1. Причини та передумови виникнення мов HDL.

Причины:
Развитие цифровых схем
Автоматизация процесса разработки чипов
Предпосылки:
Внедрение технологий Very large Scale Integration
Возрастание популярности компьютерных программ для автоматической трассировки и размещения элементов схемы
Логическое моделирование
Необходимость в создании стандартного языка для описания цифровых схем
Так появились языки описания цифровых схем – Hardware Design Languages.
Языки HDL позволяли описывать одновременные процессы происходящие в цифровых элементах.
Языки описания аппаратуры Verilog(R) и VHDL стали популярны.
Несмотря на популярность новых языков для проектирования и проверки логики цифровых схем, разработчикам все еще приходилось в дальнейшем вручную переводить проекты с языков HDL в схемы цепей и соединения между элементами.

Изобретение логического синтеза в конце 80х годов позволило радикально изменить методику проектирования.

Цифровые схемы могут быть описаны на уровне перемещения данных из регистра в регистр (Register Transfer Level - RTL) с использованием языков HDL.

Разработчики описывают, как данные перетекают из регистров в регистры и как они обрабатываются в схеме.

Детальное представление проекта\схемы на уровне элементов (gates) и соединения между ними автоматически выделяется позднее из RTL описания средствами логического синтеза.

Сегодня Verilog HDL - это принятый IEEE стандарт. В 1995 был утвержден оригинальный стандарт IEEE 1364-1995. Последний стандарт IEEE 1364-2001 значительно улучшен по сравнению с первоначальным.

2. Аспекти та рівні опису апаратури мовами HDL.

Языки описания аппаратуры (HDL-языки) имеют две основные разновидности – языки низкого уровня (аналоги языков программирования типа ассемблера) и высокого уровня .
Языки низкого уровня ближе к аппаратным средствам, вследствие чего представляют для компиляторов потенциальные возможности создания проектов с более выигрышными параметрами.
Платой за это является обычно жесткая ориентация на определенную аппаратуру и призводящую ее фирму.
Примерами таких языков могут служить языки PLDASM (фирма Intel), AHDL (Фирма Altera) и ABEL (Фирма Zilinx).
Языки высокого уровня менее связаны с аппаратными платформами и поэтому более универсальны.
Среди них наиболее распространены языки VHDL и Verilog.
Эти языки, как и другие алгоритмические языки высокого уровня, в принципе позволяют описать любой алгоритм в последовательной форме, т.е. через последовательность операторов присвоения и принятия решений.
Основное их отличие в способности отражать также и параллельно исполняемые в аппаратуре действия, представляемые отдельными параллельно выполняемыми процессами с общим инициализирующим воздействием.
Совсем недавно был создан язык HDL-уровня - System Verilog.
Это новый язык описания аппаратуры, созданный на базе Verilog и C++.
Он призван расширять возможности классического языка описания Verilog и дает возможность проводить моделирование как на функциональном, так и на системном уровне.

SystemC, как и System Verilog, разрабатывался как единый язык проектирования, способный обеспечить выполнение архитектурного моделирования и возможность синтеза разработанной системы.

3. Стисла характеристика найпоширеніших мов HDL.

Языки VHDL и Verilog (Verilog HDL) относятся к языкам описания аппаратуры. Эти языки предназначены для моделирования электронных схем на уровнях вентильном, регистровых передач, корпусов микросхем. Поэтому эти языки можно назвать языками сквозного функционально-логического проектирования.

Основные составляющие языков VHDL и Verilog

Типы данных

В более простом языке Verilog поддерживаются только самые простые типы данных - целые (32-бит со знаком), действительные (с плавающей запятой), а также специфические типы "время" и "событие".

В VHDL шире набор базовых типов, и, кроме этого, проектировщик может создавать свои типы данных, а в Verilog этого делать нельзя.

Надо отметить, что программируются-то в этих языках как данные не элементы памяти, а сигналы. В Verilog, например, они бывают только цепными и регистровыми (последние могут запоминаться где-то).

Другие элементы VHDL и Verilog

В VHDL синтаксис позволяет описывать модель в разных стилях (структурное, потоковое, поведенческое описания), а также встраивать в описание фрагменты языков программирования высокого уровня (Си, Паскаль). Этим и достигается его большая универсальность и применяемость не только для описания архитектур вычислительных систем. Например, моделирование разных физических систем у него имеет поддержку в виде типов с физическими размерностями.

Verilog также поддерживает разные описания модели системы, но интерфейса с обычными языками программирования у него нет.

Заключение

Verilog - достаточно простой язык, сходный с языком программирования Си - как по синтаксису, так и по "идеологии". Малое количество служебных слов и простота основных конструкций упрощают изучение и позволяют использовать Verilog в целях обучения. Но в то же время это эффективный и специализированный язык.

VHDL обладает большей универсальностью и может быть использован не только для описания моделей цифровых электронных схем, но и для других моделей. Однако из-за своих расширенных возможностей VHDL проигрывает в эффективности и простоте, то есть на описание одной и той же конструкции в Verilog потребуется в 3√4 раза меньше символов (ASCII), чем в VHDL.

Оба языка поддерживаются в качестве стандартов большим количеством программных продуктов, в том числе и open source, в области САПР. Имеются и компиляторы, и симуляторы для обоих языков, в том числе, например, и с первого языка на второй. Именно эти языки используются при проектировании (с помощью современных средств САПР ведущими производителями FPGA) не только самих СБИС, но и готовых модулей (ядер), мегафункций (megafunctions), предназначенных для решения достаточно сложных задач обработки сигналов.

4. Метод імітаційного моделювання.

Определим метод имитационного моделирования в общем виде как экспериментальный метод исследования реальной системы по ее имитационной модели, который сочетает особенности экспериментального подхода и специфические условия использования вычислительной техники. В имитационном моделировании важную роль играет не только проведение, но и планирование эксперимента на модели. Однако это определение не проясняет, что собой представляет сама имитационная модель. В процессе имитационного моделирования (рис. 2.1) исследователь имеет дело с четырьмя основными элементами:

- реальная система;
- логико-математическая модель моделируемого объекта;
- имитационная (машинная) модель;
- ЭВМ, на которой осуществляется имитация – направленный вычислительный эксперимент.

Исследователь изучает реальную систему, разрабатывает логико-математическую модель реальной системы. Имитационный характер исследования предполагает наличие логико - или логико-математических моделей, описываемых изучаемый процесс.

Особенностью имитационного моделирования является то, что имитационная модель позволяет воспроизводить моделируемые объекты:

- с сохранением их логической структуры;
- с сохранением поведенческих свойств (последовательности чередования во времени событий, происходящих в системе), т.е. динамики взаимодействий.

При имитационном моделировании структура моделируемой системы адекватно отображается в модели, а процессы ее функционирования проигрываются (имитируются) на построенной модели. Поэтому построение имитационной модели заключается в описании структуры и процессов функционирования моделируемого объекта или системы. В описании имитационной модели выделяют две составляющие:

- статическое описание системы, которое по-существу является описанием ее структуры. При разработке имитационной модели необходимо применять структурный анализ моделируемых процессов.
- динамическое описание системы, или описание динамики взаимодействий ее элементов. При его составлении фактически требуется построение функциональной модели моделируемых динамических процессов.

Отличительной особенностью метода имитационного моделирования является возможность описания и воспроизведения взаимодействия между различными элементами системы. Таким образом, чтобы составить имитационную модель, надо:

- представить реальную систему (процесс), как совокупность взаимодействующих элементов;
- алгоритмически описать функционирование отдельных элементов;
- описать процесс взаимодействия различных элементов между собой и с внешней средой.

5. Модельний час.

Для описания динамики моделируемых процессов в имитационном моделировании реализован *механизм задания модельного времени.*

Этот механизм встроен в управляющие программы системы моделирования.

Если бы на ЭВМ имитировалось поведение одной компоненты системы, то выполнение действий в имитационной модели можно было бы осуществить последовательно, по пересчету временной координаты.

Чтобы обеспечить имитацию параллельных событий реальной системы вводят некоторую

глобальную переменную (обеспечивающую синхронизацию всех событий в системе) t_0, которую называют модельным (или системным) временем.

Существуют два основных способа изменения t_0:

- пошаговый (применяются фиксированные интервалы изменения модельного времени);
- по-событийный (применяются переменные интервалы изменения модельного времени, при этом величина шага измеряется интервалом до следующего события).

В случае пошагового метода продвижение времени происходит с минимально возможной постоянной длиной шага. Эти алгоритмы не очень эффективны с точки зрения использования машинного времени на их реализацию.

Способ фиксированного шага применяется в случаях:

- если закон изменения от времени описывается интегро-дифференциальными уравнениями. Характерный пример: решение интегро-дифференциальных уравнений численным методом. В подобных методах шаг моделирования равен шагу интегрирования. Динамика модели является дискретным приближением реальных непрерывных процессов;
- когда события распределены равномерно и можно подобрать шаг изменения временной координаты;
- когда сложно предсказать появление определенных событий;
- когда событий очень много и они появляются группами.

В остальных случаях применяется по-событийный метод, например, когда события распределены неравномерно на временной оси и появляются через значительные временные интервалы.

По-событийный метод (принцип “особых состояний”). В нем координаты времени меняются тогда, когда изменяется состояние системы. В по-событийных методах длина шага временного сдвига максимально возможная. Модельное время с текущего момента изменяется до ближайшего момента наступления следующего события. Применение по-событийного метода предпочтительнее в том случае, если частота наступления событий невелика. Тогда большая длина шага позволит ускорить ход модельного времени. На практике по-событийный метод получил наибольшее распространение.

Таким образом, вследствие последовательного характера обработки информации в ЭВМ, параллельные процессы, происходящие в модели, преобразуются с помощью рассмотренного механизма в последовательные. Такой способ представления носит название квазипараллельного процесса.

6. Імітація паралельних процесів.

Практически любая более или менее сложная система имеет в своем составе компоненты, работающие одновременно, т.е. параллельно. Параллельно работающие подсистемы могут взаимодействовать самым различным образом, либо вообще работать независимо друг от друга. Способ взаимодействия подсистем определяет вид параллельных процессов, протекающих в системе. Также, вид моделируемых процессов влияет на выбор метода их имитации.
Рассмотрим виды параллельных процессов.

Асинхронный параллельный процесс — это такой процесс, состояние которого не зависит от состояния другого параллельного процесса (ПП).

Пример асинхронных ПП, протекающих в рамках одной системы, — это подготовка и проведение рекламной кампании фирмой и работа сборочного конвейера. Или например, из области вычислительной техники — выполнение вычислений процессором и вывод информации на печать.

Синхронный ПП — это такой процесс, состояние которого зависит от состояния взаимодействующих с ним ПП. Пример синхронного ПП — работа торговой организации и доставка товара со склада (нет товара — нет торговли).

Один и тот же процесс может быть синхронным по отношению к одному из активных ПП и асинхронным по отношению к другому.

Подчиненный ПП создается и управляется другим процессом (более высокого уровня). Примером таких процессов является ведение боевых действий подчиненными подразделениями.

Независимый ПП— процесс, который не является подчиненным ни для одного из процессов. Например, после запуска неуправляемой зенитной ракеты ее полет можно рассматривать как независимый процесс, одновременно с которым самолет ведет боевые действия другими средствами.

Способ организации параллельных процессов в системе зависит от физической сущности этой системы.

Разработка и использование любой ИМ предполагает ее программную реализацию и исследование с применением вычислительных систем (ВС). Реализация параллельных процессов в ВС имеет следующие особенности:

- на уровне задач вычислительные процессы могут быть истинно параллельными только в многопроцессорных ВС или вычислительных сетях;

- многие ПП используют одни и те же ресурсы, поэтому даже асинхронные ПП в пределах одной ВС вынуждены согласовывать свои действия при обращении к общим ресурсам;

- в ВС используется еще два вида ПП: родительский и дочерний ПП; особенность их состоит в том, что процесс-родитель не может быть завершен, пока не завершатся все его дочерние процессы.

Для организации взаимодействия параллельных процессов в ВС используются три основных подхода:

- на основе «взаимного исключения» - предполагает запрет доступа к общим ресурсам (общим данным) для всех ПП, кроме одного, на время его работы с этими ресурсами (данными);

- на основе синхронизации посредством сигналов - подразумевает обмен сигналами между двумя или более процессами по установленному протоколу. Такой «сигнал» рассматривается как некоторое событие, вызывающее у получившего его процесса соответствующие действия;

- на основе обмена информацией (сообщениями) – когда необходимо передавать от одного ПП другому более подробную информацию, чем просто «сигнал-событие».

Эти механизмы реализуются в ВС на двух уровнях — системном и прикладном.

Механизм взаимодействия между ПП на *системном* уровне определяется на этапе разработки ВС и реализуется в средствами операционной системы (частично — с использованием аппаратных средств).

На *прикладном* уровне взаимодействие между ПП реализуется программистом средствами языка, на котором разрабатывается программное обеспечение.

7. Верифікація опису засобами HDL.

В процессе развития методов, применяемых при проектировании цифровых систем, ситуация развивалась следующим образом. Сначала разработчики использовали только схемотехнику и проектировали цифровые блоки из элементарных вентилей, накапливая библиотеки отработанных блоков. Затем, с ростом количества элементарных вентилей, задействованных в проекте, постепенно произошёл качественный переход от схемотехники к языкам описания аппаратуры – HDL (Hardware Description Language). В настоящее время необходимость применения HDL языков не вызывает сомнений. К маршруту проектирования, состоящему из основных этапов – создание HDL, моделирование и верификация, синтез, размещение и трассировка, – по мере необходимости добавляются новые этапы. Однако с дальнейшим ростом объёма и функциональной сложности проектов время, требуемое на верификацию, увеличивается, доходя в отдельных случаях до 90% и более от всего времени разработки проекта, и всё чаще возникают проекты, в которых «стандартный» маршрут просто не работает. Например, функциональная сложность проекта может возрасти настолько, что команда инженеров по верификации не сможет предусмотреть все потенциальные проблемы и их верифицировать. Как следствие – законченный проект будет содержать функциональные ошибки.

Верификация может проводиться как *статически,* то есть *формальными методами,* когда корректность HDL-описания доказывается математически, так и *динамическими,* когда поведение, задаваемое HDL-описанием, сравнивается с некоторым эталоном в процессе его выполнения в среде симулятора. Хотя формальные методы в последнее время получили большое развитие, их использование всё ещё является очень ограниченным.

Существующие методы верификации HDL-описаний

Формальная верификация (аналитическая верификация)

Динамическая верификация (тестирование)

Целевой объект:

модуль (unit) / система в целом (core)

Фаза генерации теста:

в ходе статического анализа / в процессе исполнения

Тестовые воздействия:

случайные / набор / по ограничениям

Оракул:

наборы эталонных результатов / вычисление эталонных результатов по модели / ограничение на ожидаемый результат (post-условие) / контроль assertions

Метрики полноты:

по структуре HDL-описания / эталонной модели

8. Оцінки повноти тестів.

Для оценки полноты (степени завершенности) тестирования используют средства описания тестового покрытия. Структура тестового покрытия описывается явным перечислением возможных при тестировании ситуаций (тестовых ситуаций). Для описания сложных ситуаций имеются средства композиции тестовых покрытий: сложная ситуация представляется в форме совокупности нескольких более простых ситуаций.

После выполнения теста можно построить отчет о проведенном тестировании, который, помимо обнаруженных ошибок, содержит информацию об уровне достигнутого тестового покрытия — какие тестовые ситуации были покрыты во время тестирования, а какие нет. Отчеты о тестировании позволяют оценить качество тестов и сфокусировать усилия на проверке еще не охваченных аспектов функционирования целевой системы.

9. Будова та еволюція ПЛІС.

Программи́руемая логи́ческая интегра́льная схе́ма (**ПЛИС**, англ. *programmable logic device*, PLD) — электронный компонент, используемый для создания цифровых интегральных схем. В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задаётся посредством программирования (проектирования). Для программирования используются программаторы и отладочные среды, позволяющие задать желаемую структуру цифрового устройства в виде принципиальной электрической схемы или программы на специальных языках описания аппаратуры: Verilog, VHDL, AHDL и др.

Ранние ПЛИС

В 1970 году компания Texas Instruments разработала маскируемые (программируемые с помощью маски, англ. mask-programmable) ИС основанные на ассоциативном ПЗУ (ROAM) фирмы IBM. Эта микросхема, TMS2000, программировалась чередованием металлических слоёв в процессе производства ИС. TMS2000 имела до 17 входов и 18 выходов с 8-ю JK-триггерами в качестве памяти. Для этих устройств компания TI ввела термин Programmable Logic Array(PLA) — программируемая логическая матрица.

14.3.3. Составные части ПЛИС

Основой ПЛИС (рис. 14.11) является коммутационная матрица, логические блоки (ЛБ), входной (ВХК) и выходной (ВЫХК) каскады.

Коммутационная матрица при программировании устанавливает внутренние связи между частями ПЛИС, определяя взаимодействие логических блоков между собой и внешним миром.

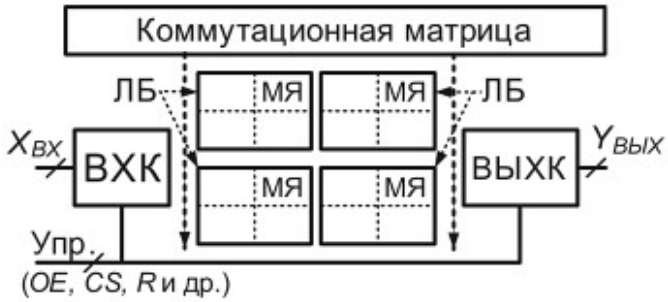

Рис. 14.11

Логические блоки в свою очередь состоят из наборов макроячеек (МЯ). Каждая из макроячеек является миниатюрным программируемым устройством, выполненным на основе локальной ПЛМ (дешифратора и коммутатора) для реализации комбинационных функций и триггерно-регистровых устройств, зависящих от типа микросхемы. Поэтому на основе макроячеек можно создавать не только комбинационные устройства, но и устройства с памятью или микропрограммные автоматы (см. п. 14.2.3), вводя при помощи управляемого коммутатора ПЛМ необходимые обратные связи. Более сложные ПЛИС могут иметь в своем составе и другие цифровые узлы, такие как компараторы, массивы ОЗУ и ПЗУ.

Входной и выходной каскады формируют необходимые характеристики, определяют нагрузочную способность, тип выхода и другие заданные внешние параметры. Отметим, что эти каскады могут быть двунаправленными, то есть работать в качестве входного или выходного вывода в зависимости от управляющего сигнала.

•1947 - Появление первых транзисторов. Изначально в них применялся германий, который, однако, вскоре был заменен более дешевым кремнием.
•1958 - Изобретение интегральных схем (ИС) общего назначения.
•1962 - Изобретение нового типа транзисторов - КМОП-транзисторы. Они работали немного медленнее, однако были дешевле, меньшего размера и потребляли существенно меньше энергии.
•1967 - Появление первых ASIC-микросхем под названием Micromosaic от компании Fairchild. Однако широкого распространения они не получили.
•1970 - Компания Fairchild представляет первую 256-битную SRAM-микросхему.
•1970 - Разработан первый тип SPLD — микросхема ППЗУ.
•1971 - Изобретение нового типа ROM(Read-Only Memory)-памяти — EPROM. Первое постоянное запоминающее устройство, которое обладает возможностью перезаписи. Осуществлялась перезапись под воздействием ультрафиолетовых лучей.
•1975 - Появление нового типа SPLD — PLA (Programmable Logic Arrays). В основе PLA лежит развитие идеи микросхем ППЗУ.
•Конец 70-х - Разработан ещё один класс SPLD — программируемый массив логики (PAL - Programmable Array Logic). На понятийном уровне устройства PAL почти полностью противоположны микросхемам ППЗУ.
•Конец 70-х - Началось активное использование ASIC-микросхем в связи с появлением нового типа данных устройств — вентильных матриц. Первая вентильная матрица называлась некоммутированная логическая матрица (ULA).
•1983 - Появление памяти EEPROM, главным отличием которой от EPROM-микросхем является высокая скорость перезаписи. Это достигается за счет стирания данных электрическим способом.
•1983 - Разработано устройство GAL — продолжение идеологии PAL.
•1984 - Впервые продемонстрирована технология Flash, основанная на EPROM- и EEPROM-технологиях.
•1984 - Появились первые сложные программируемые логические устройства (CPLD) от созданной в этом же году компании Altera. Они представляют собой несколько CPLD, соединённых специальной программируемой логической матрицей.
•1984 - Основание компании Xilinx. Росс Фриман из Xilinx разработал новый класс микросхем — FPGA-микросхемы.
•1985 - Запуск производства первой FPGA-микросхемы, Xilinx XC2064 с 1000 вентилей. В ней применялась 3-микронная технология.
•1988 - Основана компания Actel.
•1992 - NASA впервые применила FPGA в космических технологиях.
•1994 — Компания Altera перестала заниматься разработкой исключительно CPLD и вышла на рынок FPGA с серией микросхем FLEX.
•1995 — Altera и Xilinx начали выпуск FPGA-микросхем, обладающих достаточной мощностью для выполнения цифровой обработки сигналов. С этого момента FPGA начали активно использоваться в данной области.
•1996 — Actel, начав выпуск микросхем RH1280, стала первой компанией, занимающейся производством радиационно-стойких FPGA-микросхем.
•1998 - Появление семейства Virtex от компании Xilinx.
•1999 — Компания Altera выпустила FPGA с 1.5 миллионами вентилей - APEX EP20K.
•2000 — Actel запустила семейство ProASIC на Flash-технологии (первая серия - ProASIC 500K).
•2002 — Компания Lattice Semiconductor купила ПЛИС сектор компании Agere Systems и на базе полученных ресурсов начала разработку и производство FPGA- и FPSC-микросхем.

•2003 - В производстве FPGA произошел переход на 90-нм технологию (серия Spartan-3 от Xilinx). Число вентилей превысило 5 миллионов.
•2004 — Компания Cray выпустила ПЛИС-сопроцессоры XD1.
•2005 — Компания Actel представляет технологию Fusion, которая позволяет объединить логические блоки FPGA, Flash-память и аналоговые устройства на одной микросхеме.
•2006 — Появление серии Virtex-5 от Xilinx на основе 65-нм технологии. Число вентилей достигло 7 миллионов (Virtex-5 V5LX33OT).

10. Спеціальні апаратні засоби ПЛІС.

ПЛИС представляет собой полуфабрикат, на основе которого разработчик, обладающий персональным компьютером, несложными и относительно недорогими аппаратными средствами программирования и специальным программным обеспечением, называемым системой автоматизированного проектирования (САПР), имеет возможность проектирования цифрового устройства в рекордно короткие сроки.

В отличие от МП(микропроцессоры), ПЛИС имеют преимущество в гибкости разработки системы на кристалле и наличии внутренней памяти. Такие преимущества могут быть использованы для:

1) расширения структуры обеспечения отказоустойчивости и обеспечения широкого охвата ошибок вычислительного процесса на базе кристалла;

2) обеспечения последовательности восстановления типа "остановка - фиксация - перезапуск";

3) отключения не всего вычислителя, а его частей, являющихся причиной ошибок.

Перечисленные выше решения являются также удобными с учетом большой гранулярности ПЛИС, благодаря чему можно достичь высокой элементарности действий. Такие элементарные действия являются алгоритмическими средствами, с помощью которых еще при проектировании разработчик системы может определить, какие взаимодействия и пересечения процессов нужно предотвратить (по возможности), чтобы сохранить целостность работы системы. Таким образом, разделяя главный вычислительный процесс и формируя его в определенной последовательности элементарных действий, проектировщик получает возможность создавать сложные вычислители с необходимой степенью обнаружения ошибок в его вычислительном процессе и управления их распространением.

Учитывая гибкость проектирования с использованием ПЛИС-технологии, становится возможным и удобным не только построение условно-распределенного вычислительного комплекса, но и построение модуля обнаружения неисправности и управления переключением на базе одного кристалла. Это, в свою очередь, позволит реализовать аппаратно-управляемое восстановление, которое не будет выходить за рамки кристалла (системного модуля).