**Оглавление**

**1**[**. Введение в предметную область**](#h.30j0zll)

[**1.1. Понятие информационной системы. Основные виды информационных систем, примеры.**](#h.1fob9te)

[**1.2. Типовой состав ИУС РВ. Особенности построения и функционирования ИУС РВ.**](#h.3znysh7)

[**1.3. Задачи разработки ИУС РВ, требующие инструментальной поддержки.**](#h.2et92p0)

[**2. Производительность ИС. Методы наблюдения за работой ИС и измерения производительности**](#h.tyjcwt)

[**2.1. Аппаратные наблюдатели. Основные возможности.**](#h.3dy6vkm)

[**2.2. Программные средства наблюдения. Примеры. Средства измерения времени в ОС.**](#h.1t3h5sf)

[**2.3. Основные возможности средств отображения и анализа трасс (как с аппаратных, так и с программных наблюдателей).**](#h.4d34og8)

[**2.4. Понятие производительности ИС. Роль наблюдения в анализе производительности и отладке программы.**](#h.2s8eyo1)

[**3. Основы имитационного моделирования**](#h.17dp8vu)

[**3.1. Понятие модели. Основные виды моделей. Примеры. Основные варианты применения моделей.**](#h.3rdcrjn)

[**3.2. Понятие имитационной модели. Сравнение по возможностям с другими видами моделей. Способы продвижения времени в имитационных моделях.**](#h.26in1rg)

[**3.3. Основные этапы создания имитационной модели. Сравнение с этапами создания программы «общего назначения»**](#h.lnxbz9)

[**3.4. Подходы к описанию имитационной модели: событийный, процессно-ориентированный, агентно-ориентированный. Основные понятия, поддерживаемые языками и библиотеками классов имитационного моделирования**](#h.35nkun2)

[**3.5. Обобщённая архитектура системы имитационного моделирования. Краткое описание функций её компонентов**](#h.1ksv4uv)

[**4. Важные классы средств имитационного моделирования, специфичные для ИС**](#h.44sinio)

[**4.1. Эмуляторы процессоров. Назначение. Классификация по точности.**](#h.2jxsxqh)

[**4.2. Способы быстрой эмуляции процессоров.**](#h.z337ya)

[**4.3. Основные виды сверхбольших интегральных схем (СБИС). Уровни моделирования и проектирования аппаратных средств ИС.**](#h.3j2qqm3)

[**4.4. Моделирование на уровне RTL. Учёт специфики предметной области в языке Verilog.**](#h.1y810tw)

[**4.5. Моделирование на системном уровне. Учёт специфики предметной области в библиотеке SystemC. Понятие о стандарте TLM.**](#h.4i7ojhp)

[**Модули (module)**](#h.tp1let6kiq1t)

[**Сигналы (signal)**](#h.73xqagvh5vcd)

[**Порты (port)**](#h.fqknpdu6ijku)

[**Процессы (process)**](#h.2kjw24qanbbe)

[**Каналы (channel)**](#h.v4t7icxr5cdi)

[**Интерфейсы (interface)**](#h.5neg6umkvpft)

[**События (events)**](#h.1crpi0b7z60u)

[**5. Моделирование и проектирование ИС. Построение сложных имитационных моделей**](#h.2xcytpi)

[**5.1. Взаимосвязь моделирования и проектирования ИС.**](#h.1ci93xb)

[**5.2. Понятие совместной разработки программной и аппаратной части ИС. Достоинства по сравнению с традиционным процессом разработки. Виртуальный прототип.**](#h.3whwml4)

[**5.3. Основные понятия стандарта HLA взаимодействия имитационных моделей**](#h.2bn6wsx)

[**6. Элементы статистических методов для моделирования и анализа производительности**](#h.qsh70q)

[**6.1. Понятие системы массового обслуживания, её основные элементы, характеристики производительности.**](#h.3as4poj)

[**Основные понятия СМО:**](#h.7khqyfoknnv1)

[**6.2. Система М/М/1. Пуассоновский поток. Средняя длина очереди. Формула Литтла.**](#h.1pxezwc)

[**6.3. Обработка результатов эксперимента: оценка необходимого числа испытаний.**](#h.49x2ik5)

[**6.4. Общая схема проверки статистических гипотез, пример.**](#h.2p2csry)

# **1. Введение в предметную область**

## **1.1. Понятие информационной системы. Основные виды информационных систем, примеры.**

(лекция 1)

ИС

- автоматизированная система, предназначенная для организации, хранения, пополнения, поддержки и представления пользователям информации в соответствии с их запросами.

- система, которая организует хранение и манипулирование информацией о предметной области

- совокупность содержащейся в базах данных информации и обеспечивающих ее обработку информационных технологий и технических средств;

Классификация ИС

По степени автоматизации ИС делятся на:

* автоматизированные: информационные системы, в которых автоматизация может быть неполной (то есть требуется постоянное вмешательство персонала);
* автоматические: информационные системы, в которых автоматизация является полной, то есть вмешательство персонала не требуется или требуется только эпизодически.

По масштабу:

* + Персональные - предназначены для решения некоторого круга задач одного человека
  + Групповые - ориентированы на коллективное использование информации членами рабочей группы или подразделения.
  + Корпоративные - в идеале охватывает все информационные процессы целого предприятия, достигая их полной согласованности, безызбыточности и прозрачности. Такие системы иногда называют *системами комплексной автоматизации предприятия.*

По характеру обработки данных ИС делятся на:

* информационно-справочные, или информационно-поисковые ИС, в которых нет сложных алгоритмов обработки данных, а целью системы является поиск и выдача информации в удобном виде;
* ИС обработки данных, или решающие ИС, в которых данные подвергаются обработке по сложным алгоритмам. К таким системам в первую очередь относят автоматизированные системы управления и системы поддержки принятия решений.

Примеры:

* Правовые документы (Гарант)
* WWW + поиск
* Бухгалтерия (1C)
* Склад, магазин
* Биржа, банки
* Предприятие в целом (1С, SAP R/3)
* Транспортные компании
* Управление войсками на поле боя
* Управление подвижными объектами
  + Наземные
  + Морские
  + Воздушно-космические

## **1.2. Типовой состав ИУС РВ. Особенности построения и функционирования ИУС РВ.**

(лекция 1)

ИУС РВ – информационно-управляющая система реального времени. Информационно-управляющие системы реального времени (ИУС РВ) используются для управления сложными техническими системами. Например, летательными аппаратами, кораблями, искусственными спутниками и т.п. Одним из важнейших требований к функционированию ИУС РВ является выполнение ограничений реального времени

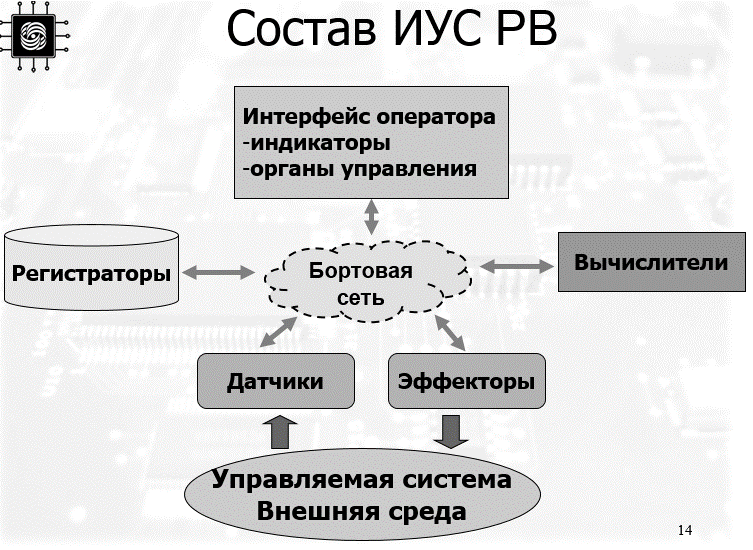
Для нормального функционирования объекта необходимо, чтобы реакция системы на поступающую от него информацию была предсказуемой и своевременной. Поскольку в общем случае процессы, происходящие на объекте, носят непредсказуемый характер, информационно-управляющую систему реального времени можно определить, как систему, реагирующую предсказуемым образом в предсказуемое время на непредсказуемо событие или непредсказуемый поток событий. Таким образом, предсказуемость и своевременность реакции – необходимые характеристики ИУС РВ. При этом для квалификации системы как системы реального времени абсолютное значение времени реакции само по себе не важно: это могут быть и доли микросекунд и десятки секунд. Важно то, что в зависимости от конкретного приложения это время заранее предсказано (вычислено) и является критерием и необходимым условием правильного функционирования системы. Таким образом, время реакции системы реального времени – характеристика всегда конкретная и связана с конкретным объектом, в чем заключается коренное отличие системы реального времени от системы просто “быстрой”.

Обощенная схема ИС:

* Рабочие места пользователей
* База данных
* Бизнес-логика
* Служебное ПО
* Вычислительные средства
* Средства хранения данных
* Каналы связи

Встроенные ИУС РВ

* Системы персонального масштаба – вездесущи
* Системы масштаба группы или предприятия:
  + Управление производственными процессами
  + Управление подвижными объектами



Специфика ИУС

* Интеграция с управляемой системой
* Критичность для управляемой системы
* Ограниченное участие оператора
* Работа в реальном времени
* Непрерывное функционирование
* «Экстремальные» условия работы
* Ограничения по ресурсам
* Устойчивость к сбоям

## **1.3. Задачи разработки ИУС РВ, требующие инструментальной поддержки.**

(лекция 1)

Жизненный цикл:

Жизненный цикл создания (разработки) и использования ИВС представляет собой последовательность стадий работ, включающих однородные по содержанию и результатам этапы работ.

* Проведение научно-исследовательских работ - обоснование состава решаемых задач, структуры и состава ИВС и подготовка проекта ТЗ на создание (разработку)
* Проектирование (эскизное, техническое) ИВС
* Реализация проекта (рабочее проектирование опытного образца ИВС)
* Внедрение (адаптация) опытного образца ИВС в конкретных условиях применения
* Эксплуатация ИВС
* Сопровождение
* Снятие с эксплуатации

Особенности ИС

* Сложность как ИС, так и процесса разработки
* Высокие требования к производительности и правильности функционирования
* Необходимость сопровождения (обновление программных и аппаратных средств в процессе использования)

Поддержка разработки ИС

* Необходимость соблюдения стандартов на процесс разработки
* Необходимость применения инструментальных средств поддержки разработки, в том числе средств исследования функционирования ИС в целях:
  + определения производительности
  + проверки правильности работы

Инструментальные средства:

* разработка требований
* управление версиями
* отслеживание проблем и изменений
* поддержка сопряжения подсистем ПО
* проектирование индикационных форматов
* проектирование алгоритмов
* верификация и тестирование ПО ИУС

Мат задачи:

* Выбор оптимальной конфигурации ИУС РВ
  + требования реального времени
  + требования надёжности
  + ограничения по ресурсам
* Построение расписания вычислений
* Построение расписания обмена
* Верификация работы ИУС РВ

Генерация тестовых покрытий

# **2. Производительность ИС. Методы наблюдения за работой ИС и измерения производительности**

## **2.1. Аппаратные наблюдатели. Основные возможности.**

(лекция 2)

Наблюдение за работой системы

* Аппаратные наблюдатели
* Программные наблюдатели
* Анализ результатов наблюдения

Аппаратные наблюдатели подразделяются на внутренние и внешние. Внешний аппаратный наблюдатель подключается к определенным точкам системы, "подслушивает" сигналы на ее линиях, обрабатывает и записывает их у себя, вне измеряемой системы. Аппаратный наблюдатель представляет собой совершенно автономную систему, которая не нуждается ни в какой помощи со стороны измеряемой системы. Он практически не вмешивается в ее работу, а, стало быть, не изменяет ее поведения.

Аппаратные:

* Встроенные в аппаратуру выч. сист.
  + Отладочные регистры процессора
  + JTAG
* Универсальные внешние
  + Электронный (цифровой) осциллограф
  + Логический анализатор
* Специализированные
  + Анализаторы периферийного интерфейса

Отладочные регистры:

* 6 регистров: DR0-DR3,DR6,DR7
* Точки останова (4 шт.):
  + по чтению;
  + по записи и чтению;
  + по выполнению;
  + по обращению к порту ввода-вывода.
* Нельзя задать условие по диапазону адресов

**JTAG** (сокращение от англ. *Joint Test Action Group*; произносится «джей-тáг») — специализированный аппаратный интерфейс на базе стандарта IEEE 1149.1

На текущий момент интерфейс стал промышленным стандартом. Практически все сколько-нибудь сложные цифровые микросхемы оснащаются этим интерфейсом для:

* выходного контроля микросхем при производстве
* тестирования собранных печатных плат
* прошивки микросхем с памятью
* отладочных работ при проектировании аппаратуры и программного обеспечения

Метод тестирования, реализованный в стандарте, получил название Boundary Scan (граничное сканирование). Название отражает первоначальную идею процесса: в микросхеме выделяются функциональные блоки, входы которых можно отсоединить от остальной схемы, подать заданные комбинации сигналов и оценить состояние выходов каждого блока. Весь процесс производится специальными командами по интерфейсу JTAG, при этом никакого физического вмешательства не требуется. Разработан стандартный язык управления данным процессом — Boundary Scan Description Language (BSDL).

JTAG:

* Набор специальных блоков в составе ИС (существенно зависит от типа ИС)
* Управление этими блоками по интерфейсу JTAG
* Возможность разрывать связи между блоками ИС, устанавливать заданные значения на разорванных связях
* Управление микропроцессором в целях отладки: точки останова, просмотр и запись регистров

Универсальные наблюдатели:

* Индикаторы

Осцилло́граф — прибор, предназначенный для исследования (наблюдения, записи; измерения) амплитудных и временны́х параметров электрического сигнала, подаваемого на его вход, либо непосредственно на экране, либо записываемого на фотоленте.

* Электронный осциллограф
  + Непрерывные и дискретные сигналы
  + Обычно 1-4 канала
  + Непрерывная и ждущая развертка
  + Задержка запуска развертки

Логический анализатор (англ. Logic Analyzer) - представляет собой электронный прибор, который может записывать и отображать последовательности цифровых сигналов. Он используется для тестирования и отладки цифровых электронных схем, например, при проектировании компонентов компьютеров и управляющих электронных устройств. В отличие от осциллографов, логические анализаторы имеют значительно больше входов (обычно от 16 до нескольких сотен), но при этом часто способны показывать лишь два уровня сигнала ("0") и ("1"), к которым иногда добавлено состояние "Z" ("высокое сопротивление").

* Логический анализатор
  + Цифровой сигнал
  + Десятки-сотни каналов
  + Задание условий запуска и окончания регистрации

Шинный интерфейс:

* Объединяет несколько устройств (абонентов) (десятки-сотни)
* Один набор сигнальных линий для обменов между всеми абонентами *(есть исключения для служебных линий)*
* В каждый момент времени возможна передача только одним абонентом
* Процедура арбитража для избежания конфликтов
* Стандартизованы: протоколы, электрические и механические характеристики

Настройки анализатора шины:

* Условия начала и окончания регистрации
* Что регистрировать (фильтр)
* Как отображать

Анализатор: интерфейс пользователя

* Прокрутка
* Измерение интервалов времени
* Фильтр по группам сигналов шины
* Поиск по группам сигналов шины
* Сохранение трассы
* Сравнение трасс
* Программное управление (API)

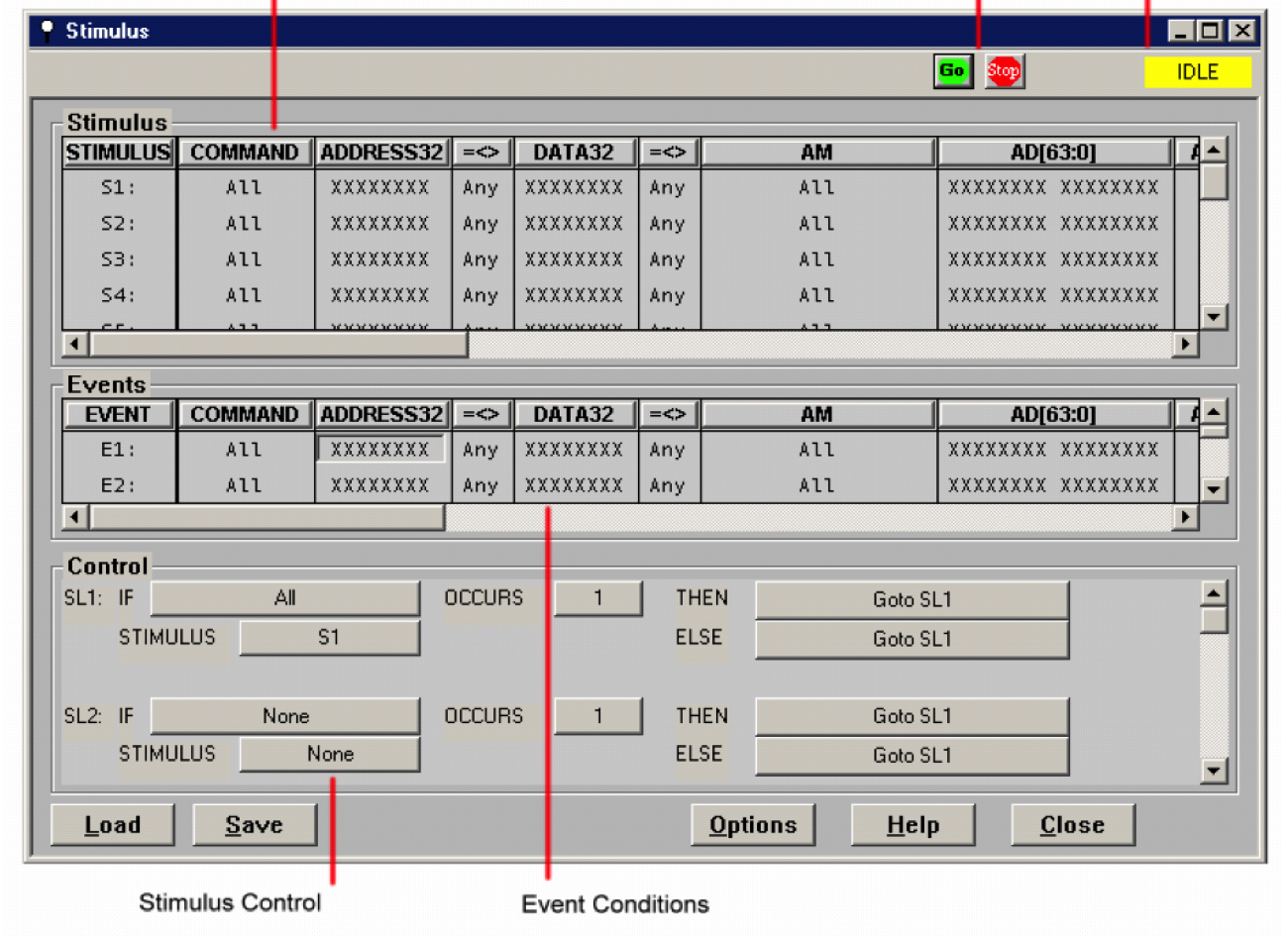
Анализ производительности:

* Загруженность
  + BusBusy, BusIdle, DataPhase, WaitState
* Пропускная способность
  + Все обмены, или по приоритетам
  + По диапазонам адресов
* Запаздывание
  + Master от выставления адреса до первого слова данных
  + Target от запроса первого слова данных до выставления
  + Arbitration от запроса до разрешения

Имитация абонента шины:

* Инициатор (master)
  + Адрес, длина
  + Режим (пословный, пакетный)
  + Запись, чтение, запись с проверкой
* Подчиненное устройство (target)
  + Диапазон адресов
  + Число тактов ожидания
  + Ответ (OK, retry, fail)
  + Инициализация памяти
  + Просмотр и сохранение

Имитация протокола шины:



## 

## 

## **2.2. Программные средства наблюдения. Примеры. Средства измерения времени в ОС.**

(лекция 2)

Программный наблюдатель – это специализированная программа (или комплекс программ), встроенная в измеряемую систему. Наблюдатель выступает посредником между теми компонентами системы, за которыми он наблюдает. Программный наблюдатель всегда изменяет измеряемую систему.

Уровень операционной системы: Трассировка системных вызовов

% strace /bin/cat

strace используется для мониторинга взаимодействий между процессами и ядра Linux, которые включают в себя системные вызовы, доставку сигналов и изменения в состоянии процесса.

Трассировка библиотечных функций

* % ltrace /bin/cat 2> ltrace.out

ltrace - регистрирует вызовы динамических библиотек с целью отладки.

Программа ltrace предназначена для отладки динамически собранных программ. Отлаживаемый код запускается под управлением ltrace, при этом вызовы динамических библиотек, а также получаемые процессом сигналы перехватываются и регистрируются. Возможна также регистрация системных вызовов со стороны отлаживаемой программы.

Для отладки программы её не нужно перекомпилировать, поэтому возможно использование ltrace с программами, исходный текст которых не доступен.

Время в операционной системе:

* #include <time.h>

struct timespec {

time\_t tv\_sec;

int tv\_usec; // нано

};

struct tm {}

time()

sleep()

* /usr/bin/time myfile
  + real
  + user
  + sys
* Команда rdtsc в x86 – ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Read Time Stamp Counter*) — [ассемблерная](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80) инструкция для платформы [x86](https://ru.wikipedia.org/wiki/X86), читающая счётчик TSC (Time Stamp Counter) и возвращающая в [регистрах](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%D0%B0) EDX:EAX 64-битное количество [тактов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%BA%D1%82_%28%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%29) с момента последнего сброса процессора

Профилирование — сбор характеристик работы [программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), таких как время выполнения отдельных фрагментов (обычно подпрограмм), число верно [предсказанных условных переходов](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D0%BA%D0%B0%D0%B7%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_%D0%BF%D0%B5%D1%80%D0%B5%D1%85%D0%BE%D0%B4%D0%BE%D0%B2), число [кэш](https://ru.wikipedia.org/wiki/%D0%9A%D1%8D%D1%88)-промахов и т. д.

Профилирование gprof

% gcc –pg a.c

% a.out *(создаётся gmon.out)*

% gprof

Выборка 100 раз в секунду

* Число вызовов функций
* Время, проведённое в каждой функции
* Граф вызовов

Valgrind

Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования.

* % valgrind myfile
* % valgrind -- tool=cachegrind myfile
* Проверка некоторых условий корректности работы с указателями и malloc/free
* Анализ работы гипотетической кэш-памяти
* Построение графа вызовов функций
* Возможность добавления инструментов
* www.valgrind.org

## **2.3. Основные возможности средств отображения и анализа трасс (как с аппаратных, так и с программных наблюдателей).**

**Основные возможности средств отображения и анализа трасс**

## 

## 

## 

## **2.4. Понятие производительности ИС. Роль наблюдения в анализе производительности и отладке программы.**

Производительность ИС характеризуют следующие параметры:

* Загруженность ИС (BusBusy, Busldle, DataPhase, WaitState)
* Пропуская способность — метрическая характеристика, показывающая [соотношение](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BE%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5) предельного количества проходящих единиц ([информации](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F), предметов, [объёма](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D1%91%D0%BC)) в единицу времени через канал, систему, узел
  + Анализироваться могут все обмены или в зависимости от приоритета
  + По диапозонам адресов
* Запаздывание. Могут анализироваться следующие временные интервалы (для шины):
  + Master от выставления адреса до первого слова данных
  + Target от запроса первого слова данных до выставления
  + Arbitration от запроса до разрешения

(Master - инициатор)

(Target - подчиненное устройство)

Отладка программы (или управляющей системы) может осуществляться посредством

* Наблюдения за поведением программы
* Проверки соблюдения заданных свойств поведения
* Воздействием на систему

# **3. Основы имитационного моделирования**

## **3.1. Понятие модели. Основные виды моделей. Примеры. Основные варианты применения моделей.**

(лекция 3)

Модель – это объект, заменяющий исходный объект в ходе достижения заданных целей и при заданных предположениях.

Важные примеры цели моделирования:

* понимание;
* исследование (проектирование);
* замещение исходного объекта в составе системы;

Любая модель обладает следующими свойствами:

* конечностью: модель отображает оригинал лишь в конечном числе его отношений;
* упрощенностью: модель отображает только существенные стороны объекта;
* приблизительностью: действительность отображается моделью грубо или приблизительно;
* адекватностью: модель успешно описывает моделируемую систему;
* информативностью: модель должна содержать достаточную информацию о системе в рамках гипотез, принятых при построении модели.

Виды моделей:

По степени абстрагирования от оригинала модели могут быть разделены на материальные (физические) и идеальные.

* Физические
  + Натурные – это реальные исследуемые системы, которые являются макетами и опытными образцами. (эксперименты с реальным объектом или его частью);
  + полунатурные – это совокупность натурных и математических моделей (часть объекта заменена моделью);
  + масштабные – это системы той же физической природы, что и оригинал, но отличающиеся от него размерами (глобус, макеты зданий, макеты в аэро(гидро)динамической трубе)
  + аналоговые – основано на аналогии процессов и явлений, имеющих различную физическую природу, но одинаково описываемых формально (одними и теми же математическими уравнениями, логическими схемами и т. п.), использование иных физических явлений с теми же законами (Аналоговые выч машины: http://habrahabr.ru/company/intel/blog/85645/)

Идеальное моделирование носит теоретический характер.

Различают два типа идеального моделирования: интуитивное и знаковое.

* Под интуитивным будем понимать моделирование, основанное на интуитивном представлении об объекте исследования, не поддающемся формализации либо не нуждающемся в ней. В этом смысле, например, жизненный опыт каждого человека может считаться его интуитивной моделью окружающего мира.
* Знаковым называется моделирование, использующее в качестве моделей знаковые преобразования различного вида: схемы, графики, чертежи, формулы, наборы символов и т. д., включающие совокупность законов, по которым можно оперировать с выбранными знаковыми элементами.
  + лингвистические – если представлены некоторым лингвистическим объектом, формализованной языковой системой или структурой. (правила, кодексы …)
  + графические – если представимы геометрическими образами и объектами (схемы, чертежи)
  + математические
* Математические - по свойствам объекта
  + структурные – если представимы структурой данных или структурами данных и отношениями между ними
  + функциональные – если представимы в виде системы функциональных соотношений.
* Математические функциональные:
  + Аналитические математические модели представляют собой явные математические выражения выходных параметров как функций от параметров входных и внутренних и имеют единственные решения при любых начальных условиях.
  + Модель будет численной, если она имеет решения при конкретных начальных условиях (дифференциальные, интегральные уравнения).
  + Модель алгоритмическая, – если она описана некоторым алгоритмом или комплексом алгоритмов, определяющим ее функционирование и развитие.
  + Модель имитационная, – если она предназначена для испытания или изучения возможных путей развития и поведения объекта путем варьирования некоторых или всех параметров модели.

## **3.2. Понятие имитационной модели. Сравнение по возможностям с другими видами моделей. Способы продвижения времени в имитационных моделях.**

(лекция 3)

* Алгоритмические ММ выражают связи выходных параметров с параметрами внутренними и внешними в форме алгоритма
* Имитационная ММ - это алгоритмическая модель, отражающая поведение исследуемого объекта во времени при задании внешних воздействий на объект.

(методические рекомендации Минобрнауки РФ; [Замятина 2005])

Имитационная модель воспроизводит процесс функционирования системы во времени, причём имитируются элементарные явления, составляющие процесс, с сохранением их логической структуры и последовательности протекания во времени [Советов, Яковлев, с.34]

Простейшая имитационная модель:

* N шариков; k ящиков;
* кладём шарики по одному в ящики с номерами 1..k последовательно, затем возвращаемся к ящику 1
* Сколько шариков будет в каждом ящике?

int box[k]; // вектор состояния;

// начальное состояние

for( int i=0; i<k; i++ )box[i]=0;

int time=0; // время

int curr\_box=0;

// цикл моделирования

while( time < N )

{ box[curr\_box]++; // кладём

curr\_box++;

if( curr\_box >= k )curr\_box=0; // след. ящик

time++; // след. момент времени

}

ИМ по способам продвижения времени:

* С постоянным шагом
  + Непрерывные модели
  + Потактовые модели

При продвижении времени с постоянным шагом t часы модельного времени продвигаются точно на t единиц времени для какого-либо соответствующего выбора значения t. После каждого обновления часов выполняется проверка, чтобы определить, произошли какие-либо события в течение предыдущего интервала времени t или нет. Если на этот интервал запланированы одно или несколько событий, считается, что данные события происходят в конце интервала, после чего состояние системы и статистические счетчики соответствующим образом обновляются.

* От события к событию
  + Дискретно-событийные модели

При использовании продвижения времени от события к событию часы модельного времени в исходном состоянии устанавливаются в 0, и определяется время возникновения будущих событий. После этого часы модельного времени переходят на время возникновения ближайшего события, и в этот момент обновляются состояние системы, с учетом произошедшего события, а также сведения о времени возникновения будущих событий. Затем часы модельного времени продвигаются ко времени возникновения следующего нового ближайшего события, обновляется состояние системы и определяется время будущих событий и т. д. Процесс продвижения модельного времени от времени возникновения одного события ко времени возникновения другого продолжается до тех пор, пока не будет выполнено какое-либо условие останова, указанное заранее.

* Гибридные модели
  + Совместная работа компонентов разного рода
  + Переключение режимов «непрерывного» компонента

Особенности ИМ:

* По сравнению с аналитическими моделями:
  + универсальность применения (+);
  + результат для конкретного набора входных данных (-);
* По сравнению с «программами общего назначения»:
  + «ТЗ формируется по ходу дела…»
  + Поддержка понятий предметной области

## **3.3. Основные этапы создания имитационной модели. Сравнение с этапами создания программы «общего назначения»**

(лекция 3)

I. Анализ требований и проектирование.

1. Постановка и анализ задачи и цели моделирования.

2. Сбор и анализ исходной информации об объекте

моделирования.

3. Построение концептуальной модели.

4. Проверка достоверности концептуальной модели.

II. Разработка модели.

1. Выбор среды моделирования.

2. Составление логической модели.

3. Назначение свойств модулям модели.

4. Задание модельного времени.

5. Верификация модели.

III. Проведение эксперимента.

1. Запуск модели, прогон модели.

2. Варьирование параметров модели и сбор статистики.

3. Анализ результатов моделирования.

IV. Подведение итогов моделирования согласно поставленной

цели и задачи моделирования.

Концептуальная модель

* Как правило, знаковая (лингв. или графич. модель) различной степени формализованности
* Построение – процесс неформальный, интуитивный
* Определяет структуру моделируемой системы, алгоритмы функционирования компонентов, их состояния, порядок взаимодействия, и т.д.
* Представляет собой решение по абстракции и упрощению иследуемой системы
* «техническое задание» на программирование имитационной модели

**СРАВНЕНИЯ С ЭТАПАМИ СОЗДАНИЯ ПРОГРАММЫ «ОБЩЕГО НАЗНАЧЕНИЯ»**

**Может это подойдет за этапы?**

Обычно, разработка программного обеспечения делится на четыре больших этапа:

1. Этап проектирования программного комплекса. На этом этапе усилиями заказчика и компании исполнителя создается техническое задание, которое служит основой для разработки программ. Без этого этапа невозможна разработка программ, поэтому он требует времени. Обычно, на него уходит от 1 до 3 недель, в зависимости от сложности проекта и квалификации сотрудников. Кроме того, на этом этапе составляется смета и график работ по созданию программы.

2. Этап разработки программного комплекса. Собственно и является процессом разработки программного обеспечения. На этом этапе, осуществляется создание ПО согласно утвержденному ранее техническому заданию и смете. По окончанию этого этапа, заказчик получает диск с уже готовой программой, а так же всю документацию необходимую для работы с программой. Длительность этого этапа, а так же стоимость написания программы рассчитываются индивидуально, и зависят от сложности проекта и от особенностей указанных в техническом задании.

3. Этап внедрения программного обеспечения. Собственно, на этом этапе разработка программного обеспечения уже закончена, и начинается процесс поддержки пользователя. Обычно, в этот этап включают в себя установку нового ПО на компьютеры в организации, а так же обучение сотрудников организации использованию нового программного обеспечения.

4. И наконец, последний этап – информационная поддержка. В этот этап обычно включается гарантийное обслуживание ПО, выявление и устранение мелких ошибок, а так же внесение изменений, не влекущих за собой изменение программной структуры.

## 

## 

## **3.4. Подходы к описанию имитационной модели: событийный, процессно-ориентированный, агентно-ориентированный. Основные понятия, поддерживаемые языками и библиотеками классов имитационного моделирования**

(лекция 4)

Подходы к организации модели:

* Событийно-ориентированный
* Процессо-ориентированный
  + Транзакты
  + «собственно» процессы и нити (threads)
* Агентно-ориентированный   
  (развитие «собственно» процессного)

При событийном подходе система моделируется путем идентификации изменений, происходящих в ней в моменты свершения событий. Задача исследователя заключается в описании событий, которые могут изменить состояние системы, и определении логических взаимосвязей между ними. Имитация функционирования системы осуществляется путем выполнения упорядоченной во времени последовательности логически взаимосвязанных событий.

Много слайдов про события

Недостатки прямолинейного событийного подхода:

* Нет структуры
* Неудобства детализации модели
* Неудобства объединения моделей и построения иерархии

Для крупных моделей используется внутри библиотечных объектов или внутри среды прогона моделей

Интересный пример: DiskSim – настраиваемая модель дисковой системы

<http://www.pdl.cmu.edu/DiskSim/>

Усовершенствование схемы планирования событий:

Переход к ОО языку

* Компонентам системы ставим в соответствие объекты (процессы)
* Переменные состояния – располагаем внутри процессов
* События помещаем в календарь с указателем на процесс
* У процесса – собственный обработчик событий

Пример библиотеки: http://www.inf.usi.ch/carzaniga/ssim/

Многие имитационные модели содержат последовательности компонентов, которые возникают в них по определенной схеме, например очередь, в которой клиенты ожидают обслуживания. Логика возникновения компонентов по требуемой схеме может быть обобщена и задана в одном операторе. Имитационный язык затем транслирует такие операторы в соответствующую последовательность событий, происходящих с компонентами модели. Имитационные языки, включающие операторы для моделирования процесса прохождения элементов через систему, обычно называются процессно-ориентированными. Эти операторы определяют последовательность событий, которые автоматически выполняются имитационным языком, по мере того как элементы продвигаются через систему.

Процессо(транзактно)-ориентированный подход:

* Сеть узлов обслуживания (ресурсов)
* Запросы на обслуживание (транзакты)
* Очереди транзактов

Классический пример: GPSS

since 1961

* GPSS World ((С) 2007 Minuteman Software)
* Расширенный редактор GPSS World ((С) Элина Компьютерс)

Процессо-ориентированный подход:

* Процесс как компонент модели - контекст и поток управления;
* Функция продвижения модельного времени;
* Взаимодействие и синхронизация процессов
* Процессы модели отображаются на нити или процессы ОС;
* Привязка процессов к календарю обеспечивается средой прогона

Агентно-ориентированный подход:

С точки зрения практического применения агентное моделирование можно определить как метод имитационного моделирования, исследующий поведение децентрализованных агентов и то, как это поведение определяет поведение всей системы в целом. При разработке агентной модели, инженер вводит параметры агентов (это могут быть люди, компании, активы, проекты, транспортные средства, города, животные и т.д.), определяет их поведение, помещает их в некую окружающую среду, устанавливает возможные связи, после чего запускает моделирование. Индивидуальное поведение каждого агента образует глобальное поведение моделируемой системы.

Создание популяции агентов

Динамическое создание и удаление агентов

Среда

Непрерывное пространство

Дискретное пространство

ГИС пространство

Связи агентов и сети

Взаимодействие агентов

Синхронизация агентов

Сбор статистики по агентам

**Основные понятия, поддерживаемые языками и библиотеками классов имитационного моделирования**

Может это?

Языки моделирования позволяют описывать моделируемые системы в терминах, разработанных на базе основных понятий имитации. С их помощью можно организовать процесс общения заказчика и разработчика модели. Различают языки моделирования непрерывных и дискретных процессов.

*Преимущества* *языков имитационного моделирования* (ЯИМ) по сравнению с универсальными языками общего назначения (ЯОН) следующие: 1) язык моделирования содержит абстрактные конструкции, непосредственно отражающие понятия, в которых представлена формализованная модель, и близкие концептуальному уровню описания моделируемой системы. Это упрощает программирование имитатора, позволяет автоматизировать выявление многих ошибок в программах; 2) языки моделирования имеют эффективный встроенный механизм продвижения модельного времени (календарь событий, методы интегрирования и др.), средства разрешения временных узлов; 3) языки моделирования, как правило, содержат встроенные датчики случайных чисел, генераторы других типовых воздействий; 4) в языках моделирования автоматизирован сбор стандартной статистики и других результатов моделирования, имеются средства автоматизации выдачи этих результатов в табличной или графической форме; 5) языки моделирования имеют средства, упрощающие программирование имитационных экспериментов, в частности, автоматизирующие установку начального состояния и перезапуск модели. *Недостатки* языков имитационного моделирования: 1) используются только стандартные формы вывода результатов моделирования; 2) недостаточная распространенность языков моделирования, которые, как правило, не входят в штатное программное обеспечение операционных систем; 3) необходимость дополнительного обучения языкам моделирования и, как следствие, недостаток программистов, хорошо владеющих языками моделирования; 4) отсутствие гибкости и широких возможностей, присущих универсальным языкам программирования.

*Архитектуру* ЯИМ, т.е. концепцию взаимосвязей элементов языка как сложной системы, и технологию перехода от системы *S* к ее машинной модели *ММ* можно представить следующим образом:

1. объекты моделирования (системы *S*) описываются (отображаются в языке) с помощью некоторых атрибутов языка;
2. атрибуты взаимодействуют с процессами, адекватными реально протекающим явлениям в моделируемой системе *S*;
3. процессы требуют конкретных условий, определяющих логическую основу и последовательность взаимодействия этих процессов во времени;
4. условия влияют на события, имеющие место внутри объекта моделирования (системы *S*) и при взаимодействии с внешней средой *Е;*
5. события изменяют состояния модели системы *М* в пространстве и во времени.

Может быть, вот это (<http://bigor.bmstu.ru/?cnt/?doc=110_Simul/3015.mod/?cou=140_CADedu/CAD.cou>) релевантно:

При использовании языков, ориентированных на процессы, в составе СИМ выделяются элементарные части и ими могут быть **источники входных потоков заявок, устройства, накопители и узлы**.

**Источник входного потока заявок** представляет собой алгоритм, в соответствии с которым вычисляются моменты  появления заявок на выходе источника. Источники могут быть зависимыми и независимыми. В зависимых источниках моменты появления заявок связаны с наступлением определенных событий, например, с приходом другой заявки на вход некоторого устройства. Типичным независимым источником является алгоритм выработки значений  случайной величины с заданным законом распределения.

**Устройства** в имитационной модели представлены алгоритмами выработки значений интервалов (длительностей) обслуживания. Чаще всего это алгоритмы генерации значений случайных величин с заданным законом распределения. Но могут быть устройства с детерминированным временем обслуживания или временем, определяемым событиями в других частях СИМ. Модель устройства отображает также заданную дисциплину обслуживания, поскольку в модель входит алгоритм, управляющий очередями на входах устройства.

**Накопители** моделируются алгоритмами определения объемов памяти, занимаемых заявками, приходящими на вход накопителя. Обычно объем памяти, занимаемый заявкой, вычисляется как значение случайной величины, закон и (или) числовые характеристики распределения может зависеть от типа заявки.

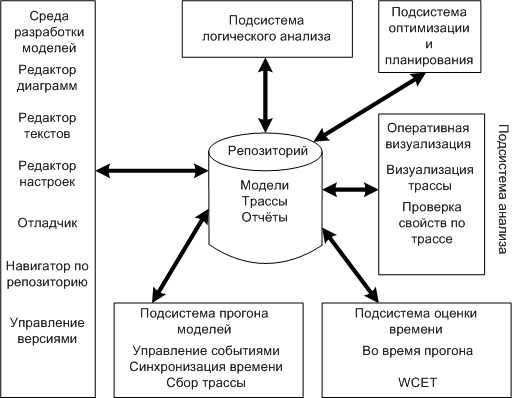
**Узлы** выполняют связующие, управляющие и вспомогательные функции в имитационной модели, например, для выбора направлений движения заявок в СИМ, изменения их параметров и приоритета, разделения заявок на части, их объединения и т.п.

Обычно каждому типу элементарной модели, за исключением лишь некоторых узлов, в программной системе соответствует определенная процедура (подпрограмма). Тогда СИМ можно представить как алгоритм, состоящий из упорядоченных обращений к этим процедурам, отражающим поведение моделируемой системы.

В процессе моделирования происходят изменения модельного времени, которое чаще всего принимается дискретным, измеряемым в тактах. Время изменяется после того, как закончена имитация очередной группы событий, относящихся к текущему моменту времени . Имитация сопровождается накоплением в отдельном файле статистики таких данных, как количества заявок, вышедших из системы обслуженными и необслуженными, суммарное время занятого состояния для каждого из устройств, средние длины очередей и т.п. Имитация заканчивается, когда текущее время превысит заданный отрезок времени или когда входные источники выработают заданное число заявок. После этого производят обработку накопленных в файле статистики данных, что позволяет получить значения требуемых выходных параметров.

## **3.5. Обобщённая архитектура системы имитационного моделирования. Краткое описание функций её компонентов**

(лекция 4)



**Краткое описание функций её компонентов**

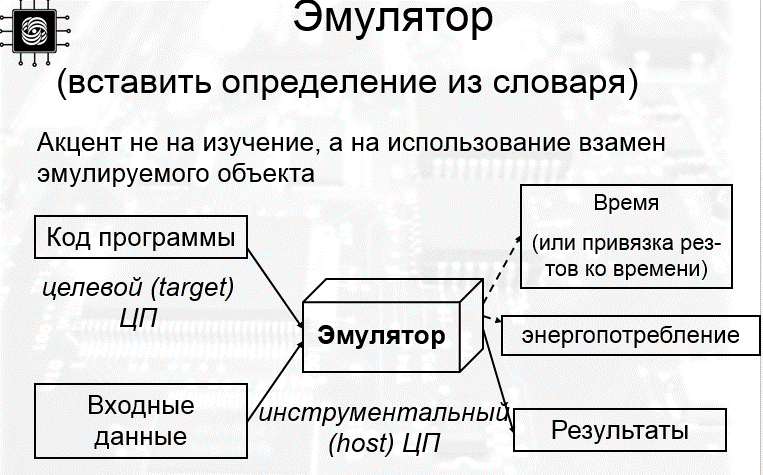
# 

# 

# **4. Важные классы средств имитационного моделирования, специфичные для ИС**

## **4.1. Эмуляторы процессоров. Назначение. Классификация по точности.**

(лекция 7)



**Эмуля́ция** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *emulation*) в [вычислительной технике](http://техника) — комплекс программных, аппаратных средств или их сочетание, предназначенное для копирования (или *эмулирования*) функций одной вычислительной системы (*гостя*) на другой, отличной от первой, вычислительной системе (*хосте*) таким образом, чтобы эмулированное поведение как можно ближе соответствовало поведению оригинальной системы (*гостя*). Целью является максимально точное воспроизведение поведения

Характеристики эмулятора:

* Архитектуры host и target
  + Для одной целевой архитектуры
  + Для различных архитектур (retargetable)
* Точность и детальность
  + Без учёта времени выполнения
  + С потактовой точностью времени (cycle clock accurate)
  + С потактовой точностью (cycle accurate)
* Скорость работы

Интерпретирующий эмулятор:

while (!stop())

{

декодировать;

выполнить;

обновить\_состояние;

выбрать следующую команду;

}

Эмуляторы с двоичной трансляцией:

Статическая трансляция

* в C/C++, затем в машинный код для хоста;
* Напрямую в команды хоста;
* В команды абстрактной машины.
* Ускорение – до сотен раз

Динамическая двоичная трансляция:

* Во время выполнения программы
* Кэш для результатов трансляции
* Единица трансляции – как правило, линейный участок (участок программы, в рамках которого не передается управление)
* Вопросы оптимизации, в т.ч. отображение виртуальных регистров на регистрыхост-машины
* (рекорд?) 1,6 раз медленнее исполнения на натурной цели [2]

Там еще дофига всего, непонятно куда надо говорить…

Примеры:

1. bochs (<http://bochs.sourceforge.net/>) - x86 эмулятор.
2. QEMU (<http://xgu.ru/wiki/QEMU>, http://www.ibm.com/developerworks/ru/library/l-qemu/)

Интерпретатор — простейшая форма эмуляции CPU. Это — программа, которая следует за потоком выполнения программы и для каждой встречающейся машинной инструкции выполняет операции, семантически эквивалентные оригинальным инструкциям, на процессоре хоста. Это возможно назначением переменных для каждого регистра и флага эмулируемого CPU. Логика работы эмулированного CPU может быть с большей или меньшей сложностью реализована в алгоритме программы, создавая программную реализацию, которая в большей или меньшей степени отражает оригинальное оборудование.

Интерпретаторы весьма популярны для моделирования вычислительных машин, так как их реализация намного проще, нежели выигрывающих в производительности других решениях, так как скорость их работы зачастую достаточна для эмуляции компьютеров, которые в буквальном смысле на десяток лет старше современных машин.

Однако применение интерпретации с присущим ей снижением производительности может быть проблемой, когда требуется эмулировать компьютер с процессором, производительность которого соответствует порядку величины производительности процессора хоста. До недавнего времени эмуляция в подобных случаях многими считалась бесполезной.

Преодолеть эти ограничения помог прогресс в технике динамической компиляции. Простая трансляция эмулированного программного кода в код, выполнимый на архитектуре хоста, невозможна априори по нескольким причинам:

* код может являться самомодифицирующимся, даже если модификация происходит только во время загрузки кода эмулированной операционной системой (например, с диска);
* может не существовать надежного способа разделения данных (не транслируются) и исполняемого кода.

Чтобы обойти эти проблемы применяют разнообразные приемы, включая известную «компиляцию на лету». Компилятор ожидает, пока поток управления процессором не перейдет в область, содержащую нетранслированный код. Только тогда («на лету») происходит трансляция блока кода в код, который может быть выполнен. Обработанный код помещается в кэш кода, при этом оригинальный код не подвергается изменению. В таком случае даже блоки данных подвергнутся бессмысленной трансляции компилятором, единственным эффектом чего будет увеличение времени работы транслятора.

## **4.2. Способы быстрой эмуляции процессоров.**

(видимо тоже лекция 7, но что именно??)

Полностью программные средства

bochs ([http://bochs.sourceforge.net](http://bochs.sourceforge.net/)) - можно эмуляровать x86 процессор на любых хостах (под который можно собрать bochs). Можно было запускать винду (win95, win98) на древних КПК (на раньше работали на XScale или mips процессорах). Производительность низкая, но win31 пользоваться было можно (что реально хороших результат, т.к процессор эмулируется полностью программно)

QEMU - быстрая эмуляция процессоров на современном железе. Динамичекая трансляция адресов и использованое аппаратных возможностей CPU. Для виртуализации устройств может использовать VT-d (виртуализация ввода-вывода), VT-x (виртуализация страниц).

VMWare - эмуляция некоторого функционала процессора (счетчики инструкций, ). Эмуляция устройств. Скорость работы за счет аппаратной виртуализации не отличается от скорость работы на хост машине.

IBM System/Z (<http://www-03.ibm.com/systems/ru/z/hardware/>). Может эмулировать старые процессоры IBM для работы старых программ. Быстро за счет аппаратной эмуляции. Низкая распространенность (самая дешевая конфигурация от 1миллиона USD). Используется во всяких фирмах, где простой может очень дорого обойтись (биржи, банки, управление ответственными системами и тд).

JIT компиляция

Java - создается некоторый байткод, который выполняется виртуальной машиной Java. Для каждого хоста используется своя виртуальная машина, использующая возможности конкретного процессора. Есть java для встраиваемых решений - смарткарты (банковские), Maxim integrated (<http://www.maximintegrated.com/en/products/ibutton/jibkit/javaringfaq.cfm>). Формфактор - ключ-таблетка (как от домофонов), при этом можно писать программу на java и она будет выполняться на этом устройстве.

Python - аналогично. Генерируется байткод для выполнения на хост машине.

LLVM - low level virtual machine. Компилятор (например, CLang) генерирует набор llvm инструкций, которые можно преобразовать в команды целевой машины. При динамической эмуляции можно во время выполнения осуществлять декомпиляцию конкретной архитектуры (брать из кэша уже дизассемблированного кода), генерировать по ним LLVM и генерировать инструкции хост машины. В данном случае будет происходить эмуляция почти без потери скорости работы (но нужно отдельно отслеживать самомодифицирующийся код, т.к из-за него может все сломаться).

## **4.3. Основные виды сверхбольших интегральных схем (СБИС). Уровни моделирования и проектирования аппаратных средств ИС.**

(лекция 8)

Основные виды БИС:

* Для заданной области применения (ASIC)
  + Широкого применения (Application specific standard products, ASSP) (микропроцессоры, модули памяти, контроллеры интерфейсов, видеодекодеры …) – крупные производители, большие тиражи, сторонние потребители;
  + Заказные (фильтры, коммутаторы …) – меньшие тиражи, возможность изготовить «для себя»

ASIC (аббревиатура от англ. application-specific integrated circuit, «интегральная схема специального назначения») — интегральная схема, специализированная для решения конкретной задачи. В отличие от интегральных схем общего назначения, специализированные интегральные схемы применяются в конкретном устройстве и выполняют строго ограниченные функции, характерные только для данного устройства; вследствие этого выполнение функций происходит быстрее и, в конечном счёте, дешевле. Примером ASIC может являться микросхема, разработанная исключительно для управления мобильным телефоном, микросхемы аппаратного кодирования/декодирования аудио- и видео-сигналов (сигнальные процессоры).

Микросхема ASIC имеет узкий круг применения, обусловленный жёстко предопределённым набором её функций.

Современные ASIC часто содержат 32-битный процессор, блоки памяти (как ПЗУ, так и ОЗУ) и другие крупные блоки. Такие ASIC часто называют система на кристалле (англ. System-on-a-Chip).

При разработке цифровых ASIC для описания их функциональности используют языки описания аппаратных устройств (HDL), такие как Verilog и VHDL.

* Вентильные матрицы (полузаказные)
  + часть слоёв металлизации делается по заказу
* Программируемые логические матрицы (FPGA)
  + Набор базовых ячеек
  + Переключатели связей между ячейками
  + Загрузка конфигурации по специальному интерфейсу

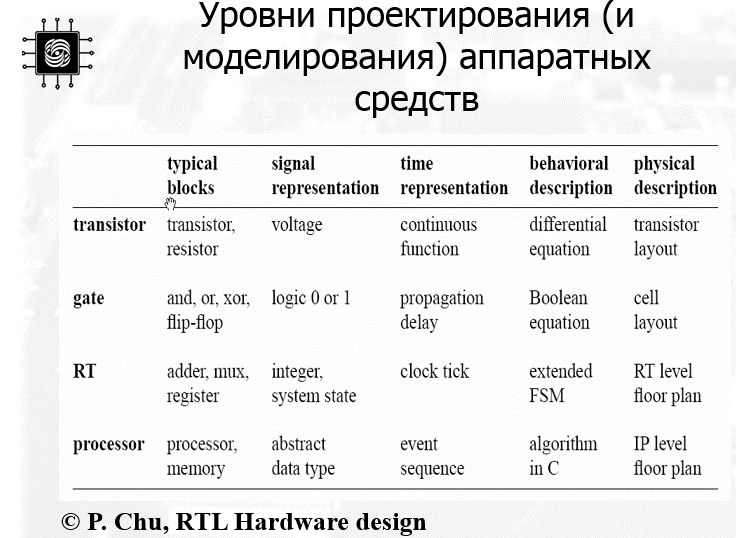
Программи́руемая по́льзователем ве́нтильная ма́трица (ППВМ, англ. Field-Programmable Gate Array, FPGA) — полупроводниковое устройство, которое может быть сконфигурировано производителем или разработчиком после изготовления; отсюда название: «программируемая пользователем». ППВМ программируются путём изменения логики работы принципиальной схемы, например, с помощью исходного кода на языке проектирования (типа VHDL), на котором можно описать эту логику работы микросхемы. ППВМ является одной из архитектурных разновидностей программируемых логических интегральных схем (ПЛИС).

ППВМ могут быть модифицированы практически в любой момент в процессе их использования. Они состоят из конфигурируемых логических блоков, подобных переключателям с множеством входов и одним выходом (логические вентили или gates). В цифровых схемах такие переключатели реализуют базовые двоичные операции AND, NAND, OR, NOR и XOR. В большинстве современных микропроцессоров функции логических блоков фиксированы и не могут модифицироваться. Принципиальное отличие ППВМ состоит в том, что и функции блоков, и конфигурация соединений между ними могут меняться с помощью специальных сигналов, посылаемых схеме. В некоторых специализированных интегральных схемах (ASIC) используются логические матрицы, аналогичные ППВМ по структуре, однако они конфигурируются один раз в процессе производства, в то время как ППВМ могут постоянно перепрограммироваться и менять топологию соединений в процессе использования. Однако, такая гибкость требует существенного увеличения количества транзисторов микросхемы.

CPLD (англ. Complex Programmable Logic Device) — программируемая логическая интегральная схема (ПЛИС) в диапазоне сложности между микросхемами PAL (англ.) (Programmable Array Logic) и FPGA (англ. Field-Programmable Gate Array), сочетающая их архитектурные решения. Основой CPLD является матрица макроячеек, в которой реализованы логические соединения вентилей или более сложные логические операции. Блоки макроячеек объединены программируемой коммутационной матрицей с последующим выходом на входы (выходы) схемы. Главным отличием между большими CPLD и малыми FPGA до недавних пор было наличие внутренней энергонезависимой конфигурационной памяти в CPLD. Это отличие становится уже не столь значимым, поскольку ряд последних моделей FPGA также включают такую внутреннюю память. Тем не менее, наличие такой внутренней энергонезависимой конфигурационной памяти, наряду с такой важной характеристой, как устойчивость показателей, делают CPLD незаменимыми для современных цифровых схем в качестве устройства для инициализации схемы, перед тем, как передать управление другим микросхемам, не обладающим такой способностью.

Уровни моделирования аппаратных средств ИС:

* Уровень электрических цепей (transistors level)
* Уровень вентилей (gate level)
* Уровень регистровых передач (register transfer level, RTL)
* Системный уровень (system level, processor level)



Уровень эл цепей:

Специализированные средства моделирования:

* PSPICE (Personal Simulation Program with Integrated Circuits Emphasis) от Cadence Design Systems
  + Задание схемы (текстовый язык или графический редактор)
  + Элементы схемы выбираются из библиотеки
  + Задаётся режим моделирования: анализ переходных процессы, расчёт установившегося режима, …
  + Выполняется моделирование
* Универсальные системы непрерывного моделирования: Matlab Simulink (Mathworks Corp.)

## **4.4. Моделирование на уровне RTL. Учёт специфики предметной области в языке Verilog.**

(лекция 8)

**Уровень регистровых передач** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Register transfer level, RTL*) — в разработке [интегральных микросхем](http://микросхема) — описание работы синхронной [цифровой схемы](https://ru.wikipedia.org/w/index.php?title=%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D1%8B%D0%B5_%D1%81%D1%85%D0%B5%D0%BC%D1%8B&action=edit&redlink=1). На данном уровне поведение схемы определяется в терминах потоков сигналов (или пересылок данных) между аппаратными регистрами и [логических операций](http://алгебра) над данными сигналами.[[1]](https://ru.wikipedia.org/wiki/%D3%F0%EE%E2%E5%ED%FC_%F0%E5%E3%E8%F1%F2%F0%EE%E2%FB%F5_%EF%E5%F0%E5%E4%E0%F7#cite_note-1)

Абстракция уровня регистровых передач используется в [языках описания аппаратуры](http://аппаратуры) (HDL), например в [Verilog](https://ru.wikipedia.org/wiki/Verilog) HDL и [VHDL](https://ru.wikipedia.org/wiki/VHDL) для создания высокоуровневых описаний схем, из которых могут быть получены низкоуровневые описания и, в конце концов, прошивка для [ПЛИС](https://ru.wikipedia.org/wiki/%D0%9F%D0%9B%D0%98%D0%A1) или маски для изготовления СБИС.

Verilog, Verilog HDL (англ. Verilog Hardware Description Language) — это язык описания аппаратуры, используемый для описания и моделирования электронных систем. Verilog HDL, не следует путать с VHDL(конкурирующий язык), наиболее часто используется в проектировании, верификации и реализации (например, в виде СБИС) аналоговых, цифровых и смешанных электронных систем на различных уровнях абстракции.

Разработчики Verilog сделали его синтаксис очень похожим на синтаксис языка C, что упрощает его освоение. Verilog имеет препроцессор, очень похожий на препроцессор языка C, и основные управляющие конструкции «if», «while» также подобны одноимённым конструкциям языка C. Соглашения по форматированию вывода также очень похожи (см. printf).

Следует отметить, что описание аппаратуры, написанное на языке Verilog (как и на других HDL-языках) принято называть программами, но в отличие от общепринятого понятия программы как последовательности инструкций, здесь программа задает структуру системы. Так же для языка Verilog не применим термин "выполнение программы".

VHDL (англ. VHSIC (Very high speed integrated circuits) Hardware Description Language) — язык описания аппаратуры интегральных схем. Язык проектирования VHDL является базовым языком при разработке аппаратуры современных вычислительных систем.

Two leading HDLs:

* **Verilog;** 
  + developed in 1984 by Gateway Design Automation
  + became an IEEE standard (1364) in 1995
* **VHDL**
  + Developed in 1981 by the Department of Defense
  + Became an IEEE standard (1076) in 1987

Учет специфики предметной области:

* Описание **структуры** системы
  + модули; входные и выходные порты; соединения
* Описание **поведения** системы
  + сигналы; время; события; операции; параллелизм

дофига слайдов еще непонятных про сам язык, надо ли?..

Описание на HDL: виды использования

* Документация на схему
* Исходные данные для средства трассировки соединений элементов (Структурное описание в заданном элементарном базисе);
* Моделирование работы схемы
* Синтез описания на уровне вентилей (При определённых ограничениях на подмножество языка);
* Логическая верификация (доказательство соблюдения свойств)

## **4.5. Моделирование на системном уровне. Учёт специфики предметной области в библиотеке SystemC. Понятие о стандарте TLM.**

(лекция 9)

SystemC — язык проектирования и верификации моделей системного уровня, реализованный в виде C++ библиотеки с открытым исходным кодом. Библиотека включает в себя ядро событийного моделирования, что позволяет получить исполняемую модель устройства. Язык применяется для построения транзакционных и поведенческих моделей, а также для высокоуровневого синтеза.

Язык SystemC использует ряд понятий, схожих с теми, которые имеют языки описания аппаратуры VHDL и Verilog: интерфейсы, процессы, сигналы, событийность, иерархия модулей. Стандарт SystemC не вносит ограничения на использование языка C++ при описании моделей систем.

Назначение SystemC:

* Моделирование функций системы, которые могут быть реализованы как программно, так и аппаратно
* Моделирование совместной работы аппаратных и программных компонентов
* Обмен информацией между разработчиками

Возможности:

* Модули Типы данных:
* Порты - битовые массивы
* Каналы - целые
* Интерфейсы - вещественные
* Процессы - с фиксированной
* Сигналы запятой
* События
* Таймеры

### Модули (module)

Модули - базовые строительные блоки в SystemC. Модель в SystemC обычно состоит из нескольких модулей, которые общаются через порты.

### Сигналы (signal)

Сигналы в SystemC являются эквивалентом посылки по проводу (wire).

### Порты (port)

Через порты происходит общение модуля с внешним миром (обычно с другими модулями).

### Процессы (process)

Процессы - главные вычислительные элементы. Процессы выполняются параллельно.

### Каналы (channel)

Через каналы в SystemC происходит общение элементов. Каналы могут быть простыми проводами или сложными соединительными механизмами наподобие очередей FIFO или шин.

Базовые каналы:

* signal
* buffer
* fifo
* mutex
* semaphore

### Интерфейсы (interface)

Порты используют интерфейсы для общения через каналы.

### События (events)

Должны быть описаны во время инициализации. Позволяют синхронизировать процессы.

Процессы

* методы модуля (methods)
  + списки чувствительности по портам
* нити (threads)
  + возможно параллельное выполнение разных нитей, ожидание условия
* тактируемые нити (clocked threads)
  + wait\_until()
  + wait() // next clock

TLM: **Transaction-level modeling** (TLM) is a high-level approach to modeling digital systems where details of communication among modules are separated from the details of the implementation of functional units or of the communication architecture. <http://en.wikipedia.org/wiki/Transaction-level_modeling>

* Представление системы в виде набора компонентов, связанных абстрактными каналами
* Конкретные каналы (шины, FIFO и т.д.) описываются как компоненты
* Стандартизован интерфейс компонент-канал
* Отделение функционирования от взаимодействия
* неблокирующий однонаправленный
* put() // действие
* can\_put() // проверка
* event ok\_to\_put() событие

Возможности:

* Взаимодействие компонентов разной степени детальности (через вспомогательные компоненты?)
* Детализация работы реального канала
* Взаимодействие программных (прикладных и системных) и аппаратных компонентов
* Взаимодействие компонентов, созданных в разных средах моделирования?!

Виртуальный прототип аппаратных средств:

Выполнение исполняемого кода программы на заданной конфигурации аппаратных средств

* эмулятор ЦП
* модели шин и периферии (TLM)
* среда выполнения моделей

# **5. Моделирование и проектирование ИС. Построение сложных имитационных моделей**

## **5.1. Взаимосвязь моделирования и проектирования ИС.**

(лекция 10)

**???????**

Этапы создания ИС: формирование требований, концептуальное проектирование, спецификация приложений, разработка моделей, интеграция и тестирование информационной системы. Методы программной инженерии в проектировании ИС.

Проектирование (от латинского projectus, что означает “брошенный вперед”) – это процесс составления описания, необходимого для создания в заданных условиях еще не существующего объекта по первичному описанию этого объекта путем его детализации, дополнения, расчетов и оптимизации.

Описание объекта может быть задано по-разному: в виде текста, алгоритма, программы, чертежа, таблицы или, что чаще всего, комбинировано в традиционно бумажном или электронном виде.

Таким образом, главной особенностью проектирования является работа с еще не существующим объектом. В этом отличие проектирования от моделирования, где объект не может не существовать.

Проектирование можно рассматривать с одной стороны как заключительную фазу исследований, а с другой как начальную фазу производства.

Проектирование ИС охватывает три основные области:

* проектирование объектов данных, которые будут реализованы в базе данных;
* проектирование программ, экранных форм, отчетов, которые будут обеспечивать выполнение запросов к данным;
* учет конкретной среды или технологии, а именно: топологии сети, конфигурации аппаратных средств, используемой архитектуры (файл-сервер или клиент-сервер), параллельной обработки, распределенной обработки данных и т.п.

Проектирование информационных систем всегда начинается с определения цели проекта. В общем виде цель проекта можно определить как решение ряда взаимосвязанных задач, включающих в себя обеспечение на момент запуска системы и в течение всего времени ее эксплуатации:

* требуемой функциональности системы и уровня ее адаптивности к изменяющимся условиям функционирования;
* требуемой пропускной способности системы;
* требуемого времени реакции системы на запрос;
* безотказной работы системы;
* необходимого уровня безопасности;
* простоты эксплуатации и поддержки системы.

Взаимодействие компонентов ИС:

* Аппаратные компоненты
  + Каналы
  + Интерфейсы и протоколы
* Программные компоненты
  + Средства межпроцессного взаимодействия, файлы
  + Промежуточное ПО (middleware)

Объединение имитационных моделей:

* Важный потребитель: моделирование военных действий
* объединение разнородных моделей
  + Подвижные объекты
  + Внешняя среда
  + Пункты управления
  + Связь, и т.д.
* Подключение унаследованных (legacy) моделей

Distributed Interactive Simulator (DIS)

Основная цель – объединение систем моделирования различных типов, распределённых географически

Основная идея – стандартизация интерфейсов

Возможные типы участников:

* Человек (virtual) [human-in-the-loop] — такие модели тебуют участия человека. Пример — симулятор полета, который требует участия пилота. См. https://en.wikipedia.org/wiki/Human-in-the-loop
* Оборудование (live) [hardware-in-the-loop] — в моделировании участвует само тестируемое оборудование, которому на вход подаются модельные сигналы. Например, при тестировании радара имитируются объекты, которые он должен регистрировать. См. https://en.wikipedia.org/wiki/Hardware-in-the-loop\_simulation
* Модель (constructive) [as-fast-as-possible]

Принципы DIS

* Независимость узлов
  + Участник не знает получателей данных
  + Независимое продвижение времени
* Узлы передают состояние частной модели
  + передаются только изменения в состоянии
  + keep-alive (heart-beat) сообщения
* Использование алгоритмов приближённого вычисления следующего состояния (счисление, dead reckoning)

Dead reckoning

Вместо постоянной пересылки состояний, участники предсказывают их изменения. Для этого строится локальная модель dead reckoning model (DRM).

Улучшения алгоритма:

* Компенсация времени

[учитывает время передачи данных]

* Сглаживание

[плавное уменьшение ошибки]

## **5.2. Понятие совместной разработки программной и аппаратной части ИС. Достоинства по сравнению с традиционным процессом разработки. Виртуальный прототип.**

(лекция 10???)

**Понятие совместной разработки программной и аппаратной части ИС. Достоинства по сравнению с традиционным процессом разработки.**

Создается прототип микросхемы на Verilog/VHDL. Проводится испытание в среде моделирования. Из бесплатных - Icarus Verilog (<http://iverilog.icarus.com/>). Из нормальных (для FPGA) <http://iverilog.icarus.com/>. Дальше трассировка полученной микросхемы послойно (современные состоят из десяткой слоев металлизации). Могут использоваться библиотеки готовых шаблонов, для которых уже проверена внутренняя электромагнитная совместимость. Специальной средой проверяется совместимость полученной схемы (электромагнитная) на основе физической модели. Для современных видеокарт (7млрд транзисторов) это крайне важно, т.к вручную уже невозможнно трассировать схему, а проверить, что из-за высоких токов не будет наводок на соседние элементы нужно. Для этого строится точная физическая модель и численно решается задача.

Дополнительно может проверяться корректность построенной микросхемы (набор физических транзисторов) и ее логической модели. Для этого опять строится модель в Verilog/VHDL и решается задача эквивалентности схем. Пример решения - объединение через XOR всех выводов микросхемы и ее модели, полученные результаты объединяются логическим ИЛИ. Дальше - решается задача SAT (булевая выполнимость). Существуют быстрые методы ее решения, так что на нормальных рабочих станциях эта задача разрешима за приемлемое время.

Собственно виртуальный прототип - набор иструкций на Verilog/VHDL, исполняемые на эмуляторе схем. Дополнительно его можно загрузить на FPGA и проверить работу на физическом устройстве.

## **5.3. Основные понятия стандарта HLA взаимодействия имитационных моделей**

(лекция 10)

HLA (High Level Architecture) – высокоуровневая архитектура для распределенного моделирования.

Архитектура и стандарты HLA ориентированы на создания распределенных динамических моделей, для которых характерно взаимодействие групп однотипных объектов, обменивающихся информацией о своем текущем состоянии. Большое внимание при этом уделяется фактору времени.

Основная идея HLA состоит в том, чтобы отделить специфические функциональные возможности моделирования от общих. Для поддержки последних разработана универсальная инфраструктура RTI (Run-Time Infrastructure). RTI, подобно распределенной операционной системе, обеспечивает функциональность, необходимую для взаимодействия объектов внутри распределенной модели.

Модель в HLA рассматривается как набор моделей разного уровня агрегирования. На нижнем уровне расположены модели компонентов системы - объектов. Объекты в HLA - совокупности, моделирующие реальный мир. Их взаимодействия отражают взаимодействия между реальными объектами.

В отдельные модели выделено управление объектами или группами объектов. Это могут быть математические или иные описания, имитируемые программными средствами. В HLA эти имитационные модели (симуляторы) имеют специальное название - **федераты**. Кроме симуляторов к федератам относятся также некоторые служебные программные компоненты, например, занятые сбором информации, а также пассивные наблюдатели.

Совокупность всех участвующих в моделировании федератов, сотрудничающих в интересах решения некоторой конкретной задачи, называется **федерацией**.

Основная цель HLA – поддержка информационных обменов в рамках федерации между сотрудничающими федератами.

В HLA информация, которой обмениваются федераты, бывает двух видов: состояние объекта и интеракция.

Каждый объект в любой момент времени характеризуется своим состоянием, которое определяется набором текущих значений его атрибутов. Федерат, управляющий данным объектом (точнее, атрибутом объекта), может изменить его состояние, изменив значение атрибута. Через сервис RTI этот федерат передает затем новое значение атрибута другим федератам. В этом случае говорят, что федерат обновляет атрибут. Про федерата, получающего новое значение атрибута, говорят, что он отображает атрибут.

Интеракции, в отличие от состояний объектов, не поддерживаются в системе постоянно, а имеют мгновенную природу. Интеракция представляет собой действие, производимое объектом данного федерата и влекущее за собой возможное изменение состояния другого объекта, управляемого другим федератом. Характерный пример – выстрел (действие) из орудия (объект, производящий действие) по цели (объект, на состоянии которого может отражаться действие). В HLA принято говорить, что федераты посылают и принимают интеракции.

Интеракции описываются специальными данными – параметрами. Они используются принимающими федератами для вычисления изменений значений атрибутов зависящего объекта.

Все обмены данными между федератами происходят через RTI. Механизм обмена реализован в виде подписки. Федерат, заинтересованный в получении определенных атрибутов и взаимодействий, должен подписаться на них через RTI. При этом, в целях снижения загрузки каналов связи, RTI отслеживает только изменения данных.

В результате, интеракции передаются всегда, а атрибуты – только, когда меняются их значения. Если атрибут, на который произведена подписка, не меняет значение, то подписавшийся объект использует старое его значение. Когда изменение произошло, оно передается как событие через RTI

Кроме механизма подписки, RTI предоставляет федератам и другие сервисы, например, подключения и выхода из состава федерации, изменения атрибутов, передачи прав владения атрибутами и т.д. Каждый из этих сервисов имеет четко прописанный интерфейс, в результате чего языковые и платформенные особенности реализации федератов становятся несущественными, то есть в рамках HLA обеспечивается интероперабельность федератов и RTI.

Еще одной особенностью HLA является наличие стандартной модели описания объектов, называемой эталонной объектной моделью **OMT** (Object Model Template), которая вводит шаблон описания объектов, федератов и федераций.

Эталонная объектная модель - существенный компонент HLA, поскольку она:

* предоставляет механизм для определения обмена данными и общей координации среди членов федерации;
* обеспечивает общий, стандартизированный механизм для описания возможностей потенциальных участников федерации;
* облегчает разработку и применение общих инструментальных средств для HLA моделей объектов.

Основные понятия HLA

Федерат – участник моделирования.

Федерация – совокупность федератов.

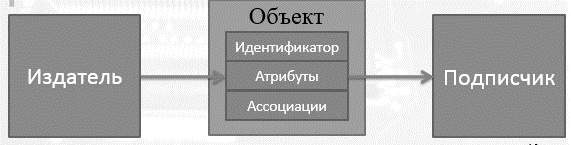
Объект – любой параметр, видимый сразу нескольким федератам в составе федерации.

Издатель – федерат владеющий атрибутом объекта.

Отражение – вызов федерата инфраструктурой RTI при изменении значения атрибута.

Подписчик – федерат, желающий получать отражения.

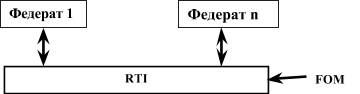
Взаимодействие (interaction) – сообщение



Компоненты HLA

* Правила [определяет принципы стандарта HLA]
* Object Model Template (OMT) [определяет формат описания FOM]
* Run-Time Infrastructure (RTI) [набор служб, доступных участникам]

Архитектура HLA



Правила (для федерации)

1. Федерации должны документировать Federation Object Model (FOM) в соответствии с OMT
2. Все представления объектов должны храниться в федератах [RTI не хранит состояния ИМ]
3. Обмен данными между федератами происходит через RTI [правильность семантики - ответственность федерата]
4. Федераты взаимодействуют с RTI только через стандартные интерфейсы
5. У атрибута может быть только один владелец в конкретный момент времени

Правила (для федератов)

1. Федераты должны документировать Simulation Object Model (SOM) в соответствии с OMT;
2. Федераты управляют продвижением локального времени;

В соответствии с SOM федераты могут:

1. Модифицировать и/или отражать любые атрибуты объектов, и посылать и/или принимать взаимодействия;
2. Передавать и/или принимать права на использование атрибутов динамически;
3. Изменять условия, при которых необходимо обновить значения атрибутов объектов.

Объекты:

* Объект – хранилище для атрибутов [не содержит методов]
* Атрибуты объекта распределены [атрибуты могут храниться не локально]
* Наследование [одиночное наследование]
* Флаги [publish/subscribe]

Свойства атрибутов:

* Класс объекта
* Имя
* Тип
* Мощность(массивы)
* Единицы измерения
* Разрешение
* Точность
* Условия точности
* Тип обновления
* Условие обновления
* Допускает прием и передачу владения
* Допускает изменение и отражение

Таблицы HLA OMT

1. Таблица идентификации объектной модели [информация о FOM/SOM]
2. Таблица структуры классов объектов [иерархии классов]
3. Таблица структуры классов взаимодействия [иерархии классов]
4. Таблица атрибутов объектов [типы и характеристики]
5. Таблица параметров взаимодействия [типы и характеристики]
6. Таблица пространственной маршрутизации [области распределения данных]
7. Словарь FOM/SOM [определяет термины, используемые в других таблицах]

Службы RTI

1. Управление федерацией [запуск, остановка, приостановка, возобновление…]
2. Управление декларациями [объявление и подписка на классы разделяемых объектов]
3. Управление объектами [создание и удаление объектов и взаимодействий]
4. Управление правами доступа (владением) [передача владения атрибутом между федератами]
5. Управление временем [продвижение модельного времени и синхронизация]
6. Управление распределением данных [фильтрация обновлений и взаимодействий по пространству маршрутизации и регионам]

Управление декларациями:

* *Object Publication*
* *Interaction Publication*
* *Object Subscription*
* *Interaction Subscription*

Управление объектами:

* 8.1 REGISTERING, DISCOVERING, AND DELETING OBJECT INSTANCES
* 8.2 UPDATING AND REFLECTING OBJECT ATTRIBUTES
* 8.5 EXCHANGING INTERACTIONS

Можно явно запросить обновление атрибута

Управление временем:

* У каждого федерата – свои локальные часы;
* Сообщению присваивается временная метка отправителя (сообщение == update || interaction)
* (один из вариантов настройки): сообщения доставляются в порядке возрастания временных меток

Управление временем (2):

Запроcы:

* queryFederateTime()
* timeAdvanceRequest( t )
* nextEventRequest( t )
* Ответ (callback)
* timeAdvanceGrant() – либо до t, либо до времени следующего события

# **6. Элементы статистических методов для моделирования и анализа производительности**

## **6.1. Понятие системы массового обслуживания, её основные элементы, характеристики производительности.**

(лекция 6)

Система массового обслуживания (СМО) — система, которая производит обслуживание поступающих в неё требований. Обслуживание требований в СМО производится обслуживающими приборами. Классическая СМО содержит от одного до бесконечного числа приборов. В зависимости от наличия возможности ожидания поступающими требованиями начала обслуживания СМО подразделяются на

1. системы с потерями, в которых требования, не нашедшие в момент поступления ни одного свободного прибора, теряются;
2. системы с ожиданием, в которых имеется накопитель бесконечной ёмкости для буферизации поступивших требований, при этом ожидающие требования образуют очередь;
3. системы с накопителем конечной ёмкости (ожиданием и ограничениями), в которых длина очереди не может превышать ёмкости накопителя; при этом требование, поступающее в переполненную СМО (отсутствуют свободные места для ожидания), теряется.

## Основные понятия СМО:

Требование (заявка) — запрос на обслуживание.

Входящий поток требований — совокупность требований, поступающих в СМО.

Время обслуживания — период времени, в течение которого обслуживается требование.

Математическая модель СМО — это совокупность математических выражений, описывающих входящий поток требований, процесс обслуживания и их взаимосвязь.

Системы массового обсуживания:

(или системы с очередями – queuing systems)

* Часто применяемые на практике модели
* Аналитическое и имитационное моделирование

Системы с очередями: основные элементы

* Входящий поток заявок
* Прибор(ы) обслуживания; время обслуживания
* Очередь заявок; длина; дисциплина обслуживания

A|B|s|q (пример: M|M|1|∞) ([1], с.30) [2], c. 14

* A – закон распределения вх. заявок
* B – закон распред. времени обслуживания
* s – число обслуживающих приборов
* q – максимальная длина очереди

A|B|s|q — это Kendall’s notation (http://en.wikipedia.org/wiki/Kendall%27s\_notation) . Буква “M” означает пуассоновский процесс (поток), в котором время между заявками распределено экспоненциально.

Пример: веб-сервер

Характеристики производительности:

* Средняя длина очереди
* Среднее время пребывания заявки в системе (или в очереди)
* Характеристики выходного потока (обслуженных заявок или отказов в обслуживании)

## **6.2. Система М/М/1. Пуассоновский поток. Средняя длина очереди. Формула Литтла.**

(лекция 6)

Входящий поток:

* Zk – интервал между событиями (заявками)
* λ(t) – количество событий к моменту t

Формула Литтла (связь между хар-ками производительности):

L = aV

L – среднее число заявок в системе

a – интенсивность поступления заявок

V – среднее время пребывания заявки в системе

N = aW

N – средняя длина очереди

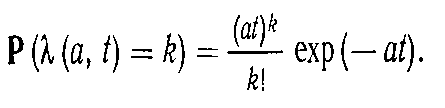
W – среднее время пребывания в очереди

Пуассоновский поток:

Пуассо́на пото́к — ординарный поток однородных событий, для которого число событий в интервале А не зависит от чисел событий в любых интервалах, не пересекающихся с А, и подчиняется распределению Пуассона. В теории случайных процессов описывает количество наступивших случайных событий, происходящих с постоянной интенсивностью.

P(z<t) = 1-exp(-at), интервалы независимы

λ(a,t): среднее at => a – интенсивность

 Распределение Пуассона

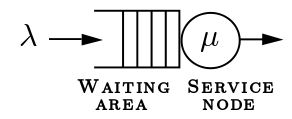
λ(a1,t)+λ(a2,t) ~ λ(a1+a2,t)

просеивание λ(a,t) с вероятностью z ~ λ(za,t)

Сведение к марковским процессам:

* Состояние – число заявок в системе
* Диаграмма переходов состояний
* Дифф. ур-я для состояний
* Условия наличия предельного распределения вероятностей состояний
* Переход к алгебраическим ур-ям для предельных вероятностей для состояний.
* Расчёт характеристик системы

М/М/1



λ - интенсивность поступления заявок, μ - интенсивностью обслуживания

ρ = λ/μ

Вероятность, что в системе k заявок:

P(k) = (1- ρ) ρk

Среднее число заявок в системе: ρ/(1- ρ)

Средняя длина очереди: ρ2/(1- ρ)

Загрузка обслуживающего прибора ρ

Сведение к марковским процессам

* Цепь Маркова с непрерывным временем

М/М/1/k

Согласно [2]:

P0 = (1- ρ)(1- ρK+1) или 1/(K+1)

Загрузка прибора Us = 1- P0

Среднее число заявок

ρ(1-(K+1)ρK+KρK+1)/((1- ρ)(1- ρK+1))

K/2, если ρ=1

При многократных экспериментах – не переинициализировать датчик сл.в.!

Для оценки установившегося режима – отбрасывание начальных наблюдений

см. [3], п. 9.5

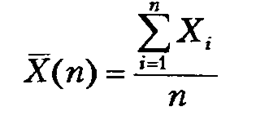
1. Матвеев В., Ушаков. Системы массового обслуживания // М.: Изд-во МГУ. – 1984. – 240с.

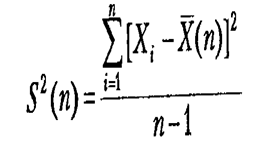
2. Dr. János Sztrik. Basic Queueing Theory. University of Debrecen, Faculty of Informatics. // [Электронный ресурс] http://*irh.inf.unideb.hu/~jsztrik/education/16/SOR\_Main\_Angol.pdf*

3. [Лоу, Кельтон]

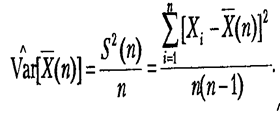
## **6.3. Обработка результатов эксперимента: оценка необходимого числа испытаний.**

(лекция 5)





оценка мат. ожидания





Оценка дисперсии

Оценка дисперсии оценки мат. ожидания

=======

Тут надо несмещенные оценки брать (если матожидание известно, то можно пользоваться обычными выборочными оценками. Если матожидание вычисляется по выборке, то нужно добавить поправочный коэффициент).

Var(X) = 1 / (n - 1) \* Sum(x - mean(X)) ^ 2

=======

Сколько нужно экспериментов для оценки м.о.?

Доверительный интервал длиной 2ε, в который μ укладывается с вероятностью γ

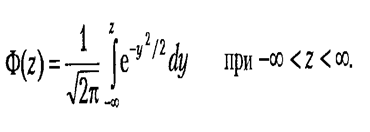
задано γ, ε, найти n

Центральная предельная теорема. Из нее не сложно вытащить число испытаний (при условии что распределение случайной величины одинаковое для всех испытаний) для заданной точности.

Ц.П.Т.: Нормированная оценка м.о. для n выборок сходится к

•Для нормированного распределения находим u(\gamma) по таблице

•Далее, \eps = u(\gamma)\*sqrt(\sigma/n)

•Определяем n исходя из требований к eps

## 

## 

## 

## **6.4. Общая схема проверки статистических гипотез, пример.**

(лекция 5)

