МОДЕЛИРОВАНИЕ АСИНХРОННОЙ ЛОГИКИ НА ПЕРЕКЛЮЧАТЕЛЬНОМ УРОВНЕ.I

Андрюхин А.И.
Кафедра ПМиИ, ДонНТУ
alexandruckin@rumbler.ru

Abstract

Andruckin A.I. Simulation of Asynchronous Logic on switch level.I *Modeling of basic primitives of Asynchronous Design on switch level are considering. The results of simulations are presented.*

Введение

За последнее время активизировалось внимание производителей чипов к построению асинхронных процессоров и, следовательно, к асинхронной логике. Так появились несколько реальных образцов таких процессоров. В отчете IBM за декабрь 2004 года [1], где проанализированы состояние технологий и рынка микропроцессоров и намечены основные направления их развития, наряду с очевидными тенденциями, как переход в наночастотный диапазон и выпуск многоядерных процессоров, указан и ряд асинхронных «альтернативных» направлений. Из четырех работ по самой престижной номинации «За выдающиеся достижения в области цифровой обработки» по итогам 2001 года журналом Microprocessor Report отмечены две работы фирмы Sun Microsystems – «Asynchronous Design Technology» и Theseus Logic за «NULL Convention Logic», которые непосредственно связаны с созданием асинхронных процессоров. В этой области активно работает Манчестерский университет силами AMULET Group[2]. Состояние дел в данной области отражено в [3, 4].

Общеизвестные преимущества асинхронных схем – быстродействие, низкое энергетическое рассеяние, модульная конструкция, иммунитет к метастабильному поведению, независимость от рассинхронизации часов, а также низкая чувствительность к электромагнитным помехам и малое их порождение [4].

В статье автор рассматривает возможности моделирования и тестирования асинхронных схем на переключательном уровне построенных по event-based методам проектирования [5], на основе модифицированной им системы [6].

Постановка задачи

Примитивы рассматриваемого метода проектирования асинхронных устройств используют комплекс динамических КМОП-примитивов. Поэтому естественен переход на переключательный уровень для моделирования и тестирования устройств, использующих эти примитивы. Для этих целей используем описанный в [7-9] подход и модифицированную систему моделирования [6]. Это применение и его результаты являются основной целью и содержанием данной работы.

Методы проектирования асинхронных устройств

Формализмы, использованные в асинхронном проектировании интегральных схем, могут быть разделены на два класса: формализм, базирующийся на булевой алгебре и формализм, основанный на последовательностях событий. На практике большинство методологий разработки асинхронных схемах используют смешение обоих формализмов.

Разработка многих асинхронных схем основана на булевой алгебре или ее производной теории переключений. Такие схемы используют модель ограниченных задержек, и примитивные элементы - вентили, соответствующие основным логическим функциям, как и, или, не. Этот формализм удобен для реализации логических функций, анализа схем на присутствие паразитных импульсов, и синтеза схем [10].

Событийный формализм имеет дело с последовательностями событий, а не переменных бинарной логики. Схемы, разработанные с помощью этого формализма и использованием модели неограниченных задержек, обычно работают в режиме ввода – вывода и имеют такие примитивные элементы, как разветвления, переключатели, и слияния. Формализм на базе событий очень удобен для конструирования асинхронных схем, при высокой степени запутанности параллелизма. Были созданы несколько инструментальных средств для автоматической проверки асинхронных схем с событийным формализмом и примерами таких формализмов являются, к примеру, сети Петри. На рис. 1 показано несколько простых примитивов и их обозначений, используемых в event-based методах проектирования.

Самый простой примитив *WIRE* (ПОТОК ДАННЫХ) – это двухполюсный элемент, который формирует выходное событие на выходе *b* после каждого входного события на входе *a*. События входа и выхода в потоке данных, должны чередоваться. Входное событие должно сопровождаться выходным событием *b* прежде, чем произойдет другое событие. Поток данных физически осуществляется с помощью провода (шины), а события – изменениями напряжения. Инициализированный

поток данных, или *IWIRE*, очень похож на *WIRE*, за исключением того, что *IWIRE* формирует событие выхода *b* вместо того, чтобы принять входное событие *a*; после этого, его режим работы совпадает с *WIRE*.

Примитив для синхронизации – *JOIN* (объединение), также называемый *RENDEZVOUS* (рандеву) . *JOIN* имеет два входа *a* и *b* и один выход *c*. *JOIN* выполняет операцию Логическое И двух событий – *a* и *b*. Он формирует событие выхода *c* только после того, как получат события оба входа, *a* и *b*. Входные данные могут измениться снова, только после того, как сформированы выходные данные. *JOIN* может быть реализован *C*-элементом Мюллера, который рассматривается далее.

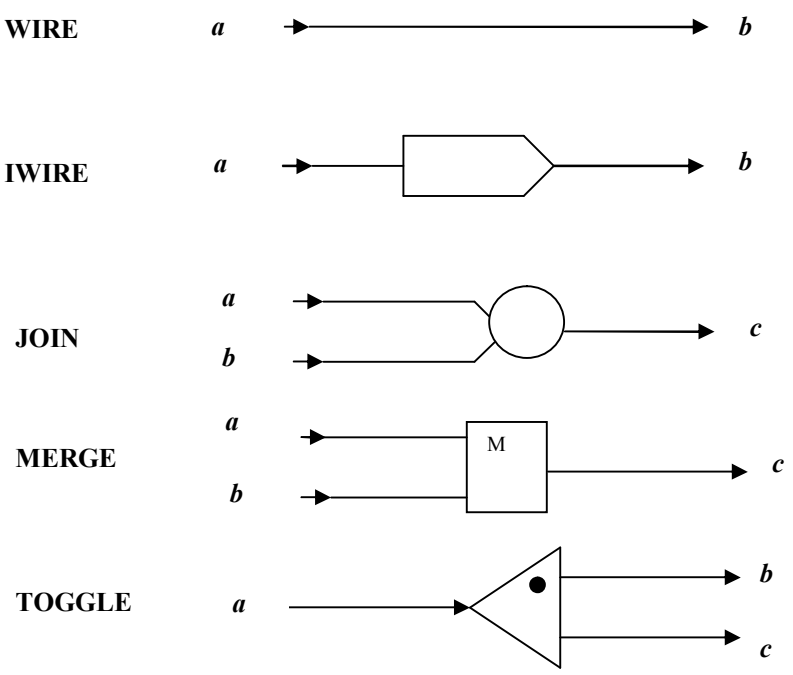

Рис. 1. Основные нечувствительные к задержкам примитивы

Элемент *MERGE* (объединение) выполняет операцию Логическое **ИЛИ** двух событий. Если элемент MERGE получает событие на любом из его входов, ***a*** или ***b***, он формирует событие выхода ***c.*** После входного события следует выходное; последовательные входные события не допускаются. *MERGE* может быть реализован логическим элементом **XOR**.

TOGGLE (переключатель) имеет один вход и два выхода ***b*** и ***c.*** После появления события на входе ***a***, оно формируется на выходе *b*. Следующее событие на входе *a* формируется на выходе *c*. Входное событие должно сопровождаться событием выхода прежде, чем произойдет другое входное событие. Таким образом, события выхода чередуются или переключаются после каждого входного события. Точка на рисунке схемы показывает, какой выход формирует событие первым.

C-элемент Мюллера назван по имени его изобретателя Д.Е. Мюллера [11]. Традиционно, его логическое поведение описывается следующим образом. Если оба входа – 0 (1), то выход становится 0 (1); иначе говоря сохраняет то же значение. Для правильности операций ***C***-элемента, также предполагается, что, как только оба входа становятся 0 (1), они не изменяются снова, пока не поменяется выход. Поведение выхода ***c*** ***C***-элемента выражается с использованием значений входов ***a*** и ***b*** и предыдущего состояния выхода ***c'*** следующей логической функцией **c=c'(a+b)+a*b**

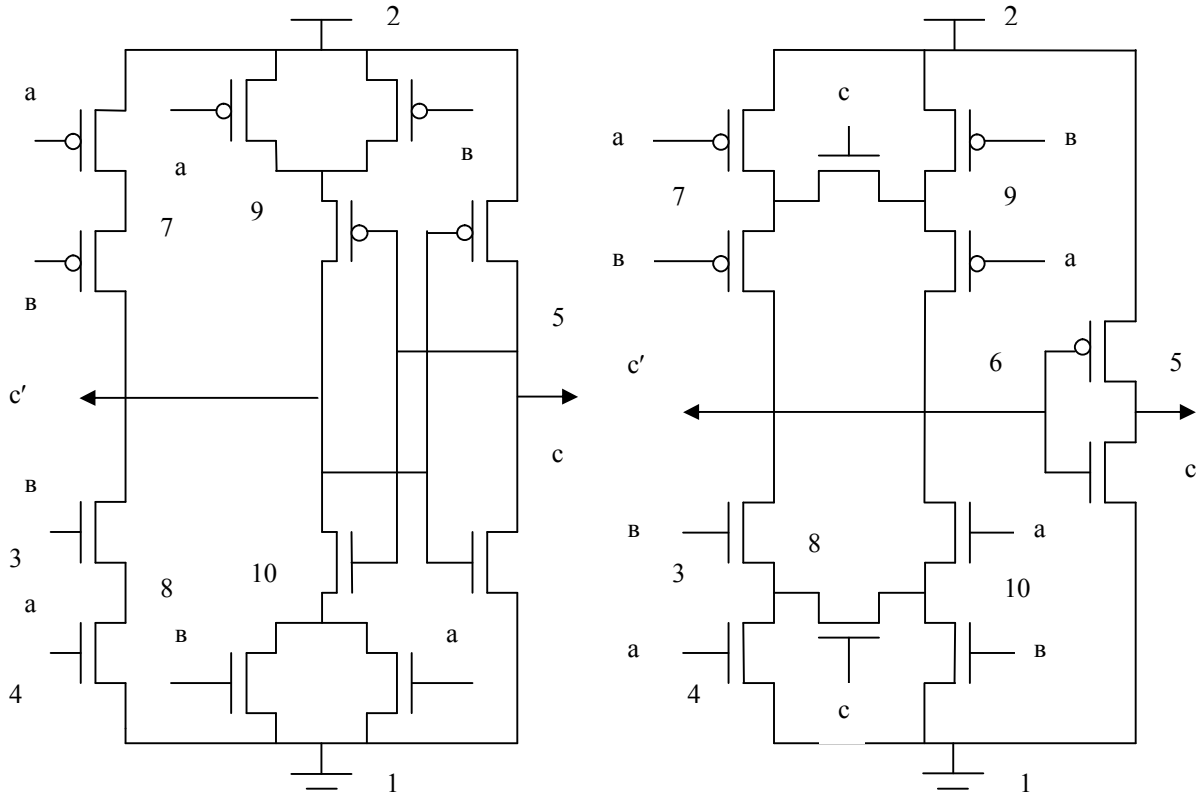

Рис. 2. КМОП-реализации С-элемента: (a) – стандартная; (b) – симметрическая.

C-элемент может использоваться для реализации *JOIN*, который имеет несколько более ограниченное поведение окружения, в том смысле, что входные данные не разрешено изменять дважды в последовательности. Мы приводим две самые распространенные КМОП реализации из их множества на рис. 2. Реализация (a) – стандартная, предложенная Сазерлендом [5]. Реализация (b) предложена Ван Беркелем [12]. Каждая реализация имеет свои собственные характеристики. Реализация (b) – считается лучшим выбором по критерию быстродействия и эффективности.

Данные таблицы в скобках относятся к симметрической реализации С-элемента. Значения в узлах схем для входов (3, 4 узлы) и выхода (узел 5)

удовлетворяют приведенному выше уравнению функционирования С-элемента.

Таб. 1. Результаты моделирования С-элемента

Номер набора	Число итераций	Значения сигналов в узлах схемы									
		1	2	3	4	5	6	7	8	9	10
1	3	D0	D1	D0	D0	D0	D1	D1	CX	D1	CX
2	4(3)	D0	D1	D1	D1	D1	D0	C1	D0	C1	D0
3	5(7)	D0	D1	D0	D1	D1	D0	D0	D0	D1	D0
4	3	D0	D1	D1	D1	D1	D0	C0	D0	C1	D0
5	5(7)	D0	D1	D1	D0	D1	D0	D1	D0	D1(0)	D0
6	4(3)	D0	D1	D0	D0	D0	D1	D1	C0	D1	C0

Рассмотрим два контролируемых событиями фиксатора Сазерленда [5], которые изображены на рис. 3. Их работа управляется с помощью двух входных управляющих сигналов: фиксирование и проход, помеченных соответственно ***c*** и ***p***. Они, также, имеют два управляющих выходных сигнала: осуществлено фиксирование – ***cd***, и осуществлен проход – ***pd***. Входные данные – ***D***, выходные данные – ***Q***. Реализация (a) состоит из трех так называемых переключателей на два направления. Реализация (b) включает *MERGE*, *TOGGLE* и контролируемый уровень фиксатор, состоящий из переключателя на два направления и инвертора. Переключатель на два направления схематично представлен инвертором и переключателем. Переключатель переключает между двумя позициями, в зависимости от логического значения управляющего сигнала.

Переключатель на два направления, фактически, является мультиплексором с двумя входами, который формирует инвертированный вариант его входного сигнала. КМОП реализация переключателя на два направления показана на рис. 4 [5]. Позиция коммутатора соответствует состоянию, при котором значение входа ***c*** низкое.

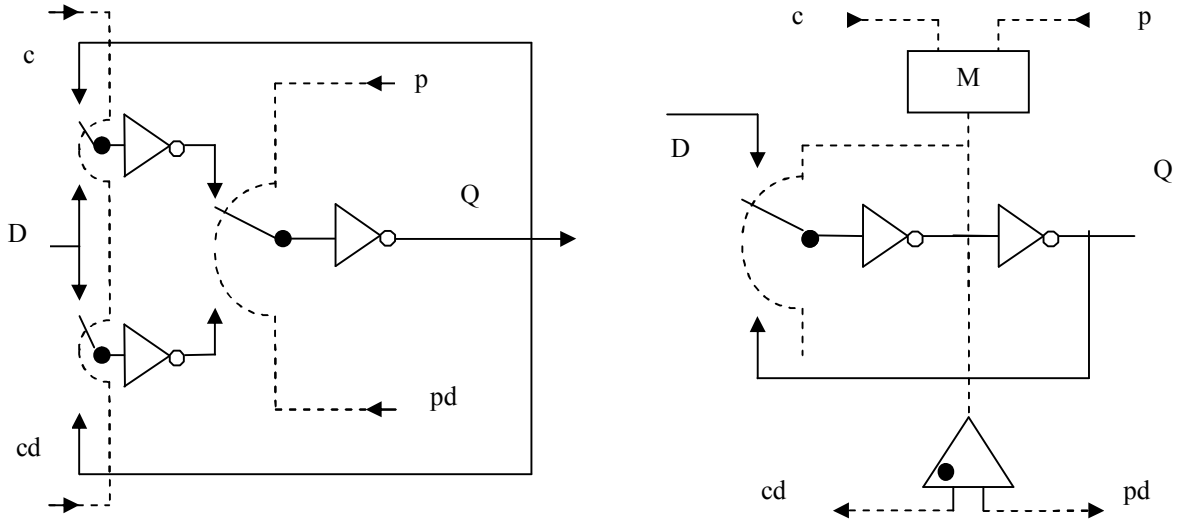

Рис.3. Две реализации управляемого событиями фиксатора

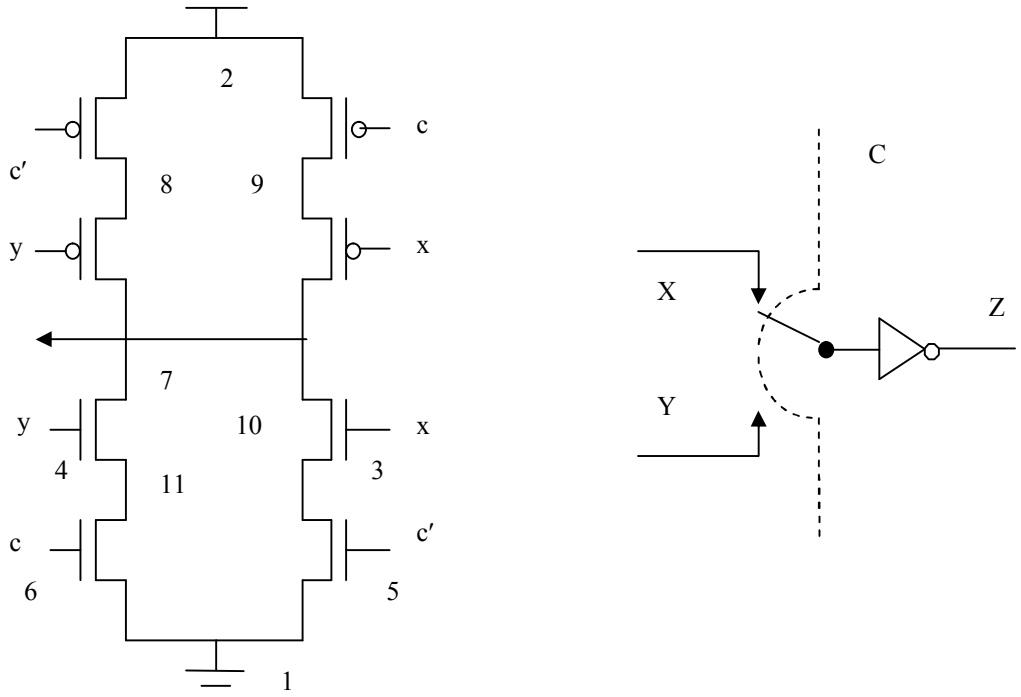

Рис. 4. КМОП реализация переключателя на два направления

Контролируемый событиями фиксатор имеет два состояния: прозрачное и непрозрачное. В прозрачном состоянии никакие данные не фиксируются, но выходные данные копирует входные, потому что путь двух стадий инвертирования лежит между входом и выходом.

Таб. 2. Результаты моделирования фиксатора

Номер набора	Число итераций	Значения сигналов в узлах схемы										
		1	2	3	4	5	6	7	8	9	10	11
1	2	D0	D1	D0	D1	D1	D0	CX	CX	D1	D0	CX
2	3	D0	D1	D0	D0	D0	D0	D1	D1	D1	C0	CX
3	3	D0	D1	D1	D1	D1	D1	D0	C1	C1	D0	D0
4	2	D0	D1	D0	D0	D1	D1	CX	CX	CX	D0	D0
5	2	D0	D1	D1	D0	D1	D1	CX	CX	CX	D0	D0
6	2	D0	D1	D0	D0	D1	D0	CX	CX	D1	D0	C0
7	2	D0	D1	D0	D0	D1	D0	CX	CX	D1	D0	C0
8	4	D0	D1	D0	D0	D0	D1	D1	D1	D1	C0	D0
9	2	D0	D1	D1	D1	D0	D0	CX	D1	D1	CX	CX
10	2	D0	D1	D0	D1	D1	D0	CX	C1	D1	D0	CX

В непрозрачном состоянии, этот путь построен так, что входные данные можно изменить без воздействия на выходные данные. Ясно, что при этом текущие данные запираются при выводе. Реализации на рис. 2 (a), 2(b) показаны в начальных прозрачных состояниях. Фиксирование и

проход сигналов в контролируемом событиями фиксаторе всегда чередуются. При переходе на ***c***, фиксатор фиксирует текущие входные данные и становится непрозрачным. Следующее перемещение на ***cd*** – подтверждение провайдеру данных, о том, что текущие данные были зафиксированы и что входные данные могут быть безопасно изменены. Последующее перемещение на ***p*** возвращает фиксатор назад к его прозрачному состоянию, чтобы следующие данные прошли к выходу. Сигнал ***p*** подтверждается перемещением на ***pd***. Обратите внимание, что в реализации (а) рис. 2, сигналы ***cd*** и ***pd*** просто запаздывают и возможно являются усиленными варианты ***c*** и ***p***, соответственно.

Сравнение различных микроконвейерных фиксаторов представлено в [13]].

Использование вышеописанных элементов наиболее привлекательно в конвейерном режиме обработки данных, который является мощной технологией для создания высокоэффективных микропроцессоров [14-16]. Самая простая разновидность микроконвейера – FIFO. Четырехуровневый FIFO-микроконвейер изображен на рис. 5. Его схема управления, состоит исключительно из взаимосвязанных элементов *JOIN*, и канала данных из контролируемых событиями регистров. Управляющие сигналы обозначены пунктирными линиями. Толстые стрелки показывают направление потока данных. Данные кодируются одноканальным кодированием, а пропускная способность информационного канала зависит от размера регистров. Смежные уровни FIFO-микроконвейера связаны двухтактным, протоколом передачи пакетных данных. Это означает переход запроса на следующий уровень, только когда данные для этого уровня становятся действительными. Кружок на входе JOIN – ускоряет работу *JOIN* с *IWIRE*. Имеется в виду, что, первоначально, событие уже произошло на входе с кружочком, и *JOIN* может сформировать событие выхода сразу после получения события на другом входе.

Изначально, все управляющие цепи FIFO имеют значение лог. 0 и данные в регистрах не действительны. FIFO активизируется повышением напряжения на ***Rin***, которое указывает, что входные данные действительны. Далее, *JOIN* первого уровня повышает напряжение выхода. Этот сигнал – запрос регистру первого уровня, для фиксирования данных и установки непрозрачного состояния. После фиксирования данных, регистр повышает напряжение на его выходе ***cd***. Это вызывает перемещение на ***Ain*** и на ***r1***, который является запросом ко второму уровню FIFO. Тем временем, данные переходят к регистру второго уровня и достигают его прежде, чем происходит перемещение на *r1*. Если оборудование не посылает никаких новых данных, первый уровень остается в режиме ожидания, а данные и сигналы запроса распространяются далее вправо. Каждый раз, когда данные зафиксированы

уровнем, предыдущему уровню посылается подтверждение, которое переключает его фиксатор обратно в прозрачное состояние. Когда данные доходят до последнего регистра, они сохраняются, и сигнала запроса *Rout* направляется «пользователю» FIFO.

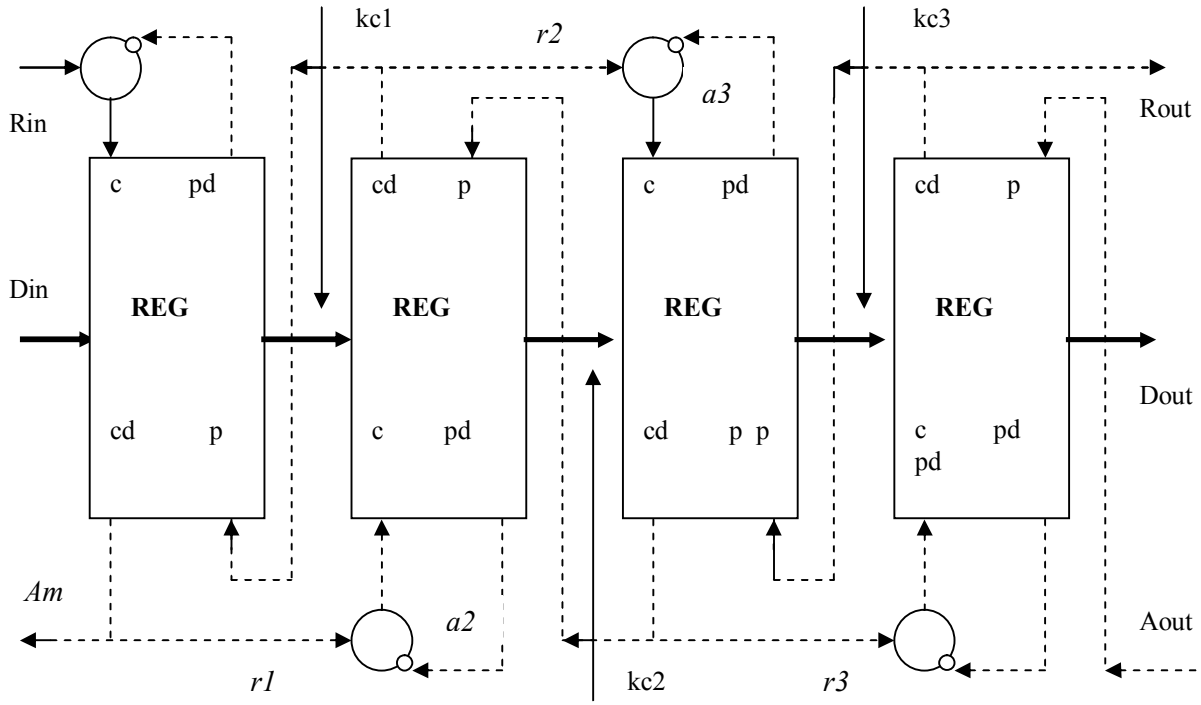

Рис. 5. Структура четырехуровневого FIFO-микроконвейера

В этой точке, все управляющие сигналы имеют значение лог. 1, исключая *Aout*. Если данные не ушли из FIFO, то есть, *Aout* остается низким, следующие данные, исходящие от источника дойдут только до третьего уровня, потому что JOIN четвертого уровня не может произвести вывод. Наконец, *Aout* тоже повышается, когда «пользователь» подтверждает получение данных. Дальнейшее сохранение и удаление данных следуют аналогично. Операция каждого JOIN может интерпретироваться следующим образом. Если предыдущий уровень послал запрос для фиксирования данных *и* текущий уровень пуст, тогда посылается сигнал фиксирования данных в текущем уровне.

FIFO может быть легко изменен для добавления обработки данных. Так канал обработки информации может состоять из поочередно управляемых событиями регистров и добавляемых комбинационных схем кс1, кс2, кс3. Управляемые событиями регистры сохраняют входные и выходные данные комбинационных схем, а комбинационные схемы выполняют необходимую обработку данных. Для FIFO-микроконвейер нет ограничений на интенсивность ввода или вывода данных, если бы не задержки, наложенные элементами схемы. В обычном синхронном конвейере интенсивность ввода и вывода данных из конвейера, диктуется внешними сигналами синхронизации. FIFO-микроконвейер более

экономичен в плане количества рассеиваемой энергии, которое пропорционально количеству переходов данных, по сравнению с синхронизируемым конвейером, который непрерывно рассеивает энергию, Другая заманчивая особенность FIFO-микроконвейера - это автоматической выключение, в период бездеятельности. Синхронизированный же конвейер, требует специального механизма управления синхронизирующими сигналами, для осуществления этой функции. Однако, этот механизм контроля, должен постоянно потреблять энергию, потому что он никогда не может войти в режим ожидания.

Заключение и перспективы дальнейших исследований

Моделирование примитивов асинхронной логики является первым шагом для их тестирования и тестирования устройств, в состав которых они входят. В настоящее время уже реализована возможность моделировать и строить проверяющие тесты для тех неисправностей, для которых это выполнять на вентильном уровне весьма затруднительно [17]. Вопросы тестирования на переключательном уровне основных классов неисправностей современных СБИС будут представлены в следующих работах автора.

Литература

1.The year in microprocessors. IBM, 2004, www.ibm.com/ developerworks /library/pa-yearend.html.

2.Л. Черняк. «Открытые системы», № 5, 2002.

3.S. Hauck, “Asynchronous design methodologies: An overview”, Proc. IEEE, Vol. 83, No. 1,Jan. 1995, pp. 69-93.

4.The Status of Asynchronous Design in Industry. www.scism.sbu.ac.uk/ccsv/ACiD-WG/AsyncIndustryStatus.pdf

5.I. E. Sutherland, “Micropipelines”, Communications of the ACM, Vol. 32, no. 6, pp. 720-738, June 1989.

6.Андрюхин А.И., Сперанский Д.В. Иерархическая компилятивная система моделирования и генерации тестов// Техническая диагностика и неразрушающий контроль. -1994. N 2. -с. 71-78.

7.Андрюхин А.И. Алгоритмы параллельного логического моделирования и псевдослучайной генерации тестов для МОП-структур // Микроэлектроника. -1995, N 5. -с. 331-336.

8.Андрюхин А.И. Параллельное логическое моделирование МОП-структур на переключательном уровне. // Электронное

моделирование. -1996, N 2, -с. 88-92.

9.Андрюхин А.И. Параллельное многозначное логическое моделирование исправных и неисправных псевдобулевых схем. // Электронное моделирование -1997, N 1. -с. 58-63.

10.J.A. Brzozowski and C.-J. Seger. Asynchronous Circuits. Springer-Verlag.1995.

11.R.E. Miller, Switching Theory Volume II: Sequential Circuits and Machines. New York NY: John Wiley & Sons, 1965.

12.K.V. Berkel. Beware the isochronic fork. Integration, the VLSI journal, vol.13, June 1992, pp.103-128.

13.P. Day and J.V. Woods. Investigation into micropipeline latch design styles.//IEEE Transactions on VLSI Systems, vol.3, June 1995, pp.264-272.

14.Myers, C. Asynchronous Circuit Design. John Wiley & Sons, 2001.

15.O.A. Petlin, S.B. Furber, “Scan testing of micropipelines”, Proc. 13th IEEE VLSI Test Symposium, Princeton, New Jersey, USA, May 1995, pp. 296-301.

16.L.A. Hollaar. Direct Implementation of Asynchronous Control Units. IEEE Transactions on Computers, Vol. C-31,No. 12, December 1982.

17. Андрюхин А.И. Параллельное моделирование неисправностей МОП-структур// Научные труды Донецкого государственного технического университета. Серия: Проблемы моделирования и автоматизации проектирования динамических систем. Выпуск 29. 2001 г., С.205-211.

Дата надходження до редакції 15.10.2006 р.