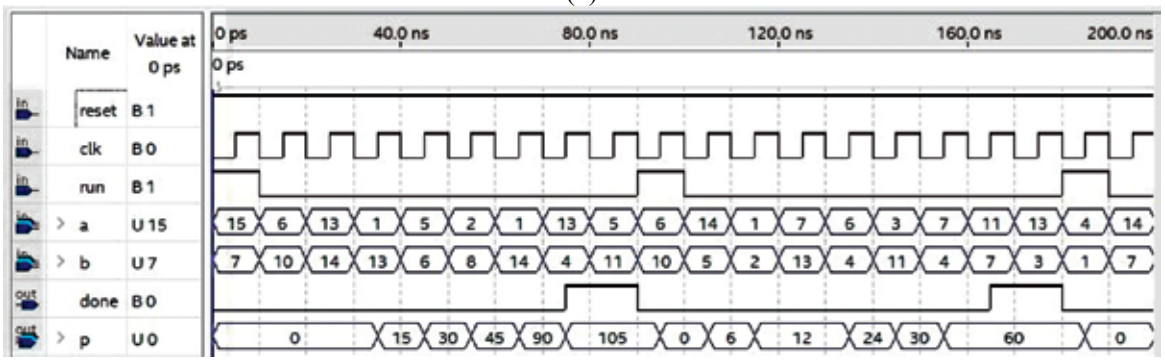


Рис. 5. Результаты моделирования умножителя, спроектированного по методике ASMD-FSMD (а) и построенного по традиционной методике (б).

операционного устройства и устройства управления, а также с помощью методики ASMD-FSMD. Проекты исследовались с шириной входных слов 4, 8, 16, 32, 64 и 128 битов. Результаты экспериментальных исследований приведены в табл. 1.

Анализ табл. 1 показывает, что использование методики ASMD-FSMD позволяет увеличить быстродействие алгоритма *c* в среднем в 1.96 раза (для примера, mult_c_4 в 1.99 раза) и алгоритма *d* в 2.45 раза (для примера, mult_c_16 в 2.96 раза). Кроме того, методика ASMD-FSMD FSMD позволяет для алгоритма *d* уменьшить стоимость реализации в среднем в 1.23 раза (для примера, mult_c_16 в 1.31 раза). Увеличение быстродействия в результате применения методики ASMD-FSMD можно также видеть на рис. 6.

Методику ASMD-FSMD также исследовали с помощью системы Vivado версии 2018.2 на FPGA семейства Kintex UltraScale при реализации алгоритмов умножения *c* и *d*. Результаты исследований приведены в табл. 2.

Анализ табл. 2 показывает, что использование методики ASMD-FSMD позволяет для алгоритма *c* увеличить быстродействие в среднем в 1.98 раза (для примера, mult_c_128 в 2.05 раза) и уменьшить стоимость реализации в 1.18 раза (для примера, mult_c_64 в 1.56 раза). Для алгоритма *d* быстродействие увеличивается в среднем в 2.10 раза (для примера, mult_d_4 в 2.38 раза), а стоимость реализации уменьшается в 1.59 раза (для примера, mult_d_128 в 1.96 раза). Увеличение быстродействия и уменьшение стоимости реализации в результате применения методики ASMD-FSMD в системе Vivado можно также видеть на рис. 7 и 8 соответственно.

Таким образом, справедливость выдвинутой гипотезы подтвердилась результатами экспериментальных исследований для компиляторов двух популярных систем проектирования: Quartus фирмы Intel и Vivado фирмы Xilinx. При этом наблюдается не только увеличение быстродействия, но также в ряде случаев уменьшение стоимости реализации. С большой вероятностью можно ожидать, что использование методики ASMD-FSMD будет также эффективно и для других систем проектирования, а также для других языков проектирования, например, VHDL.

ЗАКЛЮЧЕНИЕ

Таким образом, в рамках применения методики ASMD-FSMD к реализации синхронных умножителей исследована возможность использования в одном блоке ASMD нескольких

Таблица 1. Результаты исследования реализации алгоритмов умножения *c* и *d* в системе Quartus

Пример	Параметр					
	L_T	L_A	L_T/L_A	t_T	t_A	t_T/t_A
mult_c_4	35	35	1.00	40.25	20.22	1.99
mult_c_8	60	65	0.92	93.11	47.82	1.95
mult_c_16	112	138	0.81	183.55	97.72	1.88
mult_c_32	208	270	0.77	452.46	238.75	1.98
mult_c_64	405	540	0.75	1267.69	639.39	1.98
mult_c_128	793	1006	0.79	8216.11	4200.59	1.96
mult_d_4	32	29	1.10	41.77	20.67	2.02
mult_d_8	58	48	1.21	82.48	36.41	2.27
mult_d_16	98	75	1.31	159.26	53.86	2.96
mult_d_32	176	141	1.25	410.87	158.33	2.59
mult_d_64	338	271	1.25	940.51	355.62	2.64
mult_d_128	663	528	1.26	4851.80	2178.69	2.23

Примечания: mult_c_N и mult_d_N – проекты, реализующие алгоритмы умножения *c* и *d* соответственно; *N* – ширина входных слов умножителей в битах; L_T и L_A – число используемых логических элементов FPGA (стоимость реализации) в случае традиционного подхода и при использовании методики ASMD-FSMD соответственно; t_T и t_A – время выполнения операции умножения в наносекундах в случае традиционного подхода и при использовании методики ASMD-FSMD; L_T/L_A и t_T/t_A – отношения соответствующих параметров.

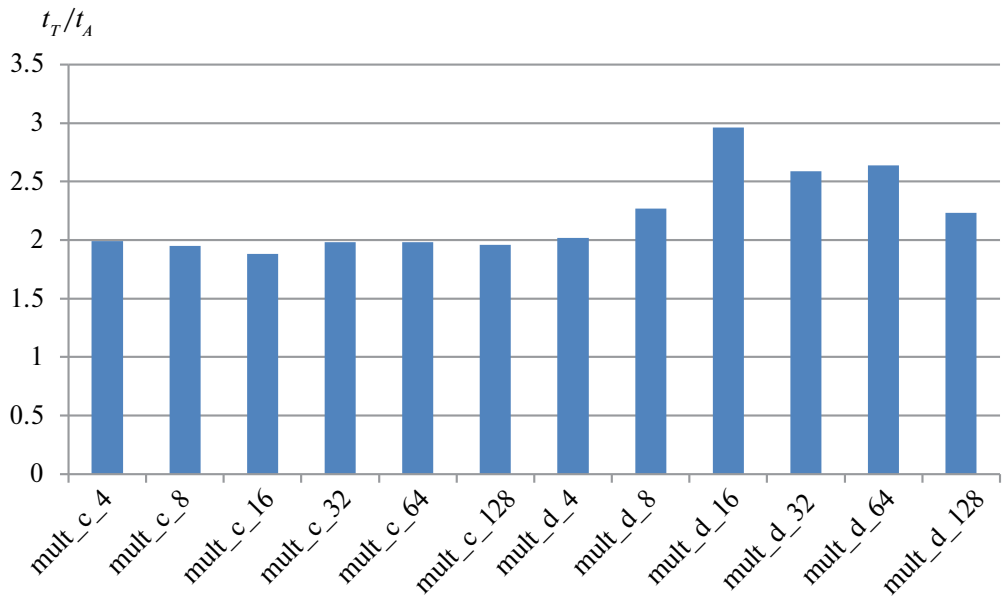


Рис. 6. Коэффициенты (число раз) увеличения быстродействия для каждого примера в системе Quartus.

операторов неблокирующего назначения языка Verilog к одной и той же переменной с целью увеличения быстродействия.

Показано, что в одном блоке ASMD можно применять несколько операторов неблокирующего назначения к одной и той же переменной. При этом все действия в одном блоке ASMD выполняются за один такт синхронизации, что приводит к значительному увеличению быстродействия устройства, а в некоторых случаях и к уменьшению стоимости реализации.

Экспериментальные исследования показали, что применение методики ASMD-FSMD для реализации синхронных умножителей позволило увеличить быстродействие в два-три раза и в большинстве случаев уменьшить стоимость реализации, в некоторых случаях почти в два раза.

Таблица 2. Результаты исследования реализации алгоритмов умножения *c* и *d* в системе Vivado

Пример	Параметр					
	L_T	L_A	L_T/L_A	t_T	t_A	t_T/t_A
mult_c_4	19	23	0.83	74.56	38.30	1.95
mult_c_8	35	33	1.06	138.79	68.59	2.02
mult_c_16	66	65	1.02	275.55	145.52	1.89
mult_c_32	132	90	1.47	589.23	299.41	1.97
mult_c_64	267	171	1.56	1241.75	629.66	1.97
mult_c_128	528	462	1.14	2537.88	1239.05	2.05
mult_d_4	20	13	1.54	89.49	37.54	2.38
mult_d_8	36	24	1.50	157.24	69.26	2.27
mult_d_16	65	41	1.59	305.59	145.28	2.10
mult_d_32	124	90	1.38	619.21	307.82	2.01
mult_d_64	267	170	1.57	1216.07	629.85	1.93
mult_d_128	525	268	1.96	2466.98	1288.97	1.91

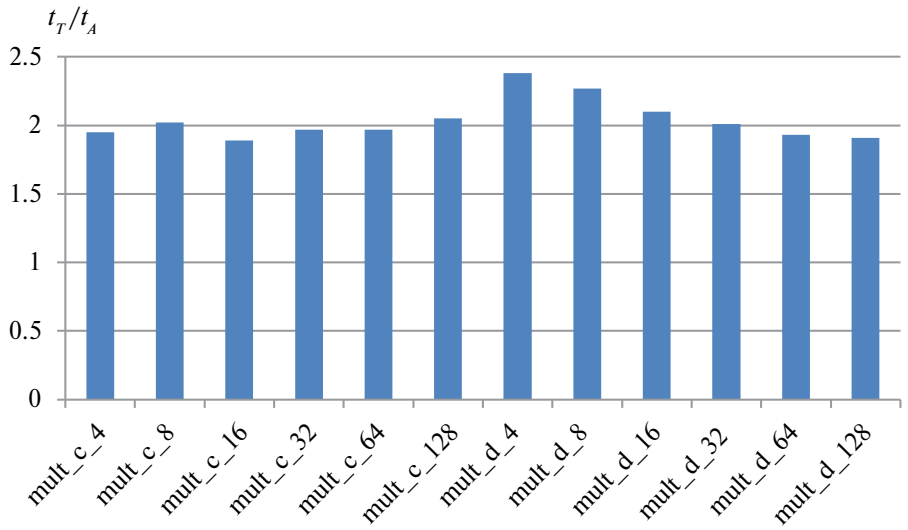


Рис. 7. Коэффициенты (число раз) увеличения быстродействия для каждого примера в системе Vivado.

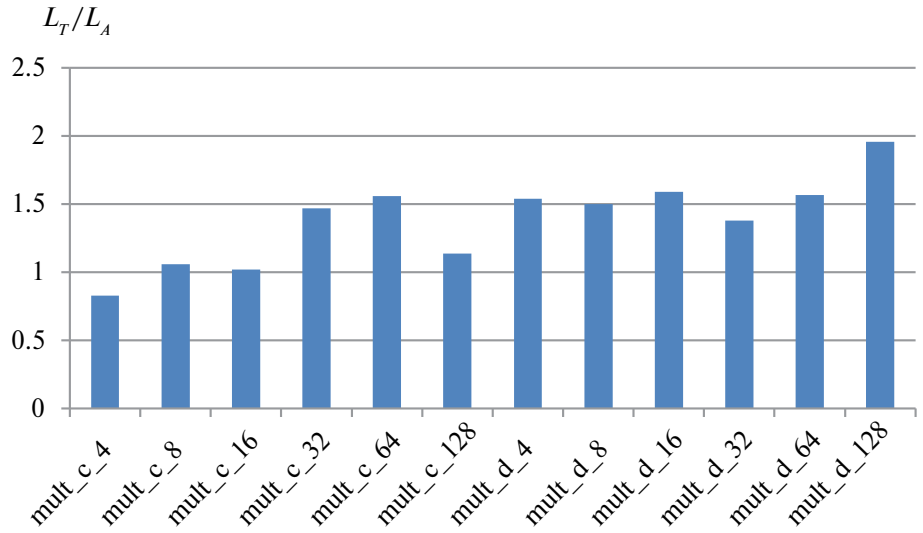


Рис. 8. Коэффициенты (число раз) уменьшения стоимости реализации для каждого примера в системе Vivado.

Авторы заявляют об отсутствии конфликта интересов.

ФИНАНСИРОВАНИЕ РАБОТЫ

Работа выполнена при частичной финансовой поддержке Белостокского технологического университета (Белосток, Польша, грант WZ/WI-III/5/2023).

СПИСОК ЛИТЕРАТУРЫ

1. *Gajski D.D., Dutt N.D., Wu A.C., Lin S.Y.* High-Level Synthesis: Introduction to Chip and System Design. Boston: Kluwer, 1992.
2. *Auletta R., Reese B., Traver C.* // Proc. Int. Conf.on Computer Design ICCD'93. Cambridge (MA). 3–6 Oct. 1993. N.Y.: IEEE, 1993. P. 178.
3. *Karfa C., Sarkar D., Mandal C.* // IEEE Trans.2010. V. CAD-29. №. 3. P. 479.
4. *Hu J., Wang G., Chen G., Wei X.* // IEEE Access. 2019. V. 7. P. 183435.
5. *Schaumont P., Shukla S., Verbauwhede I.* // Proc.Design Automation & Test in Europe Conf. Verona. 11–14 Jul. 2005. N.Y.: IEEE, 2006. V. 1. P. 6.
6. *Zhu J., Gajski D.D.* // Proc. 7th Int. Workshop on Hardware/Software Codesign CODES'99. Rome. 3 Mar. 1999. N.Y.: IEEE, 1999. P. 121.
7. *Kavvadias N., Masselos K.* // Proc. Int. Conf. onApplication-Specific Systems, Architectures and Processors. Delft. 9–11 Jul. 2012. N.Y.: IEEE, 2012. P. 157.
8. *Banerjee K., Sarkar D., Mandal C.* // IEEE Trans. 2014. V. CAD-33. № 12. P. 2015.
9. *Hwang E., Vahid F., Hsu Y.C.* // Proc. Int. Conf. on Design, Automation and Test in Europe. Munich. 9–12 Mar. 1999. P. 7.
10. *Abdullah A.C., Ooi C.Y., Ismail N.B., Mohammed N.B.* // Proc. Int. Symp. On Circuits andSystems (ISCAS). Montreal. 22–25 May 2016. N.Y.: IEEE, 2016. P. 1942.
11. *Babakov R., Barkalov A., Titarenko L.* // Proc. Int. Conf. on The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM). Lviv. 21–25 Feb. 2017. N.Y.: IEEE, 2017. P. 203.
12. *Clare C.R.* Designing logic systems using state machines. N.Y.: McGraw-Hill Book Company, 1973.
13. *Green D.H., Chughtai M.A.* // IEE Proc. E-Computers and Digital Techniques. 1986. V. 133. № . 4. P. 194.
14. *Baranov S.* // Proc. Int. Conf. EUROMICRO.Vasteras. 27–27 Aug. 1998. N.Y.: IEEE, 1998. V. 1. P. 176.
15. *Jenihhin M., Baranov S., Raik J., Tihhomirov V.*//Proc. Int. Conf. Latin American Test Workshop (LATW). Quito. 10–13 Apr. 2012. N.Y.: IEEE. 2012. P. 1.
16. *Ciletti M.D.* Advanced digital design with the Verilog HDL. New Delhi: Prentice Hall of India, 2005.
17. *Martín P., Bueno E., Rodríguez F.J., Sáez V.* // Proc. Annual Conf. IEEE Industrial Electronics. Porto.3–5 Nov. 2009. N.Y.: IEEE. P. 2811.
18. *Saha A., Ghosh A., Kumar K.G.* // Proc. Int. Conf. on Advances in Science and Technology. Bangkok.19–22 Jan. 2017. Bangkok: Elsevier, 2017. P. 138.
19. *Burciu P.* // J. Electrical Engineering, Electronics, Control and Computer Science. 2019. V. 5. № . 3. P. 1.
20. *Sowmya K.B., Shreyans G., Vishnusai R.T.* // Proc.Int. Conf. on Communication and Electronics Systems. Coimbatore. 10–12 Jun. 2020. N.Y.: IEEE, 2020. P. 176.
21. *Salauyou V.* // Proc. Int. Conf. on Dependabilityand Complex Systems. Wroclaw, Poland, June 28 – July 2. Cham: Springer, 2021. P. 391.
22. *Salauyou V., Klimowicz A.* // Proc. Int. Conf. on Computer Information Systems and Industrial Management. Elk, Poland, 24–26 Sept. 2021. Cham: Springer, 2021. P. 431.
23. *Соловьев В.В.* // РЭ. 2021. Т. 66. № 12. С. 1178.
24. *Соловьев В.В.* Язык Verilog в проектировании встраиваемых систем на FPGA. М.: Горячая линия–Телеком, 2020.
25. *Соловьев В.В.* Основы языка проектирования цифровой аппаратуры Verilog. 2-е изд. М.: Горячая линия–Телеком, 2021.

INCREASING THE SPEED OF DIGITAL DEVICES USING THE ASMD-FSMD METHOD USING NON-BLOCKING OPERATORS

V. V. Solov'ev*, A. S. Klimovicz

Bialystok University of Technology, Bialystok, 15-351 Poland
**E-mail: valsol@mail.ru*

Received March 6, 2023; revised April 5, 2023; accepted April 27, 2023

The ASMD-FSMD method for designing digital devices is considered, which consists of constructing a block diagram of an algorithmic state machine with a data path (ASMD), which describes the behavior of the device, and creating project code in Verilog in the form of a finite state machine with a data processing path. (finite state machine with datapath – FSMD). One of the directions for the development of the ASMD-FSMD methodology is the use of features of the hardware description language (HDL). A hypothesis has been put forward: in the ASMD-FSMD technique, it is possible to apply several non-blocking assignment operators to the same variable in one synchronization cycle, which will lead to an increase in device performance. The hypothesis put forward was investigated in the design of synchronous multipliers that implement the classical multiplication algorithms c and d. Experimental studies have confirmed the validity of the put forward hypothesis, while the speed of the multipliers increases two to three times, and the cost of implementation in most cases decreases compared to the traditional approach.

Keywords: algorithmic state machinewith a data processing path, finite state machine with a data processing path, hardware description language, non-blocking assignment operators, synchronous multipliers