1. Причини та передумови виникнення мов HDL.

С начала 70-х годов стала актуальной проблема создания стандартного средства документации схем и алгоритмов дискретных систем переработки информации и цифровой аппаратуры (ЦА), одинаково пригодного как для восприятия человеком, так и для обработки в ЭВМ.

Этим средством явились языки VHDL и VERILOG, обобщенно называемые HDL (Hardware Description Language — язык описания аппаратуры).

К основным достоинствам языков VHDL и VERILOG следует отнести следующие:

1. *Стандартность.* Лучше иметь плохой стандарт, чем не иметь никакого. Это подтверждает библейский опыт создания вавилонской башни, разноязычие строителей которой, по преданиям, привело к печальному результату.

VHDL и VERILOG официально признаны стандартом описания цифровой аппаратуры, который поддерживается военно-промышленным комплексом и радиоэлектронной промышленностью западных стран. Этот стандарт облегчает обмен документацией между отдельными группами разработчиков и эксплуатационщиков аппаратуры, различными системами автоматизации проектирования (САПР).

2. *Многоаспектность и многоуровневость.* Универсальное средство заменяет несколько специализированных. Языки VHDL и VERILOG пригодны для описания как схем аппаратуры, так и функциональных тестов и алгоритмов функционирования. Они покрывают широкий диапазон уровней структурной детализации описаний ЦА: от описаний архитектуры ЭВМ на уровне устройств типа процессор-память до описаний узлов типа триггер на уровне вентилей и МОП-ключей, от описаний алгоритмов ЭВМ на уровне команд до описаний алгоритмов устройств на уровне межрегистровых передач и булевских функций, от описаний функциональных тестов до тестов проверки схем. На высших уровнях абстракции VHDL- и VERILOG-описания можно рассматривать как средство спецификации требований к проекту.

3. *Человеко-машинность.* Документация пишется один раз, а читается многократно. Создатели VHDL и VERILOG нашли довольно удачный компромисс между требованиями к языку как к средству документирования, удобному для восприятия человеком, и как формальному средству, удобному для ввода и обработки описаний систем в ЭВМ. VHDL- и VERILOG-описания аппаратуры пригодны для обработки такими компонентами САПР, как подсистемы моделирования, подсистемы формальной верификации, reuse-checker'ы (программы проверки стиля HDL-описаний на синтезабельность и пригодность к повторному использованию в различных проектах), подсистемы логического синтеза, подсистемы синтеза с учетом тестопригодности, системы синтеза и анализа контрольных тестов, временные анализаторы, кремниевые компиляторы, подсистемы автоматизации конструкторского проектирования и т. п.

2. Аспекти та рівні опису апаратури мовами HDL\

3. Стисла характеристика найпоширеніших мов HDL.

VHDL изначально предназначаясь для облегчения унифицированного описания микросхем, которые включались сторонними поставщиками в различные решения для этого ведомства. VHDL – это строго типизированный язык, не чувствительный к регистру символов. В дополнение к стандартным базовым возможностям Ada, VHDL включает расширенные логические операции (например, nand и nor), двунаправленную индексацию массивов, а также дополнительные типы, такие как time, bit, bit_vector, character, string. Позже в VHDL ввели понятие 9-значной (U,X,0,1,Z,W,H,L,-) логики (см. IEEE Std 1164 [25]) и понятие знаковых/беззнаковых типов (см. IEEE standard 1076.3 [26]). Принципиальной особенностью VHDL является поддержка конструкций для задания параллелизма, свойственного аппаратуре, а именно модулей и процессов. Интерфейс модуля задается с помощью ключевого слова entity, ключевое слово architecture обозначает описание реализации, которое заключается между begin и end. Внутри такого блока могут задаваться константы (constant), сигналы (signal) и собственно поведение в виде набора операторов, в том числе, сгруппированных в виде параллельно выполняющихся процессов (с помощью ключевого слова process). Внутри процессов могут объявляться переменные (variable). Важным различием переменных и сигналов является то, что значение переменной меняется сразу после выполнения соответствующего оператора (понятие времени не ассоциируется с понятием переменной), а значение сигнала меняется только после окончания текущей итерации выполнения процесса.

Язык **Verilog** был разработан компанией Gateway Design Automation в 1985 году для целей проектирования интегральных схем на логическом уровне. Verilog наследует многое из языка C, поддерживает конструкции препроцессора C и большинство основных управляющих конструкций, таких как “if”, “while” и т.п. Verilog полностью поддерживает операторы языка C, но также обладает дополнительными возможностями для удобной обработки двоичных данных (например, операциями ~^ для XNOR или >>> для арифметического сдвига с заполнением знаковым битом). Однако в Verilog не поддерживаются пользовательские типы, структуры, указатели и рекурсивные функции. Типы данных в Verilog имеют явную битовую ширину. В отличие от VHDL, Verilog является слабо типизированным языком, что позволяет смешивать присвоения элементов разного типа за счет неявного преобразования типов. Как и в случае VHDL, принципиальным отличием Verilog от своего языка-прототипа является поддержка конструкций, выполняющихся параллельно. Модули в Verilog определяются с помощью ключевого слова module и могут быть вложенными. Не происходит явного разделения интерфейса и реализации, как в VHDL. В декларативной части модуля определяются входные (input), выходные (output), двунаправленные (inout) порты, внутренние сигналы (wire) и переменные (reg).

4. Метод імітаційного моделювання.

Основы метода имитационного моделирования

Проектирование можно представить как чередующийся процесс порождения новых вариантов системы — синтез, и оценки, проверки этих вариантов — анализ.

Одним из эффективных методов анализа сложных систем является метод имитационного моделирования. Этот метод предполагает построение математической модели системы в виде алгоритма, отображающего существенные ее свойства и реализацию модельного эксперимента на ЭВМ. Большая емкость памяти и быстродействие современных ЭВМ позволяют имитировать поведение систем, состоящих из десятков и сотен тысяч элементов, воспроизводить процессы, включающие миллионы элементарных актов.

В процессе проектирования аппаратуры и математического обеспечения (МО) цифровых систем (ЦС) этим методом решается большое число задач.

1. На этапе структурного проектирования: анализ и оценка вариантов структур ЦС и МО, рассматриваемых как система массового обслуживания с точки зрения производительности, быстродействия, времени обслуживания заявок (задач потребителей), максимальной длины очередей на обслуживание, выявления узких мест системы и т. п. (языки GPSS, GASP, SIMULA).

2. На этапе алгоритмического проектирования на имитационных моделях проверяются алгоритмы отдельных устройств. Для ЦВМ с микропрограммным управлением отлаживаются микропрограммы. На интерпретационных комплексах отлаживаются элементы МО проектируемой ЭВМ-программы технического обслуживания (тесты), компоненты операционной системы и т. п. (наряду с языками *программирования C, C++, PASCAL, здесь часто используются VHDL и VE*-RILOG).

3. На этапе функционально-логического проектирования моделирование используется при проверке функциональных описаний устройств (так называемых описаний на уровне регистровых передач (register transfer level -RTL) и отработке функциональных тестов. Кроме того, оно применяется при верификации автоматически синтезируемых логических схем (HDL-языки — VHDL, VERILOG, их подмножества и их модификации типа AHDL).

4. На этапе конструкторского проектирования имитационное моделирование помогает проверять схемы с учетом задержек в проводах, анализировать и синтезировать тесты контроля и т. п. (HDL-языки — VHDL,VERILOG, стандартные форматы EDIF, SDF и др.).

5. На стадии проектирования элементной базы этот метод используется при расчете электрических схем элементов, анализе переходных процессов и исследовании влияния разброса параметров электронных компонент (pSpice и др.).

В процессе исследования системы методом имитационного моделирования можно выделить ряд этапов:

1. Постановка задачи — определение границ исследуемой системы, целей исследования и путей их достижения.
2. Планирование исследования, подготовка исходных данных.
3. Формализация моделируемого процесса и построение его математической модели.
4. Построение моделирующего алгоритма.
5. Создание теста, отладка и верификация модели.
6. Реализация экспериментов на модели.
7. Оптимизация модели и при необходимости постановка новых экспериментов на ней.
8. Анализ конечных результатов и их документирование.

Проведение машинного эксперимента с моделями сложных систем существенно упрощается при использовании проблемно-ориентированных языков моделирования и соответствующих пакетов прикладных программ (ППП). Для этих языков характерно присутствие средств, автоматизирующих выполнение отдельных этапов модельного эксперимента. Например, наличие в языке моделирования встроенных объектов-аналогов соответствующих элементов моделируемых структур и процессов позволяет упростить и совместить этапы 3 и 4, наличие в ППП развитых средств построения тестов и отладки упрощает этап 5, наличие в ППП подсистем планирования и оптимизации помогает автоматизировать этапы 6 и 7 модельного эксперимента.

Рассматриваемые в книге языки VHDL и VERILOG являются классическим примером таких языков моделирования.

5-6. Модельний час. Імітація паралельних процесів.

Модельное время и имитация параллельных процессов

Особенности инструмента, с помощью которого реализуется модельный эксперимент (в нашем случае это ЭВМ), определенным образом сказываются на структуре описания модели и характере погрешностей.

В структурном аспекте моделируемая система (объект проекта — object, entity) представляется как композиция из элементов-компонент (component) и связей между ними (net). Компонентом принято считать объект, внутренняя структура которого не представляет интерес на принятом уровне рассмотрения.

В поведенческом аспекте моделируемая система представляется множеством процессов (process), взаимодействующих путем обмена сигналами (signal). Изменение сигнала — событие (event) в одном процессе может запускать другие процессы, которые, в свою очередь, порождают следующие события.

В реальной системе элементы могут работать одновременно, процессы протекать параллельно. При использовании обычных однопроцессорных ЭВМ для имитации таких систем приходится последовательно воспроизводить функционирование их элементов. Это приводит к необходимости введения *дискретизации модельного времени* и применения специальных мер для уменьшения погрешностей, обусловленных последовательной организацией моделирующего алгоритма. *Масштаб модельного времени* определяется *интервалом* ΔT реального времени, принятого в модели за единицу. В языке VERILOG пользователь сам задает *единицу модельного времени* и точность ее представления директивой `timescale, например, `timescale 1ns/100 ps, что означает — одна наносекунда с точностью 100 пикосекунд. В языке VHDL это делается неявно.

Следует отличать tm — *модельное время в условных единицах* (число интервалов ΔT) от tmr — *модельного времени в физических единицах времени* (tmr = tm · ΔT) и tc — *машинного времени* (продолжительности выполнения моделирующей программы на компьютере), затраченного на воспроизведение tm или tr единиц модельного времени.

Коэффициент замедления моделирования определяет отношение машинного времени к воспроизводимому (модельному) времени. Например, имитация 100 микросекунд работы логической схемы может потребовать трех секунд машинного времени (коэффициент замедления 30 000). Если интервал модельного времени равен 0,5 микросекунды, то 100 микросекунд реального времени соответствуют 200 единицам модельного времени.

Ошибка, связанная с дискретизацией модельного времени, может быть сделана сколь угодно малой за счет выбора более мелкого шага времени ΔT. Однако во многих случаях это связано с увеличением коэффициента замедления и дополнительными затратами ресурсов памяти и времени инструментальной ЭВМ. Поэтому при выборе масштаба модельного времени приходится идти на компромисс. Для действий, имеющих продолжительность меньшую, чем единица модельного времени, мы как бы допускаем в модели возможность существования причинно-следственных отношений вне времени. Это приводит к необходимости включения в HDL специальных понятий и средств, таких, как неявная бесконечно малая дельта-задержка в операторах присваивания и дельта-цикл итерации в работе моделирующей программы.

Специфика задач, возникающих в связи с дискретизацией модельного времени и имитацией параллельно протекающих процессов последовательно выполняемой на ЭВМ программой, может быть пояснена следующим примером. Пусть модель состоит из упорядоченного подмножества связанных между собой компонентов-моделей элементов Эк (к = 1 : n), не обладающих задержками. Состояние выхода некоторого K-го элемента есть функция состояний его соседей, часть из которых имеет больший, часть — меньший индекс K. Если вычисление значений новых состояний элементов, соответствующих новому T + delta-моменту модельного времени осуществляется в порядке возрастания индексов элементов, то при вычислении нового состояния Эi предшествующие (1: (i − 1)) элементы уже приняли новое состояние, в то время как оставшиеся ((i + 1) : n) еще находятся в старом. Поэтому для устранения возможной ошибки часто используются итерации (дельта-цикл). **Только после того как вычислится новое состояние всех элементов, система моделирования произведет смену старых состояний на новые, увеличит модельное время на бесконечно малое дельта и снова повторит дельта-цикл.** И так до тех пор, пока состояния элементов перестанут меняться или не будет превышен предел допустимого количества дельта-циклов (итераций). Пример системы с несходящимися итерациями — наличие среди ее элементов генератора типа Y = not Y.

7. Верифікація опису засобами HDL.

Верификация

Верификация проекта — это процесс доказательства того, что он функционирует согласно его спецификации.

Различают методы **имитационной** и **формальной верификации**.

Имитационная верификация — традиционная идеология верификации с использованием моделирования. Она предполагает моделирование HDL- описания проектируемого объекта и воспроизведение в ходе имитационного эксперимента всех (или части) возможных внешних входных воздействий и внутренних состояний моделируемой системы.

Формальная верификация (см., например, пакеты фирмы Verplex — www.verplex.com) обычно предполагает использование HDL-описания как формальной модели с целью доказательств ее корректности и эквивалентности или неэквивалентности спецификации проекта.

Ниже рассматривается только имитационная верификация.

Этапы верификации

Процесс верификации начинается с разработки тест-плана (как предполагается тестировать, каким инструментарием, в каком порядке и когда, простые и граничные тесты, запрещенные и провокационные режимы и т. п.). Затем следуют:

— создание верификационной среды (test bench);

— отладка тестирующей программы и модели исследуемого объекта;

— профилирование (выявление и по возможности устранение сильно снижающих быстродействие участков теста);

— проведение регрессионных (многократных, исчерпывающих) экспериментов.

8. Оцінки повноти тестів.

4.3. Оценка полноты функциональных тестов

4.3.1. Эвристические метрики

Эвристические метрики основаны на текущей статистике обнаружения ошибок в проекте. Подобные метрики включают:

— календарное время между моментами обнаружения ошибок проекта (к концу процесса верификации оно увеличивается);

— общее количество промоделированных тактов работы проектируемого устройства;

— общее число обнаруженных ошибок в проекте и т. д.

В свое время, когда автор работал в компании Raycer Graphics над проектом мощного профессионального графического ускорителя, состоящего из трех функциональных блоков-кристаллов, шеф верификационной команды Рик Авра (Rick Avra) оценил вероятное число ошибок в 3000, и, когда этот рубеж был достигнут в условиях успешной работы всех трех RTL-моделей блоков на сотнях тестов (от изображения простого голубого треугольника до изображений автомобиля с текстуровкой, тенями и полутонами), все вздохнули с облегчением. Психологический барьер был пройден и стал виден конец работы, так как темп обнаружения ошибок коллективом из десятков человек стал менее одной ошибки в день. Однако такие метрики весьма необъективны, и возможно, что часть функций проектируемого устройства осталась бы неверифицированной.

4.3.2. Программные метрики

В настоящее время большинство коммерческих систем оценки полноты функциональных тестов COVERSCAN, COVERMETER и т. п., а также развитые системы моделирования типа SILOS, ACTIVE-HDL, MODELSIM, VSS базируются на метриках, используемых для верификации программ (программо-метрический подход).

В числе таких метрик можно указать, например, такие:

— полнота покрытия тестом строк кода модели — количество исполнений каждой строки описания;

— полнота покрытия переходов — число исполнений ветвей операторов if и case;

— полнота покрытия путей — число исполнений всех возможных путей в графе программы;

— полнота покрытия выражений — низкоуровневая метрика, основанная на оценке числа вычислений выражений на различных наборах данных;

— полнота переключений (0 ->1 и 1 ->0) каждого бита данных.

Пример подобных статистических данных, получаемых системой SILOS-demo при прогоне VERILOG-варианта вышеприведенного теста описания объекта проекта F, представлен на рис. 4.5. Видно, что все строки кода тест-программы исполнялись, однако тест представляется неполным, так как выход B2 не перебрасывался из 1 в 0, а B1 из 0 в 1.

Ясно, что, если после прогона теста собранная статистика говорит, что часть операторов модели вообще ни разу не исполнялась или некоторые сигналы ни разу не переключались, имеет смысл попытаться понять причину этого явления и при необходимости дополнить тест. Несмотря на очевидную полезность и объективность этих метрик, никто не может гарантировать, что достижение, допустим, 100% покрытия тестом программного кода позволяет обнаружить все ошибки проекта устройства.

4.3.3. Автоматно-метрический подход

Если проектируемое устройство является конечным автоматом, то можно использовать для оценки полноты теста статистику о числе достижений автоматом различных состояний (вершин) и переходов (дуг на графе автомата). Однако, так же как и для программо-метрик, автомато-метрическая полнота — это необходимое (управляемость), но не достаточное (наблюдаемость) условие полноты теста. Более мощные результаты дают системы формальной верификации автоматов, обнаруживающие зависания, недостижимые состояния и т. п.

4.3.4. Моделирование неисправностей

Эти методы базируются на идее обнаружения тестом множества возможных неисправностей в моделируемой схеме (система моделирования SILOS 3 и др.).

В модель вносится неисправность, и если выходы неисправной системы не отличаются от эталона, т. е. на всем времени моделирования неисправность не обнаруживается тестом, то тест неполный. Эти методы хорошо зарекомендовали себя в моделях вентильного уровня (неисправности типа тождественный 0 или 1 на входе/выходе вентиля), но для моделей функционального уровня они недостаточны. Неясно, например, как моделировать неисправности оператора CASE, и трудно представить, что все возможные неисправности в микросхеме процессора INTEL сводятся к установке одного из внешних контактов микросхемы в 0 или 1.

4.3.5. Мониторинг событий и проверка контрольных соотношений в модели

Эти верификационные методики базируются на том, что для разработчика модели устройства она является «прозрачным ящиком» и он имеет возможность включить в модель дополнительные фрагменты текста (assertion — мониторы и утверждения), контролирующие соблюдение функциональных ограничений.

Пример некорректных режимов — запрет одновременного прихода сигналов сброса и установки на вход триггера или наличия двух открытых буферов на тристабильной шине и т. п.