

## ТЕОРИЯ И МЕТОДЫ ОБРАБОТКИ СИГНАЛОВ

УДК 621.3;004.3

# ИСПОЛЬЗОВАНИЕ МЕТОДИКИ ASMD-FSMD ПРИ ПРОЕКТИРОВАНИИ НА ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ ИНТЕГРАЛЬНЫХ СХЕМАХ УСТРОЙСТВ ОБРАБОТКИ СИГНАЛОВ

© 2021 г. В. В. Соловьев\*

Белорусская государственная академия связи,  
ул. Ф. Скорины, 8/2, Минск, 220076 Республика Беларусь

\*E-mail: valsol@mail.ru

Поступила в редакцию 20.03.2021 г.

После доработки 07.04.2021 г.

Принята к публикации 10.04.2021 г.

Рассмотрена методика ASMD-FSMD проектирования цифровых устройств на основе конечных автоматов с трактом обработки данных (finite state machine with datapath, FSMD), когда функционирование устройства описывается в виде блок-схемы автомата с трактом обработки данных (algorithmic state machine with datapath, ASMD). Приведено сравнение различных методик проектирования цифровых устройств на примерах проектирования синхронных умножителей и процессоров PIC на программируемых логических интегральных схемах (ПЛИС — field programmable gate array, FPGA). Показано, что методика ASMD-FSMD по сравнению с традиционной позволяет в большинстве случаев уменьшить стоимость реализации (для отдельных примеров на 47%) и заметно увеличить быстродействие (для отдельных примеров в 2.96 раза), а также значительно сократить время проектирования (приблизительно в пять–семь раз). Даны рекомендации по использованию методики ASMD-FSMD и указаны возможные направления ее дальнейшего развития.

DOI: 10.31857/S0033849421120184

## ВВЕДЕНИЕ

В последнее время наблюдается возрастание сложности систем цифровой обработки сигналов с одновременным ужесточением требований к срокам разработки и повышению надежности проектов. Одним из направлений решения указанной проблемы является внедрение в практику инженерного проектирования новых методик проектирования цифровых устройств.

Традиционно проектируемое цифровое устройство принято представлять в виде операционного устройства (datapath) и устройства управления (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] 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 [2] приведены варианты 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 работе [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 предложены расширения FSMD: в [6] — для представления архитектуры процессора и ASIC (application-specific integrated circuit); в [7] — для синхронного доступа к памяти; в [8] — для программ обработки массивов. Уменьшение потребляемой мощности путем декомпозиции FSMD рассмотрено в [9], а путем стробирования — в [10]. Сравнение эффективности FSM и FSMD приведено в [11].

Благодаря своей наглядности блок-схемы автоматов (algorithmic state machine, ASM) получили широкое распространение для представления алгоритмов функционирования конечных автоматов. ASM впервые были предложены в [12] как альтернатива графам автоматов. В работе [13] рассмотрена реализация ASM на PROM, FPLA и мультиплексорах, а в [14] предложены методы минимизации числа вершин ASM. В [15] описан

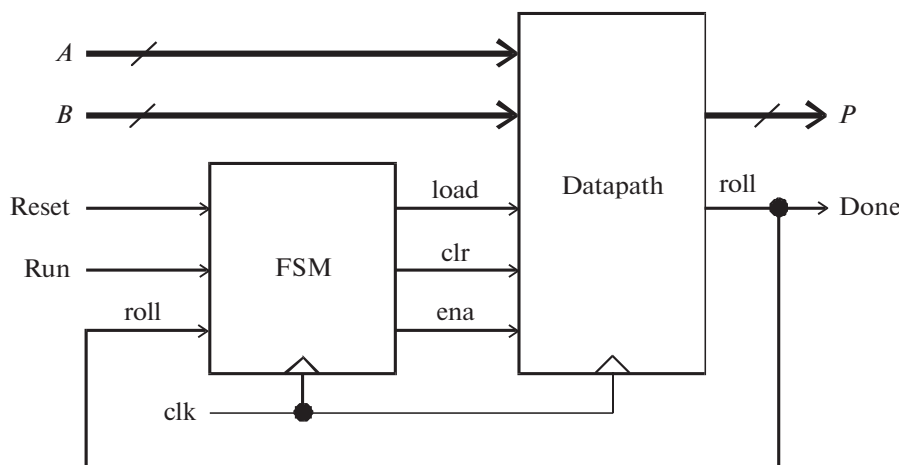


Рис. 1. Структурная схема синхронного умножителя в виде операционного устройства и устройства управления.

инструмент ABELITE синтеза контроллеров, основанных на ASM.

Традиционно ASM используют для представления алгоритма функционирования устройства управления. В [16] предложено использовать ASM как для описания поведения устройства управления, так и для описания регистровых 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 применяют в проектах на FPGA: при реализации промышленных систем управления [17]; для реализации функции  $\text{asin}$  с помощью алгоритма CORDIC [18]; при аппаратной реализации криптографического алгоритма AES [19]; при проектировании универсального асинхронного приема-передатчика UART [20].

В данной работе исследована основанная на языке Verilog методика ASMD-FSMD при проектировании синхронных умножителей и процессоров PIC на программируемых логических интегральных схемах (ПЛИС — field programmable gate array, FPGA). В методике ASMD-FSMD функционирование устройства представляется в виде схемы ASMD, которая на языке Verilog описывается в виде конечного автомата с трактом обработки данных.

## 1. ТРАДИЦИОННЫЙ ПОДХОД К ПРОЕКТИРОВАНИЮ ЦИФРОВЫХ УСТРОЙСТВ

Рассмотрим простейший школьный алгоритм умножения, выполняющий арифметическую операцию умножения  $P = A \times B$  двух двоичных чисел шириной  $N$  битов. Вначале  $P$  обнуляют. В каждом цикле умножения проверяют значение младшего

разряда множителя  $B[0]$ . Если  $B[0] = 1$ , то к произведению  $P$  прибавляется множимое  $A$ . Если  $B[0] = 0$ , то к произведению  $P$  прибавляют ноль или ничего не прибавляют. Затем содержимое множителя  $B$  сдвигают на один разряд вправо, а значение произведения  $P$  — влево. Алгоритм заканчивается после рассмотрения всех битов множителя  $B$ .

На рис. 1 представлена структурная схема синхронного умножителя в виде операционного устройства (Datapath) и устройства управления (FSM). Значения чисел  $A$  и  $B$  поступают на вход операционного устройства. На выходах операционного устройства формируется произведение  $P$  и сигнал  $\text{done}$ , указывающий на окончание процесса умножения. Кроме того, операционное устройство формирует внутренний сигнал  $\text{roll}$ , совпадающий с сигналом  $\text{done}$ . Устройство управления FSM формирует следующие управляющие сигналы:  $\text{load}$  — для загрузки в регистры операционного устройства значений умножаемых слов  $A$  и  $B$ ;  $\text{clr}$  — для сброса регистров операционного устройства;  $\text{ena}$  — для разрешения операции сдвига содержимого сдвиговых регистров.

Схема операционного устройства, реализующего алгоритм умножения, показана на рис. 2. Операционное устройство включает сдвиговый регистр влево  $ga$ , сдвиговый регистр вправо  $gb$  и регистр  $gr$  для хранения слов  $A$ ,  $B$  и  $P$  соответственно, шинный мультиплексор 2-1 и сумматор на  $2N$  разрядов, а также счетчик по модулю  $\text{count}$ , который формирует сигнал  $\text{roll}$ .

Функционирование устройства управления синхронного умножителя можно представить в виде автомата типа Мили (рис. 3а) или автомата типа Мура (рис. 3б). Отметим, что конечный автомат типа Мили содержит два состояния:  $S_0$  и  $S_1$ , в то время как автомат типа Мура содержит три состояния:  $S_0$ ,  $S_1$  и  $S_2$ . Детали описания компо-

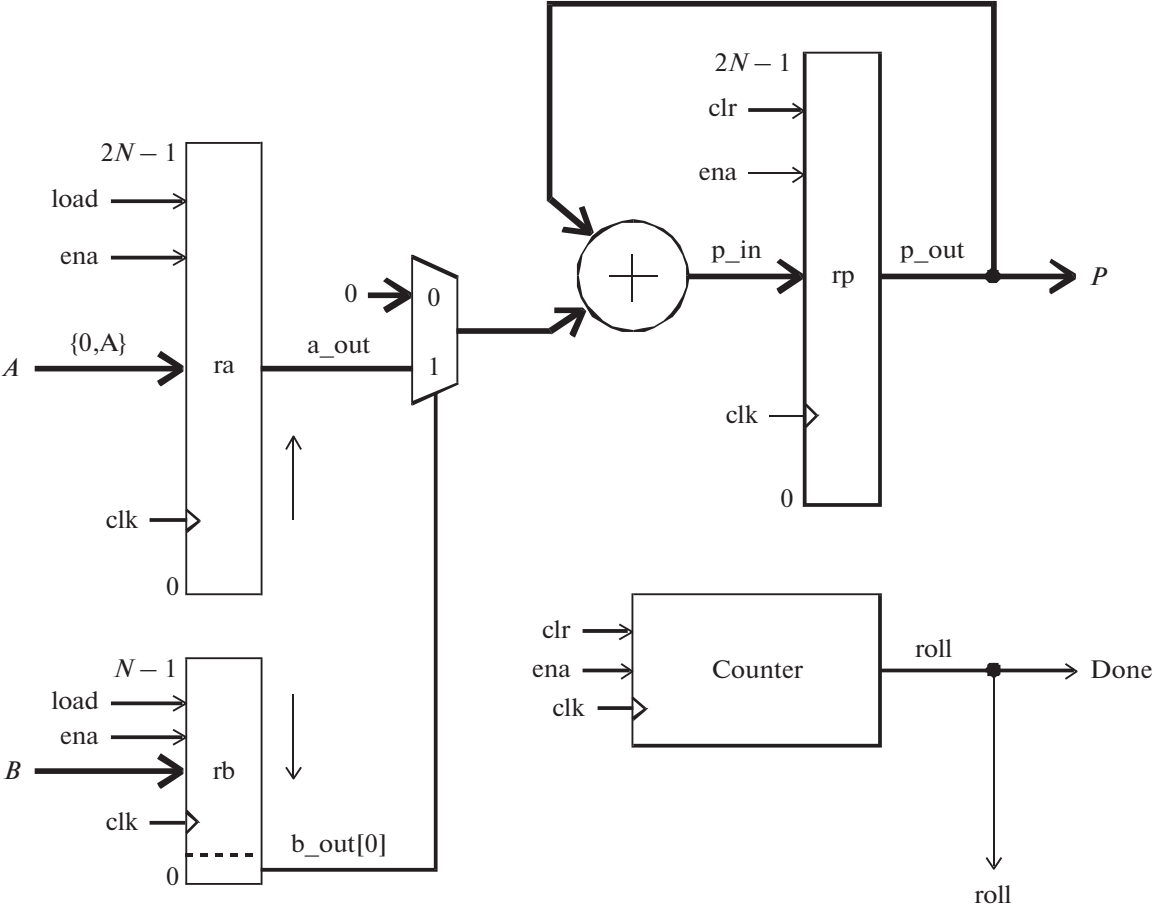


Рис. 2. Схема операционного устройства синхронного умножителя.

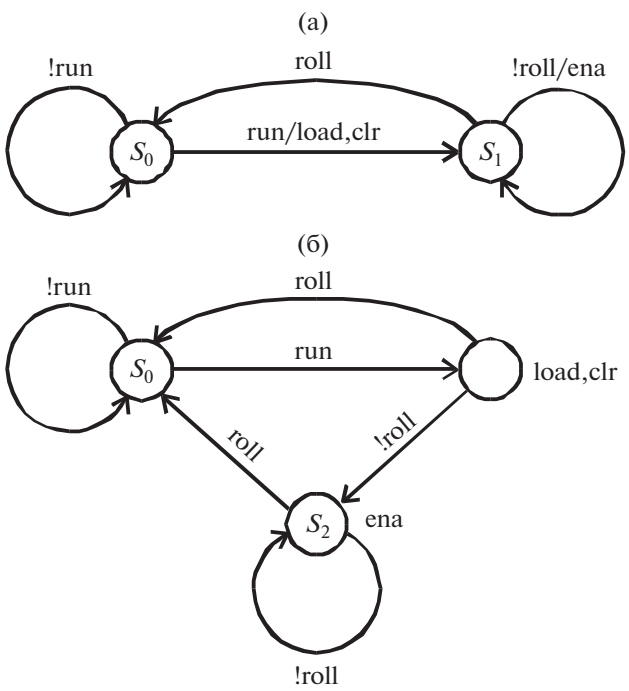


Рис. 3. Представление устройства управления синхронного умножителя в виде графа конечного автомата: а – типа Мили, б – типа Мура.

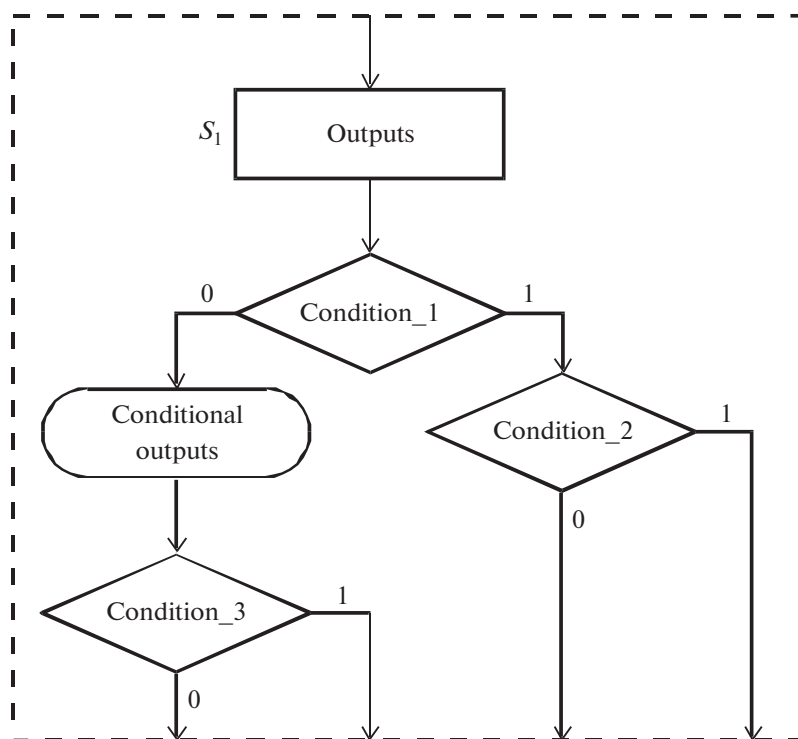


Рис. 4. Блок ASM.

нентов операционного устройства (см. рис. 2) и конечных автоматов Мили и Мура (см. рис. 3) на языке Verilog при реализации синхронного умножителя в случае традиционного подхода приведены в [21].

## 2. БЛОК-СХЕМЫ АВТОМАТОВ ASM

Блок-схема автомата (ASM) предназначена для наглядного описания алгоритма функционирования конечного автомата и представляет собой ориентированный связный граф [12], содержащий вершины трех типов:

- 1) прямоугольники — вершины состояний (state box);
- 2) ромбы — условные вершины (decision box);
- 3) овалы — вершины выходов по условию (conditional output box).

Вершина состояния ASM (прямоугольник) определяет состояние автомата. Вблизи вершины состояния может записываться имя состояния (например, S0, START, INITIAL и др.), а также двоичный код состояния. В случае автомата Мура внутри вершины состояния записывают выходные сигналы, принимающие единичное значение в данном состоянии. По умолчанию полагается, что все остальные выходные сигналы в данном состоянии имеют нулевое значение.

В условных вершинах ASM (ромбах) записывают проверяемые условия, условная вершина ASM представляет собой точку ветвления алгоритма. Выходы условной вершины обозначают значениями 0 и 1, которые соответствуют переходам в случае нулевого (ложного) или единичного (истинного) значения результата проверки условия. В качестве условия может выступать входная переменная конечного автомата, логическое выражение, единичный разряд битового вектора и др.

В вершинах выходов по условию (овалах) записывают выходные сигналы автомата Мили, принимающие единичное значение на определенном переходе. В ASM для автоматов Мура отсутствуют вершины выходов по условию (овалы), а в ASM для автоматов Мили в вершинах состояний ничего не записывают.

Главным строительным элементом блок-схемы ASM является *блок ASM* (ASM block), пример которого показан на рис. 4. Блок ASM описывает поведение автомата в одном состоянии в течение одного такта синхронизации. Поэтому блок ASM включает только 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. Входы и выходы вершин соединяют с помощью дуг.

Блок ASM имеет только один вход, который является входом в вершину состояния, и может иметь один или несколько выходов. Внутри блока ASM запрещены обратные связи. Циклы алгоритма и ждущие вершины в ASM реализуют с помощью внешних обратных связей.

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

Отметим некоторые характерные особенности, заложенные в природе ASM. Он сохраняет наглядность блок-схем алгоритмов. В ASM явно определяют внутренние состояния автомата с помощью блоков ASM. Переход от одного блока ASM к другому всегда выполняется за один такт синхронизации, т.е. алгоритм функционирования автомата непосредственно привязан к автоматному времени. Благодаря этому по ASM всегда можно проследить, за сколько тактов будет выполнен тот или другой фрагмент алгоритма.

### 3. МЕТОДИКА ASMD-FSMD

Блок-схема ASM с трактом обработки данных (ASM with datapath, ASMD) представляет собой ASM, в которой в прямоугольниках и овалах можно записывать любые операции над регистрами, которые допустимы в языке Verilog, а в условных вершинах можно проверять любые логические выражения языка Verilog. Схема ASMD, так же как ASM, состоит из блоков. Действия, описанные внутри блока ASMD, выполняются в течение одного такта синхронизации. Реализуемый по ASMD конечный автомат называется конечным автоматом с трактом обработки данных (FSM with datapath, FSMD). Методику ASMD-FSMD представим в виде следующего алгоритма.

*Алгоритм.* Методика ASMD-FSMD проектирования цифровых устройств.

*Шаг 1.* Определяют состояния FSMD.

*Шаг 2.* Для каждого состояния строят блок ASMD.

*Шаг 2.1.* В условных вершинах ASMD записывают логические функции, значения которых проверяют в данном состоянии.

*Шаг 2.2.* Для FSMD Мура в вершинах состояний (прямоугольниках) записывают операции, выполняемые с содержимым регистров в данном состоянии.

*Шаг 2.3.* Для FSMD Мили в вершинах выходов по условию (овалах) записывают операции, выполняемые с содержимым регистров на данном переходе.

*Шаг 3.* Блоки ASMD соединяют между собой в соответствии с алгоритмом работы устройства. При этом каждый выход блока ASMD может быть соединен только с одним входом данного или другого блока ASMD.

*Шаг 4.* При необходимости выполняют оптимизацию ASMD.

*Шаг 5.* Непосредственно по ASMD строят код FSMD на языке Verilog. Переменным ASMD в коде соответствуют регистры или триггеры (для однобитных переменных). Логическим функциям, проверяемым в условных вершинах ASMD, соответствуют логические выражения в операторах if. Действия, выполняемые в блоках ASMD, описывают в виде процедурных блоков begin...end. Операции, выполняемые в прямоугольниках ASMD (для автоматов Мура), описывают в начале блока begin...end, а операции, выполняемые в ромбах (для автоматов Мили), описывают в соответствующих местах операторов if (возможно с использованием операторных скобок begin...end).

*Шаг 6.* Выполняют реализацию FSMD с помощью соответствующего средства проектирования.

*Шаг 7.* Конец.

Главным этапом методики ASMD-FSMD является построение ASMD на основе алгоритма функционирования устройства. Представление функционирования цифрового устройства в виде ASMD больше напоминает алгоритмическое описание устройства [21]. Однако в отличие от алгоритмического описания в ASMD явно определены состояния FSMD, которые могут соответствовать состояниям устройства управления. В отличие от традиционного подхода, в ASMD отсутствует строгое разделение на операционное устройство и устройство управления, а также в ASMD явно не определяется структура операционного устройства. Кроме того, представленная методика ASMD-FSMD позволяет реализовывать как автоматы типа Мили, так и автоматы типа Мура, а также совмещенную модель автоматов Мили и Мура (в [16] рассмотрена ASMD только для автоматов типа Мура).

Отметим также, что один и тот же алгоритм функционирования устройства может быть описан различными ASMD, что влияет на быстродействие и стоимость реализации проекта. Для увеличения быстродействия циклы алгоритма следует описывать с минимальным числом состояний, чтобы минимизировать число состояний в пути цикла. Для этого лучше подходят автоматы типа Мили.

На рис. 5 представлена ASMD, которая соответствует FSMD типа Мили для реализации рассматриваемого синхронного умножителя. Особенностью схемы ASMD на рис. 5 является то, что здесь имеются только два блока ASMD: в состоянии  $S_0$

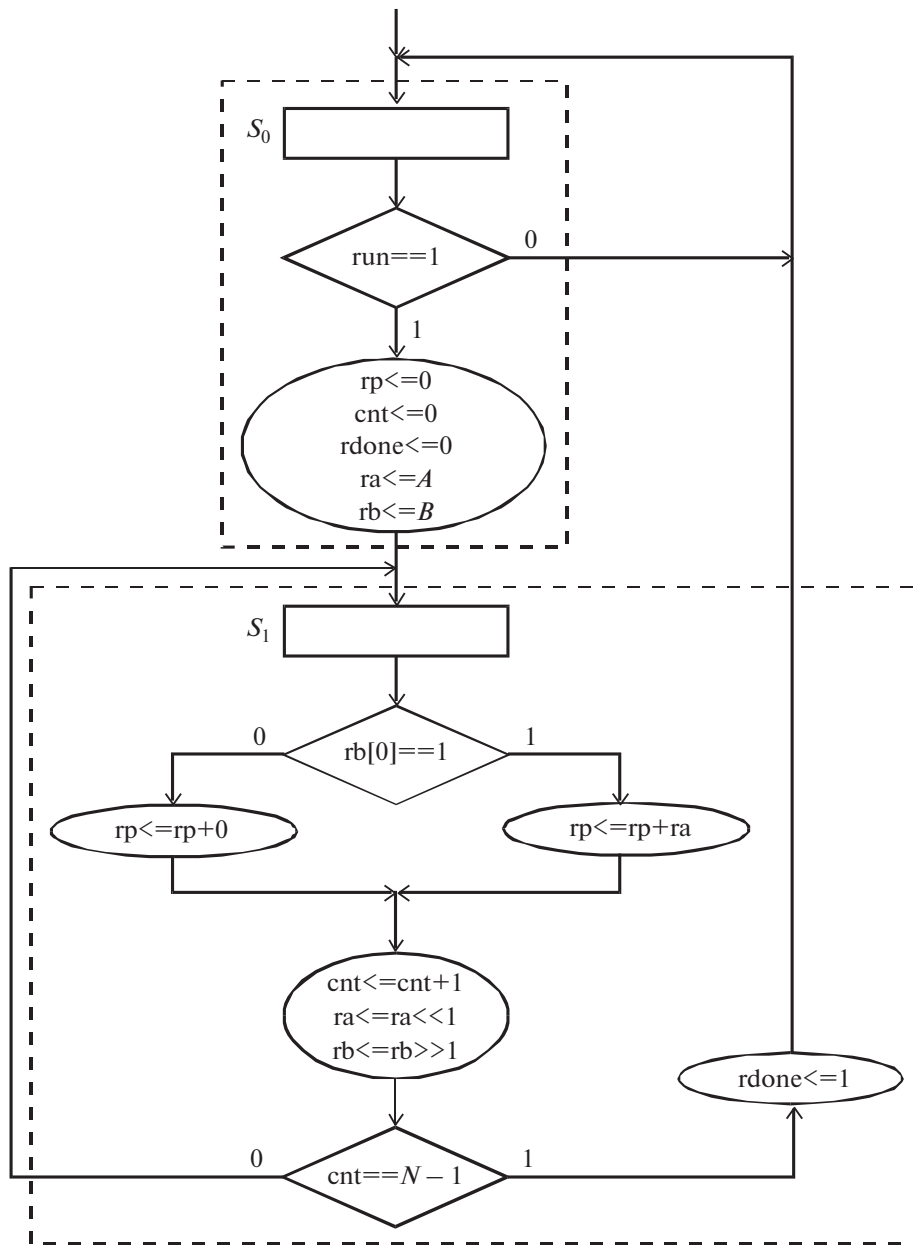


Рис. 5. Схема ASMD для автомата Мили, обеспечивающая наибольшее быстродействие синхронного умножителя.

выполняется инициализация регистров, а состояние  $S_1$  соответствует одному циклу умножения.

Из рисунка видно, что один цикл умножения выполняется за один такт синхронизации, поэтому умножение  $N$ -разрядных двоичных чисел по ASMD на рис. 5 осуществляется за  $n$  тактов синхронизации, где  $n = N + 1$ .

Отметим отличия методики ASMD-FSMD от известных подходов к проектированию цифровых устройств. Методика ASMD-FSMD позволяет:

а) описывать как автоматы типа Мили, так и автоматы типа Мура (в [16] ASMD описывают только автоматы Мура), а также совмещенные модели автоматов типа Мили и Мура;

б) объединять в одной схеме ASMD описание операционного устройства и устройства управления (в [16] ASMD отдельно описывают операционные и управляющие устройства);

в) описывать цифровое устройство на языке Verilog с помощью одного модуля, что значительно сокращает время проектирования и способствует повышению надежности проекта;

г) привлекать к разработке цифровых устройств программистов и математиков-алгоритмистов, которые не знакомы с тонкостями проектирования электронных устройств.

#### 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 известных методов умножения [22]. Были исследованы следующие классические методы умножения:

1) *алгоритм а*, когда проверяется младший разряд множителя, множитель сдвигается вправо, а множимое — влево (рассмотренный выше школьный алгоритм);

2) *алгоритм b*, когда проверяется старший разряд множителя, множитель сдвигается влево, а множимое — вправо;

3) *алгоритм с*, когда проверяется старший разряд множителя, множитель и частные произведения сдвигаются влево;

4) *алгоритм d*, когда проверяется младший разряд множителя, множитель и частные произведения сдвигаются вправо.

Кроме того, был исследован алгоритм умножения Бута и модифицированный алгоритм умножения Бута (всего шесть методов умножения).

Каждый метод умножения был реализован с помощью традиционного подхода в виде операционного устройства и устройства управления, а также с помощью методики ASMD-FSMD (всего 12 проектов). Проекты исследовали с шириной входных слов 4, 8, 16, 32, 64 и 128 битов (72 примера).

Результаты экспериментальных исследований приведены в табл. 1.

Анализ табл. 1 показывает, что использование методики ASMD-FSMD для большинства примеров позволяет уменьшить стоимость реализации и увеличить быстродействие. При этом в отдельных случаях стоимость реализации уменьшается в 1.47 раза, т.е. на 47% (пример *mult\_a\_128*), а быстродействие увеличивается в 2.96 раза (пример *mult\_d\_16*). Эти значения выделены полужирным шрифтом.

Для исследования методики ASMD-FSMD при проектировании более сложных проектов были созданы следующие проекты процессора PIC [21]:

а) *PIC\_1\_cycle\_N* — одноктактовый процессор PIC;

б) *PIC\_multi\_N* — многотактовый процессор PIC;

в) *PIC\_2\_cycle\_N* — двухтактовый процессор PIC;

г) *PIC\_4\_cycle\_N* — четырехтактовый процессор PIC,

где число *N* обозначает ширину шины данных в битах.

Каждый проект был реализован с помощью традиционного подхода и с помощью методики ASMD-FSMD (всего восемь проектов). Проекты процессора PIC исследовали при ширине шины данных *N*, равной 4, 8, 16, 32, 64 и 128 битов (всего 48 примеров). Результаты исследования проектов процессоров PIC приведены в табл. 2.

Анализ табл. 2 показывает, что при проектировании процессоров PIC методика ASMD-FSMD уступает традиционному подходу по стоимости реализации, однако в большинстве случаев выигрывает по быстродействию, для отдельных примеров в 1.46 раза (проект *PIC\_2\_cycle\_128*). Это значение выделено полужирным шрифтом.

Среднеарифметические значения отношений  $L_T/L_A$  и  $t_T/t_A$  для каждого метода умножения представлены в табл. 3.

Из табл. 3 следует, что методика ASMD-FSMD позволяет в среднем уменьшить стоимость реализации для большинства методов умножения (исключение составляют методы *mult\_c* и *mbooth*), а также для всех методов увеличить быстродействие. Особенно заметно увеличение быстродействия для методов *mult\_c* и *mult\_d* (выделено полужирным шрифтом).

Аналогичные средние значения отношений  $L_T/L_A$  и  $F_T/F_A$  для проектов процессоров PIC представлены в табл. 4. Анализ табл. 4 показывает, что использование методики ASMD-FSMD позволяет увеличить быстродействие процессоров PIC в среднем в 1.1 раза (т.е. на 10%), а для двухтактовых процессоров PIC — в 1.26 раза (т.е. на 26%) (выделено полужирным шрифтом). В то же время методика ASMD-FSMD уступает традиционному подходу по стоимости реализации в среднем от 2 до 16%.

Чтобы оценить время проектирования при традиционном подходе и с использованием методики ASMD-FSMD, все проекты создавались одним разработчиком. Время в минутах, затраченное на разработку каждого проекта, приведено в табл. 5.

Анализ табл. 5 показывает, что использование методики ASMD-FSMD позволяет значительно (в среднем в 6.25 раза) уменьшить время разработки проектов. Это связано с тем, что в случае применения методики ASMD-FSMD отпадает необходимость в разработке (написании кода и отладке) каждого компонента операционного устройства, самого операционного устройства, устройства управления, а также главного модуля проекта.

Таблица 1. Результаты исследования реализации алгоритмов умножения

| Пример            | Параметр |       |             |         |         |             |
|-------------------|----------|-------|-------------|---------|---------|-------------|
|                   | $L_T$    | $L_A$ | $L_T/L_A$   | $t_T$   | $t_A$   | $t_T/t_A$   |
| <i>mult_a_4</i>   | 36       | 28    | 1.29        | 24.46   | 17.49   | 1.40        |
| <i>mult_a_8</i>   | 66       | 49    | 1.35        | 46.96   | 31.15   | 1.51        |
| <i>mult_a_16</i>  | 123      | 90    | 1.37        | 80.08   | 84.08   | 0.95        |
| <i>mult_a_32</i>  | 236      | 172   | 1.37        | 219.03  | 191.79  | 1.14        |
| <i>mult_a_64</i>  | 464      | 335   | 1.38        | 598.42  | 582.23  | 1.03        |
| <i>mult_a_128</i> | 919      | 627   | <b>1.47</b> | 4136.21 | 3969.23 | 1.04        |
| <i>mult_b_4</i>   | 36       | 26    | 1.38        | 23.42   | 16.39   | 1.43        |
| <i>mult_b_8</i>   | 64       | 49    | 1.30        | 50.47   | 47.81   | 1.06        |
| <i>mult_b_16</i>  | 121      | 89    | 1.36        | 89.32   | 106.86  | 0.84        |
| <i>mult_b_32</i>  | 234      | 170   | 1.38        | 218.65  | 178.43  | 1.23        |
| <i>mult_b_64</i>  | 461      | 332   | 1.39        | 603.31  | 580.67  | 1.04        |
| <i>mult_b_128</i> | 913      | 653   | 1.40        | 4093.18 | 3916.21 | 1.05        |
| <i>mult_c_4</i>   | 35       | 35    | 1.00        | 40.25   | 20.22   | 1.99        |
| <i>mult_c_8</i>   | 60       | 65    | 0.92        | 93.11   | 47.82   | 1.95        |
| <i>mult_c_16</i>  | 112      | 138   | 0.81        | 183.55  | 97.72   | 1.88        |
| <i>mult_c_32</i>  | 208      | 270   | 0.77        | 452.46  | 238.75  | 1.98        |
| <i>mult_c_64</i>  | 405      | 540   | 0.75        | 1267.69 | 639.39  | 1.98        |
| <i>mult_c_128</i> | 793      | 1006  | 0.79        | 8216.11 | 4200.59 | 1.96        |
| <i>mult_d_4</i>   | 32       | 29    | 1.10        | 41.77   | 20.67   | 2.02        |
| <i>mult_d_8</i>   | 58       | 48    | 1.21        | 82.48   | 36.41   | 2.27        |
| <i>mult_d_16</i>  | 98       | 75    | 1.31        | 159.26  | 53.86   | <b>2.96</b> |
| <i>mult_d_32</i>  | 176      | 141   | 1.25        | 410.87  | 158.33  | 2.59        |
| <i>mult_d_64</i>  | 338      | 271   | 1.25        | 940.51  | 355.62  | 2.64        |
| <i>mult_d_128</i> | 663      | 528   | 1.26        | 4851.80 | 2178.69 | 2.23        |
| <i>booth_4</i>    | 44       | 38    | 1.16        | 37.11   | 30.37   | 1.22        |
| <i>booth_8</i>    | 82       | 69    | 1.19        | 80.73   | 52.54   | 1.54        |
| <i>booth_16</i>   | 152      | 125   | 1.21        | 111.02  | 111.21  | 1.00        |
| <i>booth_32</i>   | 301      | 238   | 1.26        | 263.55  | 228.73  | 1.15        |
| <i>booth_64</i>   | 590      | 464   | 1.27        | 794.75  | 654.96  | 1.21        |
| <i>booth_128</i>  | 1168     | 917   | 1.27        | 4474.04 | 4245.59 | 1.05        |
| <i>mbooth_4</i>   | 65       | 68    | 0.96        | 25.72   | 18.25   | 1.41        |
| <i>mbooth_8</i>   | 122      | 138   | 0.88        | 36.89   | 31.59   | 1.17        |
| <i>mbooth_16</i>  | 236      | 278   | 0.85        | 68.81   | 62.48   | 1.10        |
| <i>mbooth_32</i>  | 467      | 492   | 0.95        | 150.53  | 124.27  | 1.21        |
| <i>mbooth_64</i>  | 925      | 973   | 0.95        | 413.81  | 404.31  | 1.02        |
| <i>mbooth_128</i> | 1842     | 1935  | 0.95        | 2396.53 | 2236.75 | 1.07        |

Примечание: *mult\_a\_N*, *mult\_b\_N*, *mult\_c\_N* и *mult\_d\_N* – проекты, реализующие алгоритмы умножения *a*, *b*, *c* и *d* соответственно; *booth\_N* и *mbooth\_N* – проекты, реализующие алгоритм умножения Бута и модифицированный алгоритм умножения Бута; *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.

**Таблица 2.** Результаты исследования реализации процессоров PIC

| Пример                     | Параметр |       |           |       |       |             |
|----------------------------|----------|-------|-----------|-------|-------|-------------|
|                            | $L_T$    | $L_A$ | $L_T/L_A$ | $F_T$ | $F_A$ | $F_A/F_T$   |
| <i>PIC_1_cycle_4</i>       | 1002     | 1047  | 0.96      | 70.67 | 75.60 | 1.07        |
| <i>PIC_1_cycle_8</i>       | 1653     | 1712  | 0.97      | 66.80 | 72.45 | 1.08        |
| <i>PIC_1_cycle_16</i>      | 2978     | 3012  | 0.99      | 67.85 | 68.02 | 1.00        |
| <i>PIC_1_cycle_32</i>      | 5561     | 5645  | 0.99      | 53.58 | 66.86 | 1.25        |
| <i>PIC_1_cycle_64</i>      | 10804    | 10925 | 0.99      | 52.59 | 56.16 | 1.07        |
| <i>PIC_1_cycle_128</i>     | 21290    | 21543 | 0.99      | 39.02 | 42.29 | 1.08        |
| <i>PIC_multi_cycle_4</i>   | 920      | 1359  | 0.68      | 73.17 | 76.09 | 1.04        |
| <i>PIC_multi_cycle_8</i>   | 1585     | 2160  | 0.73      | 70.64 | 74.43 | 1.05        |
| <i>PIC_multi_cycle_16</i>  | 2913     | 3502  | 0.83      | 62.79 | 71.41 | 1.14        |
| <i>PIC_multi_cycle_32</i>  | 5506     | 6157  | 0.89      | 62.41 | 69.31 | 1.11        |
| <i>PIC_multi_cycle_64</i>  | 10719    | 11304 | 0.95      | 55.22 | 68.29 | 1.24        |
| <i>PIC_multi_cycle_128</i> | 21204    | 21849 | 0.97      | 41.97 | 59.41 | 1.42        |
| <i>PIC_2_cycle_4</i>       | 1009     | 1344  | 0.75      | 70.53 | 79.47 | 1.13        |
| <i>PIC_2_cycle_8</i>       | 1654     | 2143  | 0.77      | 68.25 | 73.03 | 1.07        |
| <i>PIC_2_cycle_16</i>      | 2980     | 3497  | 0.85      | 58.67 | 75.34 | 1.28        |
| <i>PIC_2_cycle_32</i>      | 5596     | 6104  | 0.92      | 52.72 | 69.59 | 1.32        |
| <i>PIC_2_cycle_64</i>      | 10792    | 11300 | 0.96      | 53.26 | 68.61 | 1.29        |
| <i>PIC_2_cycle_128</i>     | 21287    | 21834 | 0.97      | 41.20 | 59.96 | <b>1.46</b> |
| <i>PIC_4_cycle_4</i>       | 998      | 1325  | 0.75      | 95.22 | 70.49 | 0.74        |
| <i>PIC_4_cycle_8</i>       | 1670     | 2198  | 0.76      | 90.16 | 69.52 | 0.77        |
| <i>PIC_4_cycle_16</i>      | 2984     | 3477  | 0.86      | 85.42 | 67.82 | 0.79        |
| <i>PIC_4_cycle_32</i>      | 5592     | 6111  | 0.92      | 77.91 | 67.34 | 0.86        |
| <i>PIC_4_cycle_64</i>      | 10829    | 11260 | 0.96      | 62.70 | 63.71 | 1.02        |
| <i>PIC_4_cycle_128</i>     | 21308    | 21550 | 0.99      | 55.66 | 55.63 | 1.00        |

Примечание:  $F_T$  и  $F_A$  — максимальная частота синхронизации процессора в мегагерцах при традиционном подходе и при использовании методики ASMD-FSMD;  $F_T/F_A$  — отношение соответствующих параметров;  $L_T$ ,  $L_A$  и  $L_T/L_A$  имеют прежнее значение.

**Таблица 3.** Среднеарифметические значения отношений  $L_T/L_A$  и  $t_T/t_A$  для разных методов умножения

| Метод         | Значение       |                |
|---------------|----------------|----------------|
|               | $mid(L_T/L_A)$ | $mid(t_T/t_A)$ |
| <i>mult_a</i> | 1.37           | 1.18           |
| <i>mult_b</i> | 1.37           | 1.11           |
| <i>mult_c</i> | 0.84           | <b>1.96</b>    |
| <i>mult_d</i> | 1.23           | <b>2.45</b>    |
| <i>booth</i>  | 1.23           | 1.20           |
| <i>mbooth</i> | 0.92           | 1.16           |
| <i>mid</i>    | 1.16           | 1.51           |

Примечание: *mid* — среднее значение параметра.

Проведенные экспериментальные исследования позволили сделать следующие выводы:

1) при проектировании цифровых устройств, поведение которых хорошо представляется в виде алгоритма функционирования, методика ASMD-FSMD по стоимости реализации и быстродействию имеет преимущество по сравнению с традиционным подходом;

2) при проектировании сложных функциональных блоков (например, процессоров, цифровых фильтров и др.) традиционный подход и специальные методы проектирования могут иметь преимущество над методикой ASMD-FSMD;

**Таблица 4.** Среднеарифметические значения отношений  $L_T/L_A$  и  $F_A/F_T$  для проектов процессоров PIC

| Метод              | Значение       |                |
|--------------------|----------------|----------------|
|                    | $mid(L_T/L_A)$ | $mid(F_A/F_T)$ |
| <i>PIC_1_cycle</i> | 0.98           | 1.09           |
| <i>PIC_1_multi</i> | 0.84           | 1.17           |
| <i>PIC_2_cycle</i> | 0.87           | <b>1.26</b>    |
| <i>PIC_4_cycle</i> | 0.87           | 0.86           |
| <i>mid</i>         | 0.89           | 1.10           |

**Таблица 5.** Время разработки проектов

| Метод              | Величина |        |             |
|--------------------|----------|--------|-------------|
|                    | $DT_T$   | $DT_A$ | $DT_T/DT_A$ |
| <i>mult_a</i>      | 188      | 33     | 5.70        |
| <i>mult_b</i>      | 176      | 29     | 6.07        |
| <i>mult_c</i>      | 172      | 28     | 6.14        |
| <i>mult_d</i>      | 237      | 43     | 5.51        |
| <i>booth</i>       | 167      | 27     | 6.19        |
| <i>mbooth</i>      | 195      | 35     | 5.57        |
| <i>PIC_1_cycle</i> | 10560    | 1440   | 7.33        |
| <i>PIC_1_multi</i> | 6240     | 960    | 6.50        |
| <i>PIC_2_cycle</i> | 840      | 120    | 7.00        |
| <i>PIC_4_cycle</i> | 980      | 150    | 6.53        |
| <i>mid</i>         | —        | —      | 6.35        |

Примечание:  $DT_T$  — время разработки в случае использования традиционного подхода;  $DT_A$  — время разработки в случае использования методики ASMD-FSMD;  $DT_T/DT_A$  — отношение соответствующих параметров; *mid* — среднее значение.

3) во всех случаях методика ASMD-FSMD имеет преимущество над традиционным подходом по времени проектирования.

## ЗАКЛЮЧЕНИЕ

На примере проектирования синхронных умножителей и процессоров PIC рассмотрены две технологии проектирования цифровых устройств: традиционный подход и методика ASMD-FSMD.

Методика ASMD-FSMD может использоваться при проектировании цифровых устройств на любой элементной базе (не обязательно FPGA), например, на специализированных интегральных схемах ASIC. В качестве языка проектирования может использоваться любой язык описания ап-

паратуры (не обязательно Verilog), например, VHDL или SystemVerilog.

Методика ASMD-FSMD требует дальнейшего совершенствования путем:

а) использования особенностей языков проектирования, например, применение в одном блоке ASMD нескольких операторов блокирующего назначения к одному и тому же регистру [22];

б) разработки методов повышения быстродействия ASMD, например, путем уменьшения числа состояний в циклах алгоритма;

в) разработки методов оптимизации ASMD с целью повышения производительности и надежности, а также уменьшения стоимости реализации.

Перспективным направлением также видится использование методики ASMD-FSMD для высокоуровневого проектирования сложных проектов.

## ФИНАНСИРОВАНИЕ РАБОТЫ

Работа выполнена при частичной финансовой поддержке Белостокского технологического университета (Белосток, Польша, грант WZ/WI-IIT/4/2020).

## 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. on Application-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.Ch., Ooi Ch.Y., Ismail N.Bt., Mohammad N.B.* // Proc. Int. Symp. on Circuits and Systems (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, 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. P. 176.
21. *Соловьев В.В.* Проектирование функциональных блоков встраиваемых систем на FPGA. М.: Горячая линия—Телеком, 2020.
22. *Соловьев В.В.* Язык Verilog в проектировании встраиваемых систем на FPGA. М.: Горячая линия—Телеком, 2020.