Ответы на экзаменационные вопросы по курсу
«Архитектура управляющих систем реального времени» (осенний семестр 2015 г.)

Оглавление

1. (Лекция 1) Состав ИУС РВ, основные виды устройств в составе ИУС РВ. Функции и специфика работы ИУС РВ. Реагирующие системы. Градации требований реального времени.

ИУС РВ обеспечивает интеграцию частей управляемого устройства (функциональную и информационную) и взаимодействие с оператором.
Состав ИУС РВ: регистраторы, вычислители, датчики, эффекторы, интерфейс оператора (индикаторы, органы управления), бортовая сеть.
Функции ИУСРВ: контроль состояния управляемого объекта, управления движением объекта и его частей, отслеживание движения объекта или его частей, отслеживание положения объекта и его частей в пространстве, обмен данными с внешними системами, управление специализированными приборами, обмен данными с оператором (отображение и ввод данных).
Специфика ИУСРВ: работа в реальном времени (ориентация для худших случаев), непрерывное функционирование, параллельное управление многим, интеграция с управляемой системой, критичность ошибки, устойчивость к сбоям, ограниченное участие оператора, предсказуемое поведение, экстремальные условия работы, ограничения по ресурсам, координация между ИУС взаимодействующих объектов.
Реагирующая система – ВС, функционирующая в постоянном взаимодействие с окружающей средой, с необходимой скоростью, задаваемой средой (директивные сроки).
Требования реального времени бывают: hard (при нарушении фатальная ошибка), firm (бесполезность результата), soft (снижение ценности результата).

Заблуждения о системах реального времени: работа в реальном времени != быстрая работа; рост производительности современных процессоров не решает проблемы; аппаратура может дать сбой, поэтому нужно уметь реконфигурироваться, создавать избыточность.

2. (Лекция 1) Развитие архитектуры ИУС РВ. Неоднородность ИУС РВ по типам каналов, устройств, данных.

Эволюция ИУС: полностью аналоговая система -> центральный вычислитель + аналоговые устройства -> федеративная архитектура (медленные каналы связи, спец. вычислители, локальная обработка данных) -> интегрированная модульная архитектура (быстрые каналы связи, виртуализация сетевых и вычислительных ресурсов, облако вычислительных модулей)

Отличие федеративной и интегрированной модульной архитектуры:
Федеративная структура получается при декомпозиции задач по функциональному назначению, которые висят на общей магистрали.
Интегрированная структура не имеет магистраль общего пользования, там соединяются те устройства, которым это нужно + идёт декомпозиция больше по данным, чем по функциям (на сколько это возможно) + идёт унификация интерфейсов (в физическом и логическом смысле).

Неоднородность ИУС РВ: каналы (точка-точка, шина, коммутатор, 12 kbps, 1Mbps, 1Gbps), устройства (датчики, индикаторы, вычислители, органы управления, исполнительные устройства), данные (аналоговые, цифровые, числовые массивы, видеопотоки)

Под исполнительным устройством (также называют актуа́тор, актюа́тор) в теории автоматического управления понимают устройство, передающее воздействие с управляющего устройства на объект управления.

Есть *проблема унаследованных устройств*.
Темпы роста – велики (1986г – 10KB, 1992г – 100KB, 1998г – 1MB, 2008г – 15MB)
Информационное сопряжение вычислительных задач: интерфейсы задач – входные и выходные данные, обмен между задачами в одном блоке – синхронные зависимости по данным, обмен по каналам передачи данных – сообщения, расписания обмена.

Жизненный цикл ИУСРВ (V-образный): фаза планирования -> разработка системных требований -> разработка архитектуры системы -> разработка требований для ПО высокого уровня -> разработка требований для ПО низкого уровня -> *кодинг, отладка, интеграция ПО* -> тестирование компонент ПО -> функциональное тестирование ПО -> системная интеграция -> системное тестирование -> ввод системы в эксплуатацию.
Инструментальные средства: разработка требований, управление версиями, отслеживание проблем и изменений, поддержка сопряжения подсистем ПО, проектирование индикационных форматов, проектирование алгоритмов, построение расписаний, конфигурирование сред обмена данными, верификация и тестирование ПО. У цепочки средств разработки должны быть сопряжены входы и выходы форматов данных, а так же требуется фиксация выходных артефактов.
Математические задачи: выбор оптимальной конфигурации (требования реального времени, надёжности, ограниченности по ресурсам), построение расписания вычислений и обмена данными, конфигурирование коммутируемой среды обмена данными, верификация работы (доказательная) (функциональная и временная), генерация тестовых покрытий.

3. (Лекция 2) Динамическое планирование задач в ИУС РВ. Схемы планирования Rate Monotonic (фиксированные приоритеты) и Earliest Deadline First (динамические приоритеты). Условия планируемости наборов задач при директивных сроках, равных периодам.

Режимы активации задач могут быть: time driven, event driven.
Планировщик – принимает решение, какая задача будет сейчас выполняться, если задач несколько.
Планировщик влияет на: время отклика задачи, задержку и jitter, время выполнения (важно для кэша и предупредительной загрузки данных), можно оптимизировать использование ресурсов, влияет на способ выдерживания перегрузок, можно оптимизировать потребление энергии.
C – длинна задачи
T – период задачи
D – дедлайн задачи
Задача динамического планирования: выполнить все задачи до их дедлайна, доказать выполнимость тасков ещё до самого запуска и выполнения.

FPS – fixed priority scheduling - самое распространённое; в реальных системах приоритет определяется динамически в зависимости от текущих требований, а не в зависимости от корректности и целостности системы. В зависимости от preemption, более высокоприоритетная задача может сразу после своего появления прерывать менее приоритетную, или же должна будет дождаться её окончания.
!!! Что значит эта фраза про корректность и целостность? Имеется ввиду, что приоритет не ставится в зависимости от важности задачи?
RM scheduling – Rate Monotonic – планирование на основе постоянных приоритетов.
EDF – earliest deadline first
FPS vs EDF – FPS легче алгоритмически и дешевле вычислительно, легче добавлять учёт других факторов в приоритет, нежели в оставшееся время дедлайна. При перегрузке FPS предсказуемее, а у EDF дедлайны могут посыпаться как домино. Однако EDF – мощнее и лучше организует процессорное время.

Характеристики планирования: sufficient (достаточное) – если некоторое условие (тест) для данной системы задач и выбранного планирования пройден, то дедлайны не будут нарушены никогда, necessary (необходимое) – если условие (тест) не выполнены для данной системы и данного планирования, то хоть один дедлайн будет нарушен, exact (точный) – достаточное и необходимое условие планирования, sustainable (устойчивый) – система остаётся планируемой, даже при улучшении условий (при уменьшении нагрузки работами).

Простая модель задач (simple task model) – фиксированный набор задач, все задачи с заданными периодами, все задачи независимы, нету накладных расходов системы (например, на переключение задач), дедлайн равен периоду задачи, время выполнения всех задач фиксированно.

B – наихудшее время блокировки задачи (для случая использования критических секций)
N – количество задач в системе

T – период появления задачи
C – худшее время выполнения самой задачи (WCET) (время, сколько выполняется сама задача)
U – утилизация задачи = C/T
D – дедлайн задачи
P – приоритет конкретной задачи
I – the interference time of task = сумма всех C_k задач, у которых приоритет выше, чем для текущей задачи.
R – worst-case response time of the task = interference I_k данной задачи + её время выполнения C_k

Rate Monotonic Priority Assignment – каждой задаче ставится в соответствие приоритет в соответствии с периодом, чем меньше период, тем выше приоритет. (чем меньше число, тем меньше приоритет) (подразумеваются системы, в которых дедлайн = периоду)
Утверждение: Если какая-либо задача может быть планирована при помощи предварительной установки фиксированных приоритетов, то эти задачи могут быть планированы при помощи rate monotonic priority assignment.

Формулы для оценки планируемости задач:
Предположения: задачи независимы; дедлайн равен периоду появления задач; $\Phi_i = 0$ (Φ - фаза).
Для RM (1973) – планируемо, если sum (C_i/T_i) <= n * ($2^{1/n}$ -1) – при n -> ∞, сумма -> ln2 ≈ 0.69 (“максимальная утилизация процессора в среднем при большом количестве задач”)
Для EDF (1973) – планируемо, тогда и только тогда, когда sum (C_i/T_i) <= 1
Вводится так называемый «регион планируемости»
Для RM (2000) – product (C_i/T_i + 1) <= 2 («hyperbolic bound»)

4. (Лекция 2) Оценка времени отклика задач для схемы RM при директивных сроках, не превышающих периоды; её использование для анализа планируемости наборов задач. Схема обоснования формул для оценки времени отклика.

Алгоритмы доказательства выполнимости при планировании, когда дедлайн меньше периода выполнения задачи:
FPS – RTA (Response Time Analysis)
EDF – PDC (Process Demand Criterion)

RTA – Responsive Time Analysis: считаем для данной задачи interference I время для того времени отклика R, которое у нас получится (оно ещё не известно), после этого считаем её время отклика R (просто добавляя время выполнения текущей задачи). Получаем рекуррентную формулу: $R_i[w_i^{n+1}] = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i[w_i^n]}{T_j} \right\rceil C_j$. Рекуррентная формула монотонно не убывающая. Когда $w_i^{n+1} = w_i^n$, то решение найдено, w_i^0 должно быть не больше, чем R_i, например 0 или C_i. После нахождения R_i сравниваем, что оно <=, чем дедлайн D_i этой задачи.
RTA – даёт «точное» (необходимое и достаточное) условие планирования.

5. (Лекция 2) Апериодические и спорадические задачи, их обработка при динамическом планировании. Серверы процессорного времени. Схема назначения приоритетов Deadline Monotonic, обоснование её оптимальности.

Sporadic tasks – задачи, которые появляются случайным образом, однако у них есть некоторый минимальный период (дедлайн ещё меньше периода), чаще которого они появляться не могут. RTA – для их тоже работает.
Aperiodic tasks – задачи, у которых нету минимального периода возникновения, однако они запускаются с приоритетами меньшими, чем у постоянных задач, и поэтому не воруют у них процессорное время.

Execution-time Servers – у них есть «бюджет», правила его пополнения и вес, который вынимается из бюджета при появлении апериодичных задач. Так регулируются эти задачи.
Типы execution-time servers: periodic server, deferrable server.

Periodic Server – имеет бюджет, восполняемый в каждый период T, начиная, например, с момента 0, задача может запуститься в момент 0, T, 2T, … и работает, пока бюджет позволяет, иначе она замораживается. Бюджет выкидывается, если клиенты за период T отсутствуют (idles away).
Deferrable Server – имеет бюджет, восстанавливаемый каждый промежуток времени T, клиент замораживается, если бюджет кончился, клиент может запуститься в любой момент времени.
DS называется «bandwidth preserving» (retain capacity as long as possible), PS – наоборот. (слайд 83)
!!! криво описаны PS и DS

Доказательство оптимальности DMPO (Deadline monotonic priority ordering) – (если система задач Q планируема на основе схемы с приоритетами W, то она планируема с помощью deadline monotonic). Доказательство основано на том, чтобы последовательно менять приоритеты задач, пока не получится разметка приоритетами, как и у DMPO, при этом каждый шаг смены приоритета будет выполнимую систему оставлять выполнимой.
Модификация: допустим в схеме W: $P_i > P_j$ && $D_i > D_j$, для соседних приоритетов P_i и P_j, тогда пусть в схеме W' приоритеты этих задач будут поставлены наоборот.
Если множество задач Q планируемы при помощи W', то тогда данная модификация никак не повлияет на планируемость как менее приоритетных задач, так и более приоритетных, т.к. все они будут испытывать всё ту же interference. Задача j, которая теперь имеет более высокий приоритет, тем более будет планируема. Осталось показать планируемость для задачи i.
В схеме W: $R_j < D_j$, $D_j < D_i$ and $D_i <= T_i$. Поэтому задача i interferes лишь один раз во время выполнения задачи j (i перебивает выполнение задачи j). В связи с чем, $R_i' = R_j <= D_j < D_i$, поэтому задача i планируема после обмена приоритетами.
Таким образом модификация оставляет планируемую систему планируемой, и при этом позволяет свестись к схеме DMPO.

6. (Лекция 3) Критические секции. Инверсия приоритетов, пример Mars Pathfinder. Схемы наследования приоритета и потолка приоритета.

Критическая секция – непрерывная последовательность действий одной задачи, в рамках которой задачу нельзя прерывать для выполнения других критических секций из других задач.
В обыденной программистской жизни решается методом семафоров, мьютексов и блокирующих передач.
Priority inversion – пример, когда из-за блокировок, связанных с критическими секциями по факту получается, что задача с меньшим приоритетом выполняется раньше. (это происходит, когда задача с меньшим приоритетом (но без критической секции) приостановила задачу с ещё меньшим приоритетом, когда та находилась в критической секции, из-за чего самую старшую задачу выполнять нельзя, т.к. ей тоже уже (её период начался чуть позже) нужна критическая секция)

У MarspathFinder случилась ровно проблема priority inversion в VxWorks? Там потокам присваивались приоритеты. В качестве самого приоритетного потока был менеджмент общей шины, который умел блокировать по mutex доступ к ней, а самым низкоприоритетным потоком была задача сбора метеорологических данных, также была задача обмена информацией (с землёй), которая имела средний приоритет.
В итоге, запускалась низкоприоритетная и захватывала mutex, потом запускалась среднеприоритетная, и потом запускалась высокоприоритетная и ждала, пока отпустят mutex, но та задача ждала, пока закончит работать долгая среднеприоритетная. В итоге высокоприоритетная долго не выполнялась и “watchdog” это замечал, думал что всё пошло наперекосяк и делал глобальный reset.

Решение проблемы priority inversion:
Disallow preemption – запрет переключения выполняемой задачи, пока та находится в критической секции.
The priority inheritance protocol – если какая-то задача заблокировала одновременно несколько других задач, то её приоритет временно становится равен максимальному приоритету из заблокированных задач.
Схема потолка приоритета (priority ceiling protocol) (OCPP – Original ceiling priority protocol) – каждому ресурсу присваивается некоторый уровень приоритета, равный максимальному приоритету из задач, которые могут захватить этот ресурс. Задача i может войти в критическую секцию только тогда, когда приоритет задачи выше, чем приоритеты всех ресурсов, которые на данный момент захвачены другими задачами. Если задача начинает

блокировать одну из более приоритетных задач, то она временно получает приоритет равный приоритету ресурса, который она захватила.
Другая модификация потолка приоритета – ICPP – Immediate Ceiling Priority Protocol. Разница только в том, что потоку, захватывающему ресурс, сразу повышается приоритет до приоритета этого ресурса, не дожидаясь, пока придёт другая более приоритетная задача и заявит, что её заблокировали.

Плюсы ceiling protocol: невозможен deadlock, задача может быть блокирована не дольше, чем на время одной критической секции (такое могло случаться в the priority inheritance protocol), транзитивная блокировка также не может возникнуть.

Модификации RTA для более сложных случаев:
RTA – Responsive Time Analysis в случае наличия критических секций модифицируется следующим образом: $R_i[w_i^{n+1}] = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i[w_i^n]}{T_j} \right\rceil C_j$, где B_i – это максимальное время блокировки задачи в ожидании разделяемого ресурса (определяется из того, какие ресурсы задаче необходимо захватывать, и на какое максимальное время эти ресурсы могли захватить другие задачи)
!!! Вопрос: правильно ли я понимаю, что максимальное B = max (по каждому ресурсу, что блокирует эта задача; max (для каждой задачи, которая блокирует рассматриваемый ресурс; время которое будет держать этот ресурс эта задача)), альтернативным мнением является не max(max), а sum(max).

RTA – Responsive Time Analysis в случае, если дедлайн > T:
$R_i(q)$ – это worst case response time для задачи i, с учётом того, что впереди будет выпущено ещё q экземпляров.
$R_i(q)\left[w_i^{n+1}(q)\right] = (q+1)C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i(q)[w_i^n(q)]}{T_j} \right\rceil C_j =$ рекуррентная формула для вычисления $w_i(q)$
!!! А не забыли ли мы B тоже умножить на (q+1), ведь теоретически каждый выпущенный экземпляр задачи может быть прерван независимо от других. Почему мы не учитываем задержку для последующий q экземпляров?
$R_i(q) = w_i^n(q) - qT_i =$ худший случай выполнения одного экземпляра задачи i, при условии, что впереди выполнятся ещё q задач.
В $R_i(q) \le T_i$ – нужно рассматривать только те q = 1,2, …, для которых это неравенство не выполнено. (т.е. это ограничение q сверху)
$R_i = \max_{q=0,1,2,\dots} R_i(q)$ = результирующий worst case response time для задачи i

В некоторых системах множество приоритетов ограничено (Ada – 31, RT-POSIX – 32, RT_Java - 28), в связи с чем, некоторым задачам приходится присваивать одинаковые приоритеты, в таких случаях в алгоритме RTA – необходимо предполагать, что некоторую задачу с приоритетом p, будут перебивать (interfere) все другие задачи с приоритетом p.

RTA нельзя модифицировать так, чтобы он подходил для анализа в случае наличия Ф (фаза) – начальный отступ при старте задачи (arbitrary offset). Но можно прибегнуть к неоптимальному анализу:
Не оптимальный анализ:
В реальных системах часто задачи зависимы (и одна должна выполняться строга после того, как закончится другая). Можно, например, присвоить одной из 2-х задач половинчатый период и потом слить её с оставшейся задачей, получив одну задачу из 2-х, с последовательным выполнением, отсутствием и offset. Эдакий merge. (например, $T_n=T_a/2=T_b/2$, $C_n=\max(C_a, C_b)$, $D_n=\min(D_a, D_b)$, $P_n=\max(P_a, P_b)$, это можно расширить на большее количество задач)
Если планирование не удастся, это ещё не значит, что спланировать исходные задачи было нельзя, но если удастся, то это значит, что и исходные задачи планируемы и что планируемыми останутся и более приоритетные задачи (т.к. исходные задачи кушают меньше cpu). Таким образом, анализ – не оптимален.

7. (Лекция 3) Критерий потребности в процессорном времени (processor demand) для оценки планируемости задач в схеме EDF. Схема обоснования этого критерия.

Критерий потребности в процессорном времени (processor demand criterion): используется для оценки планируемости задач по схеме EDF (Earliest Deadline First).

Для каждого интервала времени (t1; t2), время вычисления g(t1, t2), требуемое всеми задачами должно быть не больше времени (t2-t1). g(t1, t2) – это сумма wcet тех задач, которые начались в момент t1 или позже и с дедлайном меньшим либо равным t2. $g(t1, t2) = \sum_{r_i \geq t_1}^{d_i \leq t_2} C_i$

Для синхронных задач (!!! может периодичных?) (каких задач? 126 слайд), можно анализировать лишь интервал от 0 до L: $g(0, L) = \sum_{i=1}^{n} \left\lfloor \frac{L - D_i + T_i}{T_i} \right\rfloor C_i \leq L$

Верхняя граница для processor demand теста: $L_a = \max(D_1, \dots, D_n, \frac{\sum_{i=1}^{n} (T_i - D_i) C_i / T_i}{1 - U})$ (U – утилизация всего множества задач) (слайд 129)

Для EDF была формула, что в случае D=T, задачи планируемы тогда и только тогда, когда sum (C_i/T_i) <= 1

При добавлении учёта блокировок формула становится = sum (k=1,i-1 ; C_k/T_k) + (C_i+B_i)/T_i <= 1, ∀i

В случае EDF, когда D<=T, то ∀i, ∀L должно выполняться $B_i + \sum_{i=1}^{n} \left\lfloor \frac{L - D_i + T_i}{T_i} \right\rfloor C_i \leq L$

!!! надо бы подробнее написать, почему формула такая

NP Scheduling – Non-preemptive scheduling – это когда все задачи на протяжении всего своего выполнения захватывают один и тот же ресурс, или по другому: это когда никакая задача не может быть прервана.

В таком случае B_i определяется как $B_i = \max\{C_k: P_k < P_i\}$.

Плюсы: уменьшение накладных расходов (не нужны семафоры и нету затрат на переключение контекста); уменьшает размер стека, т.к. лишь одна задача в данный момент выполняется; улучшаются характеристики кэша, заблаговременных очередей (prefetch queues), канальных механизмов.

Как следствие, для NP Scheduling – задачи выполняются быстрее и предсказуемее.

В некоторых случаях улучшается возможность планирования даже для случая FPS (fixed priority scheduling)

Минусы: в общем случае, возможности планирования ухудшаются; можно придумать случай, чтобы для 2-х задач утилизация стремилась к нулю, но они никогда не были планируемы.

Tunable Preemptive Systems (настраиваемые системы с вытеснением):

Можно просто вычислить (подобрать) максимальную длину непрерываемой работы, при которой система успешно планируется.

Можно установить некоторые конкретные точки, в которых система может прервать задачу и взять другую.-

8. (Лекция 3) Джиттер (флуктуация задержки) при динамическом планировании. Виды джиттера, подходы к его минимизации. Сравнение схем RM и EDF с точки зрения джиттера. Планирование задач без вытеснения: преимущества и недостатки, использование для борьбы с джиттером.

Jitter – вариация по времен задержки (дисперсия). В большинстве приложений задержка и jitter могут вызывать нестабильность.

Input Latency (start time delay) – входная латентность (задержка начала выполнения задачи после её выпуска) (есть для каждого выпуска задачи каждый период)

IJ - Input Jitter (Start time Jitter) – входной джиттер бывает абсолютный и относительный

Абсолютный входной джиттер = максимальная входной латентности – минимальная входная латентность

Относительный входной джиттер = максимальной разнице входной латентности 2-х выпуском задач идущих подряд.

Response Time (Output Latency) – время от выпуска задачи до её завершения.

RTJ - Response Time Jitter (Output Jitter) – аналогично бывает абсолютным и относительным.

Input-Output Latency – латентность входа-выхода – время от начала выполнения задачи, до её завершения.

IOJ - Input-Output Jitter – тоже бывает абсолютным и относительным (определяется аналогично – как разность max и min и как max разность 2-х подряд идущих латентностей)

RM vs EDF:

При RM (FPS – fixed priority) – у низкоприоритетных задач высокий уровень джиттера и задержек (всех видов).

При EDF – немного выше RTJ, однако IOJ = 0 (что в принципе хорошо, потому что минимизируется количество задач, которые в случае провала и прерывания заберут вместе с собой некоторое количество зазря потраченного времени).
Рассмотрев 10 задач, с возрастающим периодом (и падающим приоритетом), получилось, что с точки зрения RTJ, RM немного стабильнее выполняет задачи с высоким приоритетом, по сравнению с EDF, но гораздо менее стабильно выполняет задачи с наименьшим приоритетом.
(на слайдах есть хорошие картинки)

Два метода, для уменьшения эффекта задержки и джиттера:
1) Компенсировать их за счёт действий по управлению.
2) Уменьшить их как можно больше.
Однако даже после компенсации, обычно можно ещё уменьшить задержку и джиттер (поэтому рассматривается дальше именно уменьшение)
3 способа уменьшения джиттера, вызванного перебиванием одной задачей другой (interference).
1) *Разбиение задачи* – у любой задачи есть ввод, вычисления и вывод. Ввод выполняется сразу после выпуска задачи, вывод – непосредственно перед дедлайном, вычисления между ними – в соответствии с планированием задач. Ввод и вывод можно даже делать по тригерам, а не планированию.
Плюсы: джиттер всегда будет минимален, и если вводы и вывод малы, то данное решение эффективно почти всегда при любом планировании.
Минусы: Джиттер уменьшается, но увеличивается задержка; ввод и вывод вводя дополнительные прерывания других задач, что усложняет анализ и планируемость; ввод и вывод разных задач могут хотеть выполниться одновременно, надо это как-то решать.
2) *Advancing deadlines* – идея в ужесточении дедлайнов, чтобы уменьшить активное окно, в течение которого задача должна быть выполнена.
Плюсы: легко реализовать (не нужна поддержка ОС); нету дополнительных вмешательств (interference); и задержка и джиттер уменьшаются.
Минусы: не для всех задач можно уменьшить джиттер до 0, для некоторых нужно вводить отступ, но сложность в этом случае экспоненциальная, ужесточение дедлайнов ухудшает планируемость системы.
3) *Невытесняющее планирование* – Плюсы: для него IOJ = 0, IOL = C_i
Плюсы и минусы: традиционные плюсы и минусы NP планирования (см. предыдущие билеты)

9. (Лекция 3) Планирование вычислений при перегрузке системы. Особенности работы схем RM и EDF в условиях перегрузки. Схема компенсации перегрузки на основе растяжимых («эластичных») периодов задач.

RM vs EDF:
RM – высокоприоритетные задачи выполняются как надо, низкоприоритетные задачи – полностью блокированы.
EDF – все задачи выполняются так, что порой падают, но зато все задачи время от времени выполняются.
(примеры на слайдах)

EDF под нагрузкой: Теорема (Ceervin ‘03)
Если утилизация системы U > 1, то EDF будет выполнять задачу i с общим периодом $T_i' = T_i * U$.

Составляющие уменьшения нагрузки:
1) Увеличение дедлайна или периода задачи
2) Каждая задача должна задавать диапазон значений, которые может принимать период
3) увеличение периодов при перегрузках, и уменьшение при нормальном функционировании.
(часто период задачи зависит от каких-то внешних факторов (например, ближе препятствие – чаще надо делать до него замеры))

Elastic task model – гибкая модель задачи, в ней задаются, но мимо обычных параметров задачи – её минимальный период, максимальный период, и E_i – elastic coefficient.

При перегрузке период вычисляется следующим образом = $T_i = C_i / U_i$, где $U_i = U_{i0} – (U_0 - U_d)E_i/E_s$, где U_d – желаемый уровень утилизации системы, U_0 – текущий уровень утилизации системы, U_{i0} – текущий уровень утилизации задачи, E_s – сумма всех коэффициентов эластичности.

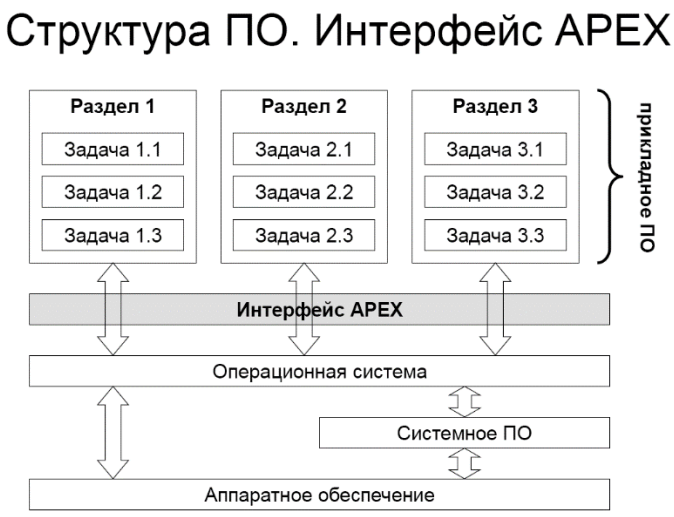
Road map: считаем наихудшее время выполнения задач системы, используем подходящий алгоритм планирования и подходящий протокол доступа к ресурсам (алгоритм обслуживания критических секций), вычисляем максимальные времена блокировок для задач (на основе алгоритма работы с критическими секциями), применяем анализ планируемости (чтобы понять может ли наша система задач так работать), и в конце немного меняем различные параметры задач и проверяем, что система всё ещё планируема.

10. (Лекция 4) Архитектура интегрированной модульной авионики (ИМА), её отличия от федеративной архитектуры ИУС РВ. Организация программного обеспечения в системах ИМА: разделы, информационное взаимодействие между разделами и внутри разделов. Схема статикодинамического выполнения задач в системах ИМА.

Федеративная ИУС РВ: блоки специализированы по назначению и архитектуре, ПО различных подсистем на различных блоках (нет конкуренции за процессор, изоляция по памяти).
Недостатки: низкая переносимость и повторная используемость, «зоопарк» архитектур и интерфейсов.

ИМА - Интегрированная, модульная архитектура: логически единый распределённый вычислитель (единая архитектура, унифицированные модули, унифицированные программные интерфейсы), разделение ресурсов между ПО различных подсистем.
Проблемы: конкуренция за процессорное время, изоляция по памяти.
Для ИМА характерно:
Стандартное API со стороны ОС.
Статическое разделение времени, памяти и ресурсов.
Преимущества: надёжно, переносимо, возможность повторного использования, модульность, упрощение верификации и сертификации.

Структура ПО в ИМА:

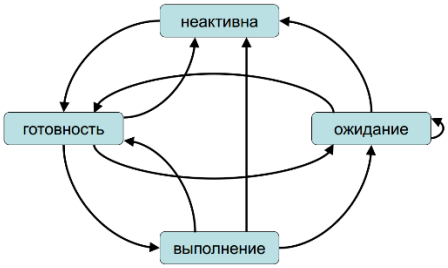


часто каждой подсистеме даётся свой раздел
APEX – Application Executable (например, ARINC 653 – регламентирует временное и пространственное разделение ресурсной авиационной ЭВМ в соответствии с принципами интегрированной модульной авионики)
Взаимодействие между разделами:
Порты с очередью сообщений
Порты с перезаписью сообщений (буфер фиксированной длинны, сообщение перезаписывается, отправка с заданным периодом)
Взаимодействие внутри разделов:
Обмен данным (передача сообщений, общая память), механизмы синхронизации (семафоры и события)

Рабочая нагрузка: периодические задачи, сообщения, зависимости по данным.

На практике данные могут передаваться между следующими задачами:
1) задачи с одинаковой частотой («синхронная» зависимость по данным)
2) отправитель выполняется чаще получателя (у получателя всегда актуальные данные, ждать нет смысла)
3) отправитель выполняется реже получателя (если получатель будет ждать, то сорвёт свои сроки)

Состояние задач:



Выполнение задач в системе:
Выделяются окна (статическое расписание, границы окон одинаковы для оного модуля (в случае многоядерного модуля)). В рамках одного окна могут выполняться задачи только одного раздела, и переключение контекста между разделами происходят только при смене окна.
В рамках окна работает динамическое планирование задач: очередь выполнения, приоритеты, вытеснения, ожидания входных данных, …
Незавершённая задача из раздела, может быть продолжена в следующем окне.
!!! Слайд 183 – что за волшебная красная линия?

11. (Лекция 4) Задача планирования вычислений в системах ИМА. Жадный алгоритм привязки разделов к процессорным ядрам. Алгоритм построения набора окон. Роль модели вычислительной системы при планировании вычислений в системе ИМА, схема работы модели. Проблема использования только наихудших оценок времени выполнения задач.

Входные данные задачи планирования:
Описание системы ИМА (набор модулей с набором и типом процессорных ядер у которых есть верхняя граница загрузки) (модуль -> ядро -> макс нагрузка)
Описание рабочей нагрузки (наборы задач (период, приоритет, WCET для типа ядра), сообщений (отправитель, получатель, размер, длительность передачи (через память, сеть)), разделов (задачи, допустимые ядра), свободные задачи (допустимые ядра))
(свободные задачи – это задачи, которые ещё не сгруппированы в разделы)
Цели задачи планирования:
Составить разделы из свободных задач (минимизация трафика между разделами и загрузка ядра не выше его максимальной утилизации (при выполнении на одном ядре))
Привязать разделы к ядрам (минимизация трафика между разделами, загрузка ядра не больше максимальной утилизации, выполнение условий динамической планируемости - sum (T_i*F_i) <= n * ($2^{1/n}$ -1) (T – период задачи, F – ? !!!), инкрементальная привязка (расширение прежней))
Построить расписание окон для каждого ядра (корректность проверяется моделированием работы динамического планировщика; расписание корректно, если все работы выполняются в пределах директивных сроков)
Ограничения корректности: раздел привязан ровно к одному ядру, не больше одного раздела на ядро, окна не пересекаются во времени, работы выполняются в рамках окон раздела, в каждый момент времени выполняется не более одной работы, все работы выполняются полностью, выполняются зависимости по данным, соблюдаются приоритеты.

Жадный алгоритм:
1) Выбрать непривязанный раздел P с максимальным трафиком между P и привязанными разделами.
2) Для каждого модуля рассчитать трафик между P и разделами, привязанными к ядрам других модулей.

3) Упорядочить модули по убыванию этого трафика.
4) В цикле по модулям:
- если раздел P может быть привязан к какому-либо ядру данного модуля без нарушения ограничений на загрузку ядра, то привязать P к этому ядру и выйти из цикла
- иначе, если разделы на данном модуле могут быть перераспределены между ядрами этого модуля так, чтобы в достаточной мере «разгрузить» одно из ядер, то выполнить перераспределение, привязать P к этому ядру и выйти из цикла
5) Если на шаге 4 не выбрано никакое ядро, то стоп (неуспех).
6) Если остались непривязанные разделы, то перейти к шагу 1, иначе стоп (успех)

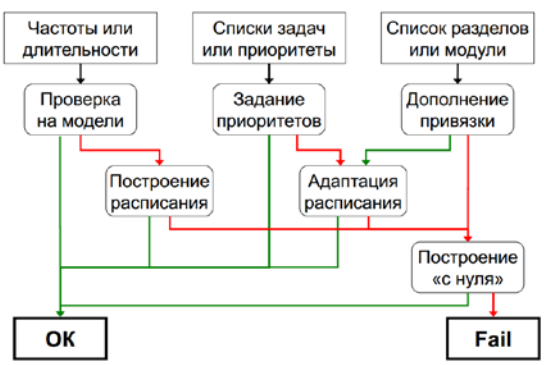
Другие алгоритмы привязки:
Метод ветвей и границ (критерий отсечения: загрузка ядра превышает допустимую, или ранее найдено лучшее решение, недостаток: время выполнения на реальных данных)
Упаковка в контейнеры: ядро = контейнер, раздел = объект (объём – вклад раздела в загрузку ядра, стоимость – трафик, становящийся внутренним для модуля в случае привязки раздела к ядру этого модуля), проблемы: объём объекта зависит от выбора контейнера, стоимость контейнера зависит от расположения других объектов.

Построение расписания окон (схема алгоритма):
1) Построение графа зависимостей работ
2) Уточнение директивных интервалов работ на основе зависимостей
3) Построение последовательностей выполнения работ «слева направо», параллельно для всех ядер
Работа помещается в список готовых для выполнения, если:
- Начался ее директивный интервал
- Получены все синхронные входящие сообщения для данной работы
Очередная работа для размещения выбирается из списка:
- Если отсутствуют готовые работы с большим приоритетом
- По критерию EDF (если приоритет еще не определен)
4) Построение расписания окон на основе построенной последовательности работ
5) Назначение приоритетов задач на основе построенной последовательности работ

Модель вычислительного процесса - служит для проверки корректности расписания окон.
Модель основана на событиях. Типы событий:
Открытие окна / Закрытие окна / Начало директивного интервала / Завершение работы / Доставка сообщения
Схема работы:
1. Создание начальных событий
2. Выбор событий с минимальным временем
3. Обновление списка готовых к выполнению работ
4. Обработка выбранных событий
5. Если список событий не пуст, переход к п.2

Недостатки планирования: может получится так, что если некоторая задача выполнится быстрее, и пошлёт результат работы другой задаче, то та другая может на своём ядре начать выполняться, прервав 3-х задачу, и сорвав её сроки, а, если бы сообщение пришло позже, то 3-я задача успела бы выполниться и всё было бы хорошо.

Адаптация к изменениям:



Перспективки:
Построение расписания с минимизацией затрат на переключение контекста
Корректировка распределения разделов по ядрам при неуспешном построении расписания
Построение расписания, гарантированно корректного при временах выполнения задач, меньших чем WCET
Минимизация сетевой загрузки с учётом синхронности зависимостей по данным
– сообщения, которые ожидает задача-получатель, следует передавать через память (=> привязка отправителя и получателя к одному модулю)
Совместное планирование задач и конфигурирование сети

12. (Лекция 5) Ограничения на процессоры в ИУС РВ, источники этих ограничений. Проблемы применения в ИУС РВ высокопроизводительных процессоров общего назначения («настольных», «серверных»). Примеры специализации процессоров: мультимедийные команды, специализированные регистры, множественные банки и шины памяти, устройства вычисления адресов, адресация по модулю.

Ограничения на процессоры ИУС РВ:
Технологические ограничения:
- вынужденное применение «грубого» технологического процесса
- жёсткие ограничения по энергопотреблению и тепловыделению
Откуда берутся ограничения
- требование к устойчивости к внешним воздействиям (излучение и т.п.)
- неразвитость технологических процессов производства микросхем
- общий лимит на энергопотребление ИУС РВ
- ограниченные возможности теплоотвода

Производительность на единицу энергии падает, универсальность растёт:
Процессоры общего назначения -> ASIP (Application Specific instruction set processors) (микроконтроллеры, DSP (Digital signal processors)) -> программируемое железо (FPGA – Field Programmable Gate Arrays) -> ASIC (Application-specific integrated circuits)
Эффективность затрат энергии растёт примерно в 10 раз, примерно каждые 5 лет уже с 1990

Процессоры общего назначения: имеют высокую производительность (очень оптимизированы, используют параллелизм (конвейер, предсказание jmp, dynamic scheduling of insructions), сложная иерархия памяти (кеши)).
Из-за этого: время выполнения задачи непредсказуемо из-за большой динамичности.
Большая производительность, но и большое потребление.
У многоядерных процессоров: потенциально дают большую скорость за счёт параллелизма, очень полезны для встроенных систем (совместных). Проблемы: увеличение interference для разделяемых ресурсов, шин и кэшей, ещё меньше предсказуемости, часто параллелизм – ограниченный (не всегда можно что-то распараллелить, и тогда у нас будет ненужное ядро, кушающее питание)

Особенности для специализированных систем:
В них меньше гибкости, но её должно быть достаточно для поздних модификаций и отладки, специализированная система должна покрывать некоторый класс задач (т.е. есть некоторые варьирующиеся параметры - настройки).
Для создания специализированной системы, нужно проанализировать свойства целевого приложения.
Примеры: мультимедиа операции (сейчас регистры на машинах большие, но цвета например – всего 8 бит и с ними можно делать операции сразу пачками), гетерогенные регистры (используемые для разных целей), множественные банки памяти (!!! слайд 215 – что там происходит?), address generation units (слайд 216 – что там происходит?), modulo addressing (удобно для реализации кольца или циклического буфера в памяти)

13. (Лекция 5) Специализированные процессоры: микроконтроллеры, процессоры цифровой обработки сигналов (DSP), процессоры с длинным командным словом (VLIW). Проблема загрузки ресурсов VLIW-процессора. Программируемые логические интегральные схемы

(ПЛИС, FPGA), их структура и принцип применения. Специализированные микросхемы (ASIC), ниша для их рационального применения.

Control Dominated Systems – динамические системы на основе событийного принципа работы.
Часто семантика такой системы представима в виде сетей Петри или конечной машины состояний.

Микроконтроллер – может управлять приложениями (поддерживает синхронизацию и планирование процессов, переключение контекстов, малые величины задержек), низкое энергопотребление, периферийные устройства часто интегрированы, подходит для задач реального времени.
Пример микроконтроллера, как «System-on-chip» (Philips 83 C552): процессор, таймеры, watchdog, ROM, RAM, A/D converter (аналого-цифровой преобразователь), parallel ports, I2C bus (такая последовательная шина данных).

Data Dominated Systems – системы, ориентированные на потоки данных, с как правило периодичным поведением. Семантика обычно может быть описана графом потока управления. (пример приложений – обработка сигналов, control engineering)
DSP – Digital Signal Processor – Цифровой сигнальный процессор – оптимизирован для цифровой обработки сигналов, подходит для простого потока управления, содержит аппаратные элементы параллельной обработки (VLIW), специализированный набор команд, высокая скорость обработки данных, zero-overhead loop (например, команда «повтори следующую инструкцию n раз»), специализированная память.
Подходит для вычислений реального времени.
MAC – Multiply and Accumulate – инструкция, которая берёт значение по некоторому указателю (регистру) и что-то с этим значением делает (mul, sum …), а регистр увеличивает на 1, тем самым трактуя его как счётчик.

VLIW – Very Long Instruction Word – параллельные операции закодированы одной длинной командой, каждая закодированная инструкция в команде контролирует конкретную функциональную единицу (например, память, регистр …). Идея в том, чтобы компилятор находил возможный параллелизм в инструкциях и составлял соответствующим образом ассемблерный код.

!!! Проблема загрузки ресурсов процессора VLIW (из формулировки билета) – что отвечать?
Наверно нужно сказать, что не всегда код работает с различными ресурсами одновременно, и не всегда можно выявить параллелизм в ассемблерных командах (часто они друг от друга зависят, многое зависит от компилятора)

FPGA – Field Programmable Gate Arrays – состоит из логических единиц, входов и выходов и соединений.
Логические единицы могут быть: памятью, gate, tables, функциональными блоками (ALU, processor, control, data path)
!!! Что за gate и tables???
Коммуникационная сеть: матричная, дерево или иерархическая сеть.
Конфигурация вшивается во время производства, изобретается при дизайне и динамически меняется во время исполнения.

!!! Что нужно сказать о принципе применения ПЛИС?
ПЛИС программируются на определённом языка в отдельной IDE.

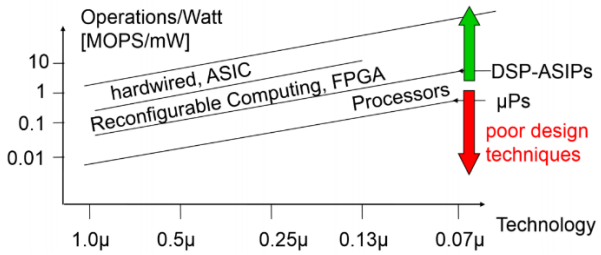
ASIC – Application Specific Circuits (ASIC) – необходимы, если нужна очень большая скорость и энергетическая эффективность, и при этом может быть продано много экземпляров (или цена за 1 шт будет заоблачная).
Минусы подхода: длинные сроки создания, отсутствие гибкости, высокие цены.

14. (Лекция 5) Проблема энергопотребления вычислительной системы, актуальность этой проблемы для ИУС РВ. Связь между мощностью и энергопотреблением; основания для минимизации каждой из этих характеристик системы. Связь между частотой работы процессора, напряжением питания и затратами энергии на такт работы. Использование

параллелизма для снижения энергопотребления (конвейер, множественные устройства, VLIW-архитектура).

Энерговыделение процессоров активно растёт. Измеряется сейчас в сотнях ват на квадратный сантиметр. Вся электроэнергия уходит исключительно в тепло.

Operations per Watt:

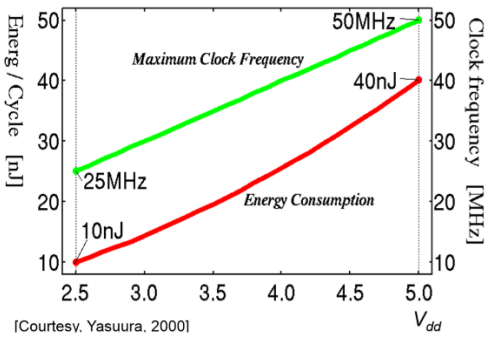


Из-за этого необходимо оптимизировать софт- и хард-ware и использовать гетерогенные архитектуры.
Если производительность увеличивается не экстенсивным путём (т.е. на за счёт потребления), то это обычно означает меньшее потребление.

Связь между мощностью и энергопотреблением: интеграл.
Минимизация потребляемой мощности (ватт в секунду) важна для: дизайна энергосети и регуляторов вольтажа, размеров подключения (физические размеры), охлаждение (как цена за охлаждение, так и ограничение на количество тепла в объёме).
Минимизация потребляемой энергии (ватт на операцию) важна для: ограниченного запаса энергии в мобильных системах, ограниченных запасах батарей (медленно заряжаются), большая стоимость электроэнергии (солнечные батареи), долгое время службы и низкие температуры.

DVS – Dynamic Voltage Scaling:
*Мощность ~ <(!!!) switching activity - переключение активности> * <load capacity - производительность> * <напряжение питания>^2 * <clock frequency - частота>*
Задержка для CMOS цепей (это технология построения электронных схем) ~ *<load capacity - производительность> * <напряжение питания> / (<напряжение питания> - <порог напряжения>)^2*
Т.к. порог напряжения << напряжения питания, то фактически задержка обратно-пропорциональная напряжению питания.
Таким образом, уменьшение напряжения питания уменьшает потребление пропорционально квадрату, а замедление задержки (т.е. максимальная частота процессора (т.е. производительность процессора)) происходит обратно пропорционально напряжению в первой степени.
Поэтому нужно уменьшать производительность железа до минимума, чтобы решать поставленную задачу, и при этом экономя на энерговыделении в квадрате.
Таким образом, сначала вычисляем, до скольких мы можем снизить частоту процессора (линейно снижает производительность и потребление), отсюда вычисляем до скольких можно снизить напряжение (в квадрате снижает потребление).

Пример зависимости напруги, максимальной частоты и потребления на одну операцию:



Параллелизм (например, x2) позволяет выполнять операции на каждой из параллельных компонент в 2 раза медленнее, а значит там можно в 4 раза снизить общее потребление, т.е. параллелизм на 2 приводит к выигрышу по энергии в 4 раза.
Аналогично и с конвейерной обработкой – если разбить задачу на 2 последовательных и выполнять каждую в 2 раза медленнее, то при той же производительности можно снизить потребление в 4 раза.

VLIW – даёт большую степень параллелизма (много отдельных вычислений), при этом без особого усложнения железа (т.к. инструкции обрабатываются независимо друг от друга), и само распараллеливание производится компилятором.

Есть 2 подхода, как сделать железо:
1) сделать несколько компонент с разным вольтажом и частотой
2) сделать одну компоненту, которая способна изменять свой вольтаж и частоту в зависимости от ситуации

15. (Лекция 5) Динамическая регулировка напряжения процессора. Выбор оптимального напряжения питания (на примере). Алгоритм YDS планирования вычислений с минимизацией энергопотребления за счёт регулировки напряжения питания. Динамическое управление питанием; характер вычислительной нагрузки, при котором эффективна эта технология.

YDS алгоритм офлайн планирования.
Предварительное планирование независимых задач, для которых известно их время выпуска, дедлайн и время выполнения для процессора в «нормальном состоянии» (с нек. стандартной частотой).
Интенсивность G(z, z') для некоторого интервала: суммарная утилизация системы задач (общее время выполнения задач (в «нормальном» режиме) делить на длину интервала), которые выпустились в этот интервал и дедлайн у них тоже в этом интервале.
Шаг 1: Перебираем все комбинации интенсивности G (надо рассмотреть все возможные пары выпуска какой-либо работы и дедлайна какой-либо работы), после этого выбираем интервал с максимальной интенсивностью («критический интервал») и считаем для него необходимую частоту процессора, чтобы справиться с задачами, полностью вошедшими в этот интервал.
Сами задачи в рамках этого интервала планируются по алгоритму EDF.
Шаг 2: Изменяем исходные параметры всех задач (их время выпуска и дедлайн), «выкусывая» интервал из предыдущего шага.
Шаг 3: снова выполняем шаг 1 для новой комбинации задач.
Шаг 4: Восстанавливаем по шагу 1, порядок и нужную скорость процессора для всего промежутка времени с 0, до конца.
Результаты алгоритма:
Алгоритм гарантирует минимальные энергозатраты, при выполнении директивных сроков.
Сложность алгоритма O(N^3), где N – количество задач. O(N^2) занимает поиск критического интервала, а количество итераций – максимум N.
Для периодических задач, у которых дедлайн = периоду, работа на постоянной скорости с 100% утилизацией с EDF планированием – имеет минимальные энергозатраты, и выполняет директивные сроки.
(пример на слайдах)

Алгоритм online планирования (для одного процессора):
Снова используем EDF для планирования задач, однако теперь, в каждый момент времени, смотрим какие задачи мы должны сейчас выполнить, и подбираем нежную частоту процессора, чтобы успеть выполнить задачи до их дедлайна.
При сравнении с оптимальным offline планированием, online даёт потребление в не более, чем 27 раз большее.

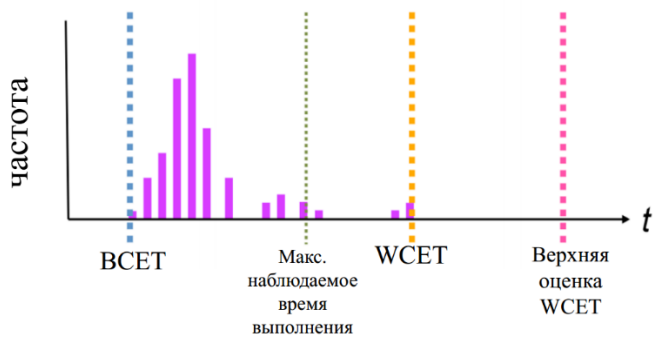
DPM – Dynamic Power Management – задача в том, чтобы правильно выбирать оптимальные режимы работы (работа, ожидание (процессор не используется, но следит за прерываниями) или сон (выключение активности чипа)). Обычно можно менять режимы как угодно, но нельзя из sleep в idle.
Sleep ест меньше всех энергии, но требует период выключения и включения. Overhead должен окупаться.

Procrastination Schedule – отложенное планирование:
Используем алгоритм YDS для планирования, но если где-то вольтаж опускает ниже чего-то, то его нужно поднять до некоторого критического порога (выбран заранее).
После этого выполнение некоторых задач нужно как можно сильнее отложить, чтобы уменьшить количество засыпаний, и увеличить длину периодов сна.
Отдельная проблема: выбор критического вольтажа.

16. (Лекция 6а) Понятие WCET. Актуальность WCET для анализа времени отклика задач в ИУС РВ. Типичное распределение времён выполнения программы на различных данных. Требования к оценке WCET. Проблемы при оценке WCET методом замеров. Два основных фактора, определяющих WCET. Зависимость длительности выполнения пути в программе от длительностей команд, входящих в путь (для простой и для реалистичной архитектуры).

Простейшая вычислительная задача WCET:
Входные данные доступны в момент старта
Выходные данные готовы в момент завершения
Нет блокировок в процессе выполнения
Нет синхронизации или обмена данными в процессе выполнения
Время выполнения зависит только от (входных данных и состояния задачи в момент старта)

Типичное распределение времён выполнения программы на различных данных:



Наихудшее время выполнения программного кода (worst case execution time, WCET) – это максимальное время, которое требуется для выполнения данного фрагмента кода, в данном контексте (входные данные, состояние), на заданном аппаратном вычислителе.
BCET – Best-case execution time
Цель анализа WCET – оценить сверху время выполнения фрагмента кода. Оценка должна быть:
Безопасной (недопустимо ошибаться в меньшую сторону)
Точной (завышенность приведёт к излишнему резервированию ресурсов системы)
Разумность затрат на анализ

Измерение WCET: запуск задачи -> {хронометр, анализатор шины, …} -> завершение задачи.
Измерение WCET не подходит, потому что:
1) Замер времени выполнения на всех путях выполнения реалистичной программы – на практике невозможен
2) При определении тестовой выборки могут быть упущены редкие сценарии выполнения (обработка ошибочных ситуаций и т.п.)
3) Выбранные тестовые данные могут не породить самую длинную (по времени) трассу выполнения
4) Внутреннее состояние процессора на момент старта измерений может не быть наихудшим
Простые замеры могут послужить источником первоначальной (грубой) нижней оценки WCET

Статический анализ WCET:
При статическом анализе WCET вычисляется верхняя оценка времени выполнения фрагментов кода
Моделируются аппаратные и программные средства, а также контекст выполнения

– Программные средства: исходный код, двоичный код (с привязкой к физическим адресам)
– Аппаратные средства: процессор (в т.ч. конвейер), память (в т.ч. кэш-память)
– Контекст: начальное состояние аппаратных и программных средств

Чем определяется WCET:
Возможные пути (последовательности действий) выполнения задачи, определяются:
- Семантикой программного кода (спецификой реализации, в т.ч. аппаратно-зависимой семантикой)
- Входными данными, возможными в данном контексте вызова программы
Длительность выполнения каждого действия на каждом возможном пути выполнения
- Аппаратной реализацией команд процессора
- Состоянием аппаратных средств, влияющих на тайминги (кэш-память, конвейер и т.п.) (как связанные с самой задачей, так и связанные с внешними факторами (состояние на момент старта, вытеснение задачи))

Длительность выполнения пути:
В простом случае - длительность каждого действия константа.
В реалистическом случае – длительности варьируются (конвейер, кэш-память, параллелизм процессора).

17. (Лекция 6а) Фазы анализа WCET. Анализ потоков. Оценка числа итераций циклов, выявление недопустимых путей. Использование абстрактной интерпретации. Аннотирование кода для поддержки анализа WCET - для простейшей и для реалистичной архитектуры процессора.

Фазы анализа WCET:
1) Анализ потоков: Ограничить (сверху) число выполнений различных фрагментов программы (в основном, анализ программной составляющей) (даёт верхнюю оценку, которая корректна для всех возможных трасс)
Примеры предоставляемой информации: ограничение на число итераций цикла, ограничение на глубину рекурсии, недопустимые пути выполнения.
Источники информации: статический анализ программы и ручные аннотации кода.
2) Низкоуровневый анализ: Ограничить (сверху) время выполнения различных фрагментов программы (сочетание анализа программной и аппаратной составляющих)
3) Вычисление: Объединить результаты анализа потоков и низкоуровневого анализа, чтобы получить верхнюю оценку WCET

Анализируется граф потока управления.
Фактически возможные пути <- статически допустимые (например, if(x<23){}) <- базовая ограниченность (например, число итераций цикла < 100) <- структурно допустимые пути (бесконечно много)

Ограничение числа итераций в общем случае неизвестно, но для многих частных случаев вычисляется.
Требование базовой ограниченности: для каждого цикла должно быть известно (вычислено или задано) ограничение на число итераций.
Недопустимые пути (например, найденные на основе if) исключаются из множества статически допустимых путей.

Абстрактная интерпретация (АИ – не artificial intelligence☹):
1) Ограничивает число итераций циклов и выявляет недопустимые пути
- Вычисляет безопасную (расширенную) оценку множества значений каждой переменной для различных точек выполнения программы
- В ходе АИ, переменной сопоставляется не конкретное значение, а множество значений («абстрактное» значение)
2) Программа «выполняется» с использованием абстрактных значений переменных
3) Результат выполнения: безопасная (расширенная) оценка множества допустимых путей выполнения
- Все фактически допустимые пути входят в полученное множество
- Также в него могут входить некоторые фактически НЕ допустимые пути
- Пути, не вошедшие в полученное множество, гарантированно не допустимы
Проблема треугольного цикла

Виды аннотаций кода:
Простейшая архитектура: сведения о частоте выполнения действий
- Границы и соотношения для частот выполнения
- Нотация: метки (marker), соотношения (relation), области (scope)
Сложная (реалистичная) архитектура: сведения о частоте выполнения последовательностей действий
- Информация о (не)допустимых путях
- Нотация: на основе регулярных выражений, например, IDL (path Information Description Language)

Проблема соответствия между исходным и двоичным кодом:
• Анализ потоков проще проводить на уровне исходного кода
- Более ясная семантика кода
- Проще получить потоковую информацию (и для программиста, и для автоматических инструментов)
• Низкоуровневый анализ работает с двоичным кодом, фактически выполняемым на процессоре
• Как отобразить потоковую информацию с уровня исходного кода на двоичный код?
• Компиляторы для встроенных систем реального времени выполняют глубокую оптимизацию кода
- Нужно уложиться в ограничения по времени и объему памяти
• Оптимизации могут существенно изменить размещение кода и данных
- В результате оптимизаций потоковая информация с уровня исходного кода становится неприменимой
• Решения:
- Реализовать в компиляторе средства отображения потоковой информации (недостижимый идеал)
- Использовать компиляцию с отладочной информацией (работает только при отсутствии или с минимумом оптимизаций)
- Для систем с большим объемом памяти – не производить оптимизации кода с целью сокращения объема
- Проводить потоковый анализ на двоичном коде (так чаще всего и делают)

18. (Лекция 6а) Фазы анализа WCET. Низкоуровневый анализ. Проблемы моделирования временных характеристик аппаратуры, важность предсказуемости задержек от аппаратуры. Анализ влияния конвейера, отражение результатов анализа на графе потока управления. Анализ влияния кэш-памяти. Актуальность совместного анализа влияния конвейера и кэш-памяти на время выполнения участков кода.

Фазы анализа WCET:
1) Анализ потоков: Ограничить (сверху) число выполнений различных фрагментов программы (в основном, анализ программной составляющей) (даёт верхнюю оценку, которая корректна для всех возможных трасс)
Примеры предоставляемой информации: ограничение на число итераций цикла, ограничение на глубину рекурсии, недопустимые пути выполнения.
Источники информации: статический анализ программы и ручные аннотации кода.
2) Низкоуровневый анализ: Ограничить (сверху) время выполнения различных фрагментов программы (сочетание анализа программной и аппаратной составляющих)
3) Вычисление: Объединить результаты анализа потоков и низкоуровневого анализа, чтобы получить верхнюю оценку WCET

Низкоуровневый анализ:
Цель - ограничить время выполнения различных фрагментов программы
- Основная задача большей части исследований по WCET
Использовать модель целевой аппаратуры
- Не требуется моделировать все подробности работы аппаратуры
- При этом необходимо безопасно (сверху) оценить все задержки при работе аппаратуры
Применяется к скомпонованному двоичному коду (исполняемой программе)

Проблемы моделирования аппаратуры:
Высокая сложность моделирования внутренней работы процессора
- Конвейер, предсказание ветвлений, суперскалярность, внеочередное выполнение…

Высокая сложность моделирования иерархической памяти
- Необходимо детальное моделирование кэш-памяти
- Другие виды памяти также являются источником задержек
Многие аспекты функционирования сложных процессоров должны моделироваться совместно
- Время выполнения команд процессора зависят от предыстории
Разработка безопасной временной модели функционирования процессора – сложная задача
- Занимает месяцы и даже годы работы
- Необходимо учесть все факторы, влияющие на время выполнения (как минимум, оценить сверху их влияние)
Поэтому аппаратура с предсказуемым временем работы важнее, чем очень быстрая аппаратура.

Простой конвейер:
Instruction fetch (выборка команды) -> instruction decode (декодирование команды) -> execution -> memory access (загрузить/ сохранить значения в/из памяти) -> write back (запись результата)
В идеале конвейер ускоряет во столько раз, сколько у него ступеней, однако из-за зависимостей по данным это не так.
Виды конвейеров:
Отсутствует / скалярный (один конвейер) / VLIW (несколько конвейеров, статическое планирование из загрузки на уровне компилятора) / Суперскалярный (несколько конвейеров, внеочередное выполнение команд (процессор может на ходу переставить порядок команд))

Чтобы учесть задержки в случае простого конвейера – в графе потока управления ребрам присваиваются отрицательные веса.

Иерархическая память:
cpu <- кеш <- основная память
кеш обычно иерархичен + может быть отдельный по данным и командам.

При анализе кеш промахов анализируется двоичный код.

Учет совместного влияния кэша и конвейера:
Анализ влияния конвейера должен брать на вход результаты анализа влияния кэш-памяти
- Команды помечаются попаданием/промахом в кэш
- Попадания/промахи влияют на задержки в конвейере
Сложная аппаратура требует совместного анализа влияния кэша и конвейера

19. (Лекция 6а) Фазы анализа WCET. Вычисление оценки WCET. Методы расчета WCET: по синтаксическому дереву программы; по путям выполнения; метод неявного перебора путей.

Фазы анализа WCET:
1) Анализ потоков: Ограничить (сверху) число выполнений различных фрагментов программы (в основном, анализ программной составляющей) (даёт верхнюю оценку, которая корректна для всех возможных трасс)
Примеры предоставляемой информации: ограничение на число итераций цикла, ограничение на глубину рекурсии, недопустимые пути выполнения.
Источники информации: статический анализ программы и ручные аннотации кода.
2) Низкоуровневый анализ: Ограничить (сверху) время выполнения различных фрагментов программы (сочетание анализа программной и аппаратной составляющих)
3) Вычисление: Объединить результаты анализа потоков и низкоуровневого анализа, чтобы получить верхнюю оценку WCET

Вычисление WCET: исходные данные: информация о задержках и потоковая информация
Примеры подходов:
• Расчёт по синтаксическому дереву (дерево обходится снизу-вверх)
- Фиксированные времена выполнения узлов
- Времена выполнения листьев известны

- Времена выполнения внутренних узлов рассчитываются по формулам для типов узлов:

Для оператора ветвления: берем максимум из значений для узлов-потомков и добавляем время на проверку условия

Для цикла: Суммируем оценки для узлов-потомков и умножаем на оценку числа итераций цикла

• Расчёт по путям выполнения

- Без конвейера:

Надо подготовить циклы, убрав обратные дуги и перенаправив их на специальные узлы «continue»

Находим самый длинный путь, рассматривая итерации цикла по одной, если путь – невозможен, то ищем другой самый длинный путь.

- С конвейером:

Упорядочиваем все пути по убыванию «грубой» оценки WCET (без конвейерной экономии)

Для каждого пути вычисляем его WCET с учётом конвейера, и если значение окажется больше, чем грубые WCET для всех оставшихся путей, или если других путей не осталось больше, то (по-моему, у них тут не правильно !!!) нужно из всех рассмотренных WCET с учётом конвейера нужно выбрать тот, что максимален. (слайд 321)

• Неявный перебор путей (IPET)

- Пути выполнения не обрабатываются в явном виде.

X-entity – число выполнений, t-entity – информация о задержке. WCET = max(sum(X-entity * t-entity)),

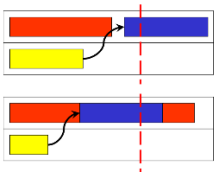
Причём X-entity удовлетворяет ограничениям: начальным и конечным условиям, структура программы, ограничения на число итераций, прочая потоковая информация. (из графа потока управления вынимаются ограничения и уравнения, т.е. фактически всё предстаёт огромной системой уранений)

Систему ограничений (уравнения) – можно решить целочисленным линейным программированием или разрешением ограничений (метод constraint satisfaction).

В результате получаем число выполнений узлов и дуг и оценку WCET.

20. (Лекция 6б) Критичность временной предсказуемости функционирования ИУС РВ. Критерии производительности для систем реального времени и «обычных» вычислительных систем. Линеаризация кода. Предикатное выполнение команд и его использование для линеаризации кода. Свойства линейного кода (с точки зрения сложности анализа и производительности). Обеспечение константного времени выполнения линейного кода. Общая схема оптимизации WCET на этапе компиляции. Проблема изменения наихудшего пути в результате оптимизации.

Важен не только WCET, но и BCET:



Критерий производительности для конкретной задачи:

Традиционный: минимизация sum по всем входным данным от (<вероятность входных i> * <время выполнения конкретной задачи на входных данных i на конкретной машине>)

HRT: минимизация max (<время выполнения данной задачи при всех возможных данных на конкретном железе>)

Традиционное программирование:

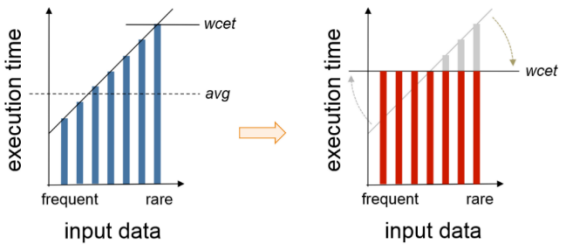
- Цель: хорошая производительность в среднем.
- Стратегия: профилировать и оптимизировать производительность на наиболее часто встречающихся наборах входных данных => возможное ухудшение WCET.

Причина: сложность неравномерно распределена между наборами входных данных; оптимизация частых сценариев приводит к ухудшению производительности на редких сценариях.

WCET-ориентированное программирование:

- Стремимся освободить код от ветвлений в зависимости от входных данных.
- Минимизируем число действий, выполняемых только для отдельных наборов входных данных.

Традиционное vs. WCET-ориентированное программирование:



Алгоритм	Традиционное		WCET-ориентированное	
	Среднее	WCET	Среднее	WCET
Bubble*	599	724	609	663
Find-first*	68	122	103	103
Bin-search	94	124	105	106

* В традиционном варианте: без goto; используется признак раннего завершения

Свойства WCET-ориентированных программ:
- WCET меньше, чем у традиционных программ
- Анализ путей проще в связи с устранением/сокращением зависимостей от входных данных (сокращение ветвлений по входным данным)
- Оценка WCET проще из-за меньшего числа путей выполнения (или единственного пути)
- Меньше разброс времен выполнения => более стабильна работа системы в целом
- Низкая сложность потока управления (акцент на потоке данных, а не на потоке управления) => хорошо подходит для управляющих программ, критичных для безопасности
- Нетрадиционные алгоритмы
- Предикатное выполнение (управление с предсказаниями)

Следствия WCET-ориентированного программирования и линеаризации кода:
- Константное время выполнения и низкий WCET
- Тривиальный анализ путей и простой анализ WCET

=> полная временная предсказуемость при приемлемой производительности

Линеаризация: код, оптимизированный под средний случай, замедляется сильнее.

Линеаризация (получение линейного кода):
- Устранить зависимости по данным из потока управления

Предикатное управление: это условное выполнение или невыполнение команды в зависимости от значения булевой переменой, называемой предикатом.

Процессор безусловно выбирает все команды из памяти, если предикат истинен, то нормальное выполнение команды, если предикат ложен, то команда не изменяет состояние процессора.

Аппаратная поддержка предикатного выполнения:

Предикатные регистры

Команды для работы с предикатами (определить, установить, сбросить, загрузить, сохранить)

Поддержка полной/частичной предикатности: выполнение любых/некоторых (напр., только копирование данных или присваивание значений) команд управляется предикатами.

Специфика частичной предикатности:

Опережающее выполнение вычислений
- Команды, не являющиеся предикатными, выполняются безусловно
- Их результаты сохраняются во временных переменных
- В дальнейшем предикат определят, содержание какой из временных переменных следует использовать

Внимание: команды, вычисляющие значения временных переменных, не должны генерировать исключения (пример: деление на ноль, обращение к недопустимому адресу памяти).

Минимальная поддержка предикатности: movCC dst, src – if CC then dst:=src

Эмуляция условного копирования: можно использовать битовые маски.

Свойства линейного кода:

Каждое выполнение порождает одну и ту же трассу команд, т.е. одну и ту же последовательность обращений к памяти команд. => получаем фиксированное время выполнения

Анализ путей тривиален – существует единственный путь.

Два выполнения, начинающиеся с одинакового состояния кэша команд, порождают одинаковые последовательности промахов/попаданий в кэш команд.

Источники разброса времени выполнения:
Команды с неконстантным, зависящим от данных, временем выполнения вызывают флуктуации времени выполнения кода.
Разные начальные состояния памяти (в т.ч. кэша) могут привести к различным задержкам при доступе к памяти команд и данных, а значит к различным временам выполнения кода.

Обеспечение константного времени выполнения:
Не позволяем внешним факторам влиять на последовательность выполняемых команд и длительность команд
Всегда начинать с одного и того же состояния кэша команд, конвейера, логики предсказания ветвлений и т.п.
Обеспечивать константные задержки при доступе к данным
Обеспечивать константные длительности всех операций процессора
Все внешние воздействия (в т.ч. вытеснение задач) должны быть предсказуемыми
Что делать:
• Сбрасывать кэши при старте задач
• Размещать данные по фиксированным адресам
• Избегать сложного вычисления адресов

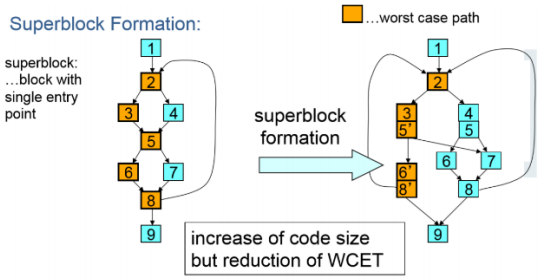
Константные длительности операций процессора:
- Все операции процессора должны быть реализованы так, чтобы выполняться за константное время, независимо от значений операндов (пример: целочисленный сумматор всегда отрабатывает за 1 такт)
- В частности, предикатные команды должны выполняться за константное время. Если предикат ложен, команда выполняется, но её результаты отбрасываются и не изменяют состояние процессора.
- У циклически выполняемых команд в коде должны быть константные значения числа итераций.

Производительность линейного кода:
Время выполнения линейного кода сильно дольше, по сравнению с ветвистым кодом.
Отсутствуют переходы, поэтому конвейер никогда не сламывается (в некоторых ситуациях это приводит к ускорению программы для линейного предикатного кода).

Свойства процедуры линеаризации кода:
- Полнота: любой участок кода с ограниченным WCET может быть линеаризован
- В линеаризованном коде путь выполнения – единственный
- Анализ WCET тривиален: запустить код и измерить время (при «наихудшем» начальном состоянии системы)
- Код выполняется заметно дольше ветвящегося кода

Оптимизация WCET на этапе компляции:
- находим наихудший путь
- выбираем соответствующие оптимизации
- применяем оптимизации к соответствующим частям кода
В результате оптимизации другой путь может стать наихудшим, причём даже может стать ещё хуже, чем был исходно модифицируемый путь.

Пример изменения WCET:



21. (Лекция 6б) Измерение WCET: в каких случаях это допустимо? Схема оценки WCET с помощью измерений, основные методы инструментирования систем для оценки WCET. Оценка

WCET как оптимизационная задача. Применение эволюционных алгоритмов для оценки WCET. Безопасность получаемых оценок.

Можно ли замерять WCET?:
- Замер времени выполнения на всех путях выполнения реалистичной программы – на практике невозможен
- При определении тестовой выборки могут быть упущены редкие сценарии выполнения (обработка ошибочных ситуаций и т.п.)
- Выбранные тестовые данные могут не породить самую длинную (по времени) трассу выполнения
- Внутреннее состояние процессора на момент старта измерений может не быть наихудшим

С другой стороны:
Не во всех случаях строго необходима безопасная (не заниженная) оценка WCET
- Системы мягкого реального времени (например, мультимедиа)
- Системы, устойчивые к редким превышениям директивных сроков

Для новой аппаратной платформы быстрее всего можно начать именно замеры времени выполнения (а статические методы анализа аппаратных задержек адаптировать долго)
Низкие затраты на аннотирование кода => быстрое получение грубой оценки времени выполнения
Замеры дополняют статический анализ, предоставляя реальные значения задержек
Замеры могут использоваться для уточнения результатов статического анализа WCET
- Моделировать некоторые современные процессоры действительно сложно
- Для некоторых задач зависимость времени выполнения от входных данных по-настоящему сложна (вещественные числа, массивы и т.п.)
- Люди из промышленности требуют подтверждения теоретических оценок WCET данными «из жизни»

Оценка WCET с помощью измерений:
- Информация о таймингах получается путём измерения времени выполнения кода на реальной целевой аппаратуре
- Точки инструментирования кода формируют наблюдаемые извне события, используемые для старта и завершения измерений

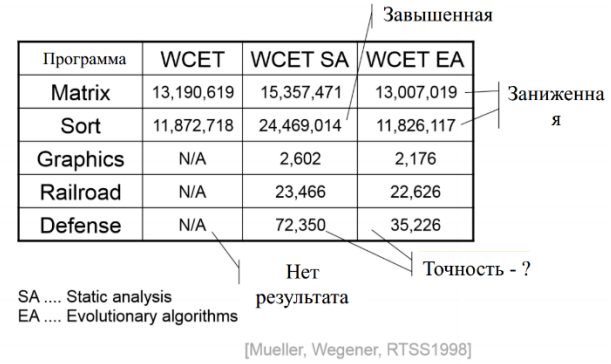
Методы инструментирования:
- Чисто аппаратное инструментирование
- Внешние замеры времени выполнения при помощи программно формируемых сигналов вовне
- Чисто программное (внутреннее) инструментирование

- Трасса выполнения содержит собираемую совместно информацию о путях в программе и временах их выполнения (путь = последовательность линейных участков)

Важные соображения по замерам. Как измерить именно то, что нужно:
- Инструментирование кода не должно изменять путь выполнения или время выполнения программы непредсказуемым или неизвестным способом. Точки инструментирования должны располагаться в фиксированных местах кода.
- Нужна уверенность в том, что запуски для замеров всегда стартуют с известного требуемого состояния процессора (кэш, конвейер, предсказание ветвлений и т.п.)

Поиск «наихудших» входных данных – это оптимизационная задача:
Массивы входных данных – ограниченный перебор.
Случайные входные данные – случайный поиск.
EA – Evolutionary Algorithm:
ген – набор свойств индивида
индивид – вектор генов
популяция – множество индивидов
отбор – вероятность выживания абстрактного индивида (выполняется с учётом значения целевой функции)
скрещивание – обмен генами индивидов (например, одно- или n-точечное скрещивание)
мутация – случайное изменение гена индивида
обычно происходит в порядке: отбор -> скрещивание -> мутация

Для WCET и эволюционного алгоритма:
Ген – значение переменной (входной/внутренней), целевая функция – измеренное время выполнения, результат – особь с максимальным выполнением.
Даёт хорошую, но небезопасную оценку WCET.



Программа	WCET	WCET SA	WCET EA
Matrix	13,190,619	15,357,471	13,007,019
Sort	11,872,718	24,469,014	11,826,117
Graphics	N/A	2,602	2,176
Railroad	N/A	23,466	22,626
Defense	N/A	72,350	35,226

Выводы об оценке WCET:
Оценка WCET – сложная задача, разрешимая (безопасно и с приемлемой точностью) только для некоторых типов процессоров и программ
На практике для оценки WCET применяется сочетание формальных методов и измерений
Система реального времени должна быть устойчива к отдельным/локальным превышениям имеющихся оценок WCET
– сторожевые таймеры для обнаружения превышения WCET
– устойчивость к сбросу задачи-нарушителя (и пропуску итерации её выполнения)
– «растяжимые» периоды выполнения задач для динамической регулировки загрузки процессора

22. (Лекция 7) Технологические ограничения на вычислительные блоки ИУС РВ, источники этих ограничений. Характеристики однопроцессорных центральных ЭВМ на примере марсоходов. Мезонинная архитектура одноплатных компьютеров. Пример системы из однопроцессорных блоков со слабой интеграцией.

Однопроцессорные блоки:
• Технологические ограничения:
– вынужденное применение «грубого» технологического процесса
– жёсткие ограничения по размерам и энергопотреблению
• Откуда берутся ограничения
– требование к устойчивости к внешним воздействиям (излучение и т.п.)
– неразвитость технологических процессов производства микросхем
– общий лимит на размеры и энергопотребление ИУС РВ

Характеристики однопроцессорных центральных ЭВМ на примере марсоходов:
EEPROM - Electrically Erasable Programmable Read-Only Memory - электрически стираемое перепрограммируемое ПЗУ

Аппарат	CPU	RAM	Flash	EEPROM	ОС
Sojourner (1997)	Intel 80C85, 2 МГц, 8-разрядный, 6000 транзисторов (аналог Intel 8080 1974 г. разработки)	512 Кбайт	176 Кбайт	Нет	Однозадачная, жёсткий порядок выполнения задач
Mars Exploration Rover (2004)	IBM/BAE RAD6000, 20 МГц, 32-х разрядный, 1.1 млн транзисторов	128 Мбайт	256 Мбайт	3 Мбайт	VxWorks, многозадачная
Mars Scientific Laboratory (2011)	BAE RAD750, 132 МГц, 10.4 млн транзисторов	256 Мбайт	2 Гбайт	256 Кбайт	VxWorks, многозадачная

Расширение одноплатного компьютера: карта-мезонин

• На одноплатном компьютере – локальная шина (например, PCI)

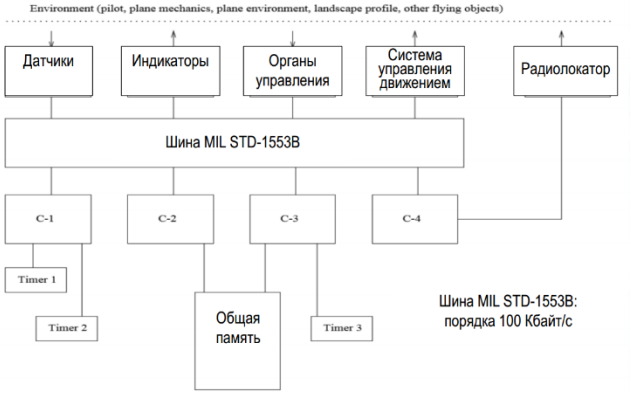
• К локальной шине присоединяется карта расширения (например, адаптер канала)

• Карта расширения не зависит от «основного» разъёма, через который компьютер подключается к управляемой системе

Внедрения RAD750: Deep Impact (полёт к комете), Mars Reconnaissance Orbiter (спутник Марса), Lunar Reconnaissance Orbiter (спутник Луны), WorldView-1 (спутник оптической съёмки), Fermi Gamma-ray Space Telescope (орбитальный телескоп), Kepler space telescope (орбитальный телескоп), Wide-field Infrared Survey Explorer (орбитальный телескоп), Mars Scientific Laboratory (марсоход).

Слабая интеграция, однопроцессорные блоки на медленной шине:

Environment (pilot, plane mechanics, plane environment, landscape profile, other flying objects)



Назначение блоков:

• C-1: контроль состояния бортовых систем, выбор режима работы ИУС РВ, управление обменом по шине (контроллер канала)

• C-2: вычисление управляющих параметров полёта для передачи в систему управления движением, подготовка данных для индикаторов

• C-3: вычисление параметров движения самолёта на основе показаний датчиков, обеспечение движения самолёта по маршруту, управление датчиками

• C-4: управление полётом на малой высоте по показаниям радара

23. (Лекция 7) Шина VME. Роли модулей на шине VME. Процедура передачи данных по шине VME. Механизмы прерываний и блочной передачи данных на шине VME. Недостатки шины VME. Стандарт VPX как путь к устранению этих недостатков.

Тесная интеграция: шина VME:

• Параллельная шина с арбитражем

• Реализует прямой доступ к памяти модулей

• Объединяет модули в блок (крейт)

• Разрядность шины данных: 32 или 64 бита

• Пропускная способность:

– 40 Мбайт/с (VME32)

– 80 Мбайт/с (VME64)

– до 320 Мбайт/с (VME64 в блочном режиме, на одну передачу адреса – несколько передач данных)

VME - Адресная шина данных с арбитражем и прерываниями
(напр., использовалась как процессорная шина Motorola 68000)

Роли модулей на шине VME:

Ведущий (Master) - может инициировать передачу данных

Подчинённый (Slave) - отвечает на запросы от ведущего

Источник прерывания (Interrupter) - модуль, способный формировать прерывание (обычно – подчинённый)

Обработчик прерывания (Interrupt handler) - модуль, способный обрабатывать прерывания (как правило, одноплатный компьютер)
Арбитр (Arbiter) - модуль, управляющий доступом к шине и осуществляющий мониторинг обмена по шине. Устанавливается в слот №1

Процедура передачи данных на VME:
1) Ведущий устанавливает запрос на передачу данных на шине арбитража (Ш.А.)
– при этом ведущий устанавливает на Ш.А. «свою» линию запроса в активное состояние (логический 0)
2) Шина освобождается от текущей передачи данных =>
– арбитр определяет, какие линии запроса активны на Ш.А.
– арбитр выбирает ведущего, которому отдать шину, и устанавливает в активное состояние линию Ш.А. «доступ дан» для этого ведущего
3) Получив доступ к шине, ведущий устанавливает:
– на шине данных: значения передаваемых данных (в случае отправки); разрядность данных – не больше разрядности шины данных
– на шине адреса: номер подчинённого устройства, адрес в памяти подчинённого устройства, разрядность передаваемых данных (8, 16, 32 бита; также 64 бита для VME64), признак «чтение» или «запись»
4) Подчинённое устройство:
– на Ш.А. признак «чтение» => устанавливает на шине данных значения данных заданной разрядности с заданного адреса своей памяти
– на Ш.А. признак «запись» => записывает по заданному адресу своей памяти данные заданной разрядности с шины данных

Прерывания VME:
• Прерывание – способ для подчинённого устройства оповестить какое-либо из ведущих устройств о необходимости обмена данными
• Запрос прерывания выставляется на одной из 7 линий шины прерывания
• Ведущие устройства сами разбираются, кому адресован запрос

Блочная передача данных:
• Поддерживается в VME64
• В начале обмена задаётся адрес и число передаваемых блоков данных
• Выполняется передача заданного числа блоков данных (каждый блок - не шире шины данных)
=> значительная экономия времени на задании адреса

Недостатки VME:
• Медленная шина (по современным меркам)
• Параллельная шина (много линий на материнской плате, сложность повышения частоты работы)
• Невозможно одновременное выполнение нескольких обменов данными

Борьба с недостатками VME: VPX:
• Программа «VME Renaissance»: дать VME будущее в мире скоростных процессоров
• Обратная совместимость с VME: возможность установки в VPX-систему существующих VME-модулей
– механическая совместимость
– поддержка протокола и «идеологии» VME
• Уход от общей шины с арбитражем: поддержка коммутируемых сетей
– Gigabit Ethernet, 10-Gigabit Ethernet
– PCI Express
– InfiniBand (высокоскоростной прямой доступ в память)
(Высокоскоростные каналы из мира x86)

VPX: плавный уход от VME:
• Современная VPX-система:
– нет VME (т.к. нет унаследованных модулей)

– x86 процессор
– технологии из настольных систем (PCI Express, Gigabit Ethernet, USB, HDMI/DisplayPort)
– межмодульное взаимодействие – по каналам Ethernet (дорожки на материнской плате крейта)
• Фактически – вычислительный кластер на технологиях «мира x86»
– для создания системы нужно меньше экзотических знаний
• Модули в закрытых кожухах – можно вставлять и вынимать из работающей «в поле» системы

24. (Лекция 7) Интегрированная модульная авионика (ИМА). Архитектура систем ИМА, преимущества этой архитектуры. Шина данных и сервисная шина в системах ИМА. Примеры модулей в системах ИМА.

ИМА - Интегрированная, модульная архитектура: логически единый распределённый вычислитель (единая архитектура, унифицированные модули, унифицированные программные интерфейсы), разделение ресурсов между ПО различных подсистем.
Проблемы: конкуренция за процессорное время, изоляция по памяти.
Для ИМА характерно:
Стандартное API со стороны ОС.
Статическое разделение времени, памяти и ресурсов.
Преимущества: надёжно, переносимо, возможность повторного использования, модульность, упрощение верификации и сертификации.

Интегрированная модульная авионика:
• Недостаток традиционных многомодульных блоков: чрезмерная внутренняя интеграция
• VME: шина ограничена пределами блока
– Единственный арбитр
– Параллельная шина (сложно провести между блоками, в т.ч. защитить от помех)
– Выход арбитра из строя – фатален (слот №1 - единственный)
– Идеология прямого доступа к памяти: у модуля «слишком много» знания о внутреннем устройстве других модулей того же блока
• В итоге:
– система неоднородна (блоки сильно различаются)
– модули одного блока тесно интегрированы друг с другом
– модули разных блоков слишком изолированы
– низкая отказоустойчивость и реконфигурируемость (блок выходит из строя целиком)

• Среда обмена данными на основе коммутатора
– использование виртуальных каналов для разделения пропускной способности
– ограниченные задержки при передаче сообщений
• Модули всех блоков равноправно подключены к среде обмена данными => система – «облако» модулей
• Сервисная шина – последовательная, относительно медленная (помехоустойчивость), может объединять все модули системы ИМА
– CAN bus - Controller Area Network – сеть контроллеров
• Трафик по сервисной шине минимален (низкоуровневые данные о состоянии, простейшие команды вроде вкл/выкл модуля)
• Высокая отказоустойчивость и реконфигурируемость
– модуль вышел из строя => заменить его может модуль из другого блока
– поддержка виртуальных каналов => возможность миграции вычислительных задач

Интегрированная модульная авионика (5 поколение)
• Унификация: вычислительные модули, сетевое оборудование и протоколы
• Интеграция: программное обеспечение, потоки данных
• Виртуализация: процессоры, память, сеть

Сервисная шина CAN:
• Абонент CAN на модуле ИМА – отдельная подсистема
– включается до активизации основной логики модуля
– продолжает функционировать при сбое основной логики модуля
• Короткие сообщения (до 8 байт данных)
– периодическая выдача модулями статусных сообщений
– запрос статуса модуля + ответ на запрос
– команды вкл/выкл основной логики модуля
• Арбитраж
– каждый абонент «слышит» передачу данных по каналу
– нет передачи данных => можно начинать отправку своих данных; иначе ждать освобождения канала
– при одновременном начале выдачи: доминантные и рецессивные биты, сообщение начинается с адреса абонента-отправителя, абонент с «меньшим» адресом слышит в сети не то, что выдал, и прекращает выдачу
• Как ограничить задержку доставки данных?
– ИМА: низкая загрузка канала, нет проблемы
– CAN с централизованным управлением

Примеры модулей ИМА:
1) Модуль процессора данных - функциональный модуль общего назначения
Задачи: обработка данных, выполнение вычислительных задач, принятие управленческих решений.
2) Модуль ввода-вывода - функциональный модуль специального назначения
Задачи: прием/выдача сигналов по «унаследованным» бортовым интерфейсам, преобразование унаследованных форматов сообщений в стандартный формат ARINC 653.
3) Модуль графического контроллера - функциональный модуль специального назначения
Задачи: построение изображений на основе данных, полученных от вычислительных задач, обработка входных видеоизображений, прием/выдача изображений по оптическим видеоканалам.
4) Модуль коммутатора FC - предназначен для обеспечения взаимодействия в сети Fibre Channel
5) Модуль источника питания - вспомогательный модуль, обеспечивающий вторичное питание модулей с требуемыми характеристиками.

25. (Лекция 8) Схема функционирования канала с централизованным управлением и роли устройств на нём. Преимущества схемы с централизованным управлением. Канал MIL STD-1553B и его использование на Международной космической станции. Эволюция стандарта MIL STD- 1553B: каналы EBR-1553, MIL STD-1760, STANAG 3910. Организация обмена с централизованным управлением на шине CAN.

Состав ИУС РВ: регистраторы, вычислители, датчики, эффекторы, интерфейс оператора (индикаторы, органы управления), бортовая сеть.

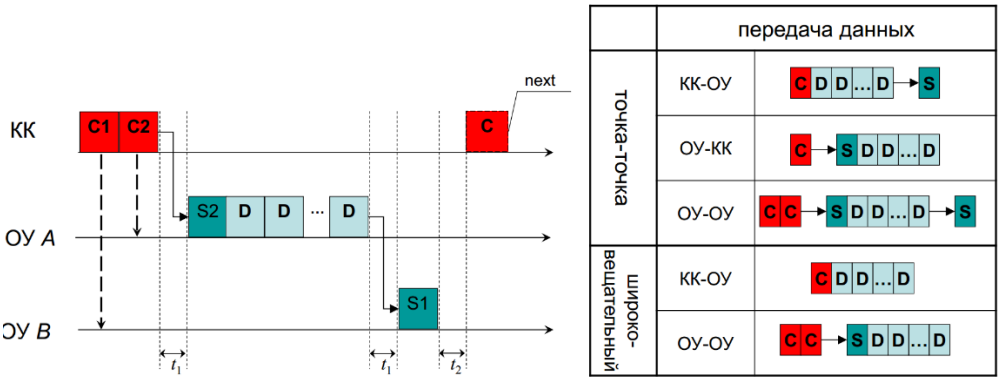
Канал с централизованным управлением:
Контроллер – управляет обменами в соответствии с предварительно построенным расписанием обменов.
Информация передается в виде сообщений.
Обмен информацией осуществляется путем поочередной передачи данных по принципу "команда-ответ".

Преимущества каналов с централизованным управлением:
Отсутствие конфликтов.
Гарантированная передача сообщений в режиме реального времени.
Минимальное количество «проводов» в сети обмена.

Промышленные каналы с централизованным управлением:
• MIL STD-1553B
• Модификации MIL STD-1553B:
– MIL STD-1773 (оптика)

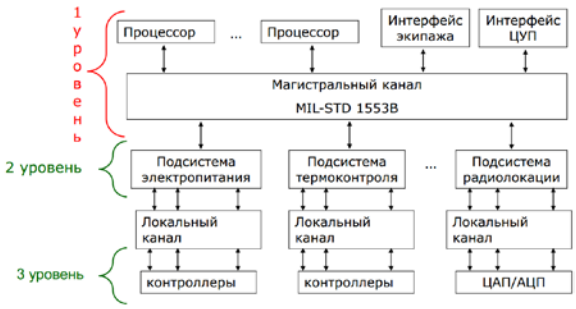
– Space Shuttle MIA bus (более длинное слово)
– EBR-1553 (топология «звезда»)
• MIL STD-1760 (иерархия)
• STANAG 3910
• FC-AE-1553
• CAN bus с «искусственной» централизацией

MIL STD-1553B: <u>передача сообщения и формат сообщений</u>:



C – командное слово
D – слово данных
S – ответное слово
t_1 – пауза перед выдачей ответного слова
t_2 – пауза между сообщениями

Иерархическая организация ИУС РВ МКС:



Режимы работы ИУС на МКС:
• стандартный режим
• режим микрогравитации для выполнения научных экспериментов;
• режим сближения и стыковки с транспортными кораблями;
• режим для выхода экипажа в открытый космос;
• режим выживания с отключением наименее важных экспериментов и систем;
• режим аварийного покидания экипажем МКС.
Режим - это набор сообщений, расписание передачи.

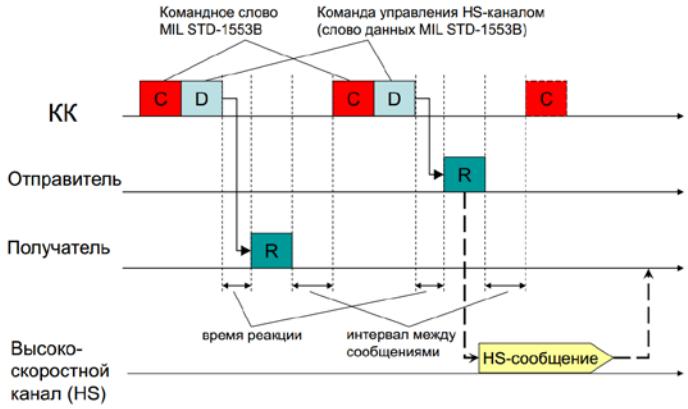
«Фокусы» с каналом MIL STD-1553B
• Ракета-носитель ARIANE 5: обнаружение разделения ступеней (разрыв магистрали)
• Транспортный корабль ATV: индикация успешной стыковки (соединение периферийного канала)

EBR-1553 (Enhanced Bit Rate):
• Развитие MIL STD-1553B
• Новые возможности: скорость 10 Мбит/с, топология «звезда» (контроллер в середине), контроллер объединен с сетевым концентратором (hub)
• Нет обменов ОУ-ОУ

MIL STD-1760 (иерархическая):
• Расширяет MIL STD-1553B для поддержки бортовых хранилищ данных

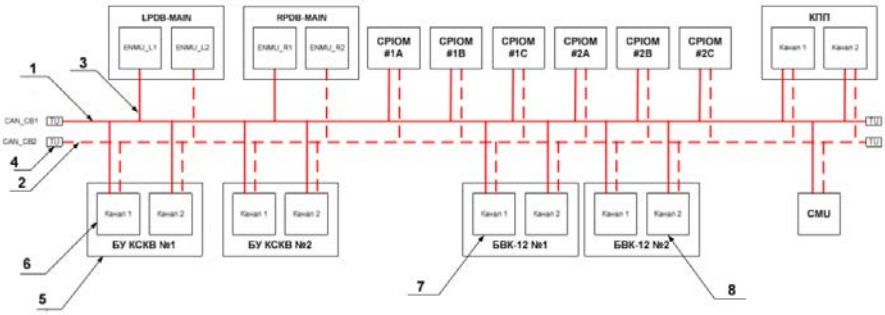
• Определяет стандартные форматы сообщений для функций управления хранилищами
• Акцент на надежности:
– резервированная шина
– контрольные суммы (CRC) в заголовках сообщений
– специальные форматы сообщений для контроля целостности данных
– подтверждение получения критичных управляющих сообщений

STANAG 3910 (топология):
КК и ОУ подключены одной шиной MIL STD-1553 (Управляющие команды и низкоскоростная передача данных (1 Мбит/с))
И подключены топологией звезды чем-то высокоскоростным (оптоволокно, например), для передачи данных (20 Мбит/с)
Обмен данными:



Шина CAN – служебная шина, короткие сообщения.
Арбитраж шины CAN:
• Каждый абонент «слышит» передачу данных
• Нет передачи данных => можно начинать отправку своих данных
• Конфликт => абонент с «меньшим» адресом отступает
• Проблема: как ограничить задержку доставки данных?

CAN с центральным управлением:



1) Основная магистральная шина
2) Резервная магистральная шина
3) Ответвитель основной магистральной шины
4) Согласующий резистор
5) Абонент интерфейса (БУ КСКВ №1)
6) Устройство интерфейса
7) Контроллер основной шины/Резервный контроллер резервной шины
8) Контроллер резервной шины/Резервный контроллер основной шины

26. (Лекция 8) Задача построения расписания выполнения работ в одноприборном устройстве. Задача построения расписания передачи сообщений по шине с централизованным управлением. Технологические ограничения на обмен для схемы с подциклами и схемы без

подциклов. Жадный алгоритм построения расписания передачи сообщений, основные недостатки этого алгоритма. САПР циклограмм: основные функции, схема процесса применения.

Задача построения расписания выполнения работ в одноприборном устройстве:
Шина (канал) с централизованным управлением может рассматриваться как одноприборное устройство, обслуживающее исходно заданный набор работ без прерываний.
Расписание – это упорядоченный набор работ, суть в том, что нам даны работы с их временами выпуска, директивным сроком и временем выполнения, а расписание H содержит последовательность работ, для которых задано начало, когда они будут выполняться, и конец, когда работа перестанет выполняться.
Ограничения на множество корректных расписаний:
Директивные сроки выполнения работы в нашем расписании должны находиться в рамках допустимых директивных сроков работы по условию.
Время выполнения работы в расписании должно ***быть равно*** времени выполнения этой работы по условию
Времена выполнения работ в расписании не должны пересекаться.
(на слайдах - формулы)
Задача: максимизировать время выполнения расписания (учитывая, что выше есть «быть равно», то максимизация времени расписания – это увеличение количества интервалов (т.е. запаса) между задачами)
!!! Не уверен, что здесь прав, возможно имеется ввиду, что в расписание все задачи не влазят, и наша цель – распределить время наиболее продуктивно.
В теории расписания задача называется «Задача о выборе максимального числа совместимых заявок» и является NP-трудной.
Частный случай: для частной задачи, где $t_j = f_j - s_j$, т.е. если в исходных задача директивные сроки выполнения как раз обрамляют время выполнения задачи. Существует оптимальный жадный алгоритм сложности O(n * log n)

Преобразование сообщений:
Сообщение m (F, f1, f2, t) (частота, сдвиг от времени выпуска, время завершения после времени выпуска, время передачи).
Дано:
множество работ, которые должны выполняться на системе J
Технологические ограничения на корректность расписания
Вектор параметров технологических ограничений

Есть период (каждый период нужно выполнить работы), в периоде могут выделяться подцикл (один), который должен заканчиваться резервным временем.

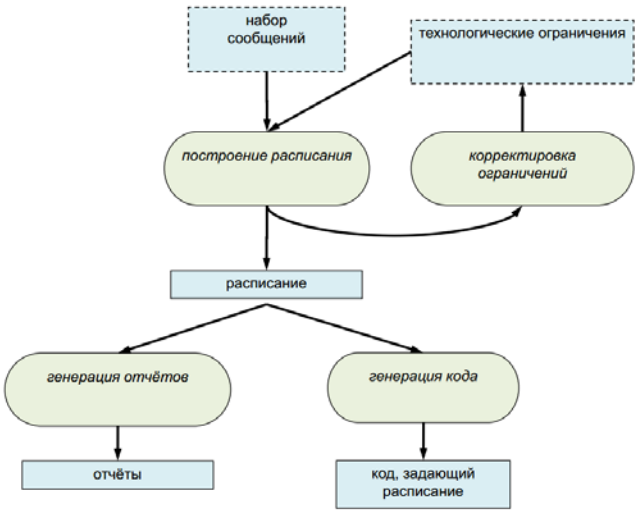
(!!! надо попросить преподавателя побольше рассказать про 450 слайд)
Пример технологических ограничений на расписания передачи сообщений по шине:
- *Одна цепочка работ в подцикле*
- *Резерв времени в конце подцикла*
- *Максимальная длительность цепочки работ*
- *Максимальное отклонение расстояния между последовательными работами одного сообщения от периода сообщения*

Ограничения для схемы с подциклами:
g4 - в каждом подцикле может находиться не более одной цепочки работ.
g5 - интервалы выполнения работ не должны пересекать границы подцикла.
g6 - время начала цепочки работ относительно начала подцикла не должно быть меньше заданного значения.
g7 - в конце подцикла должен быть зарезервирован интервал времени не меньше заданной длительности.
g8 - число работ в цепочке не должно превышать заданного значения.
Ограничения для схемы без подциклов:
g4 - число работ в цепочке не должно превышать заданного значения.
g5 - суммарная длительность выполнения работ цепочки не должна превышать заданного значения.
g6 - интервал времени между последовательными цепочками должен быть не меньше заданного значения.

Жадный алгоритм построения расписания:
t: самое раннее время, на которое допустимо планировать работы, t монотонно возрастает.
1) t := 0
2) выбрать, в соответствии с детерминированным критерием CR (например, EDF), работу, которая может быть начата в момент t без нарушения ограничений
нет такой работы => t := <минимальное значение, для которого такая работа существует>; перейти к шагу 5
3) поместить выбранную работу в расписание, с временем начала t
4) t := <время завершения выбранной работы>
5) учесть технологические ограничения
- обновить значение t
- обновить директивные интервалы оставшихся работ (начало >= t)
6) директивные интервалы некоторых работ короче самих работ => внести эти работы в список исключенных
7) есть работы, не исключенные и не вошедшие в расписание => перейти к шагу 2

Недостатки жадного алгоритма:
- Критерий выбора работ должен быть определен заранее
- Для каждого критерия (RM, EDF, …) существуют наборы работ, на которых он дает плохой результат => ручной выбор критерия или перебор критериев
- Нет «обучения» по результатам неуспешных запусков => если алгоритм неуспешен на конкретных данных, он всегда будет неуспешен на них при том же критерии CR
Возможное решение: адаптивный алгоритм с элементами недетерминизма (например, муравьиный)

САПР циклограмм (система автоматизированного проектирования циклограмм):
1) Создание проекта: наполнение базы данных информацией о структуре бортовой сети и протоколах информационного взаимодействия.
2) Автоматическое построение расписания.
3) Возможность ручной корректировки расписания.
4) В случае, если нельзя построить полное и корректное расписание: автоматическая корректировка значений ограничений на расписание.
5) Генерация программного кода, задающего расписание.
6) Генерация отчётов о входных данных и построенных расписаниях.
Диаграмма тех.процесса:



27. (Лекция 8) Кольцо с арбитражем Fibre Channel, схема его функционирования. Процедура арбитража. Протокол FC-AE-1553 и его использование для работы унаследованных устройств, поддерживающих протокол MIL STD-1553B.

Топологии и классы обслуживания стандарта FibreChannel:
- точка-точка (Point-to-Point)

- коммутируемая сеть (Switched Fabric)
- кольцо с арбитражем (Arbitrated Loop)

Кольцо с арбитражем Fibre Channel (классы обслуживания):
- класс 1 – выделенное соединение с подтверждениями
- класс 2 – передача без установки соединения с подтверждениями
- класс 3 – передача без установки соединения без подтверждений

Описание работы кольца с арбитражем:
• MONITORING - все принятые портом слова ретранслируются далее, т.е. порт передает в выходной канал принятый набор из 40 бит.
• ARBITRATING - порт переходит в это состояние, когда ему необходимо получить доступ к кольцу для передачи информации.
• ARBITRATION WON - состояние, в котором порт считается выигравшим арбитраж.
• OPENED - порт-приемник переходит в это состояние, когда он получает слово OPN с указанием своего адреса.
• OPEN - порт начинает передавать кадр с данными.
• EMITTED CLOSE - порт переходит в это состояние, когда у него больше нет данных для передачи, и для закрытия портов-приемников он передал служебное слово CLS.
• RECEIVED CLOSE - порт-приемник переходит в это состояние, когда он получает служебное слово CLS. В этом состоянии порт ретранслирует слово CLS и переходит в состояние MONITORING.

Протокол FC-AE-1553:
• Протокол FC-AE-1553 является протоколом верхнего уровня (в стеке Fibre Channel) и эмулирует работу канала MIL STD-1553B в кольце с арбитражем (это предназначение данного протокола)
• Оконечные устройства кольца с арбитражем сами не инициируют информационные обмены. За это отвечает специально выделенное оконечное устройство кольца с арбитражем, называемое контроллером кольца.
Передача данных от A к B:
1) передача сообщения от контроллера кольца оконечному устройству с адресом A с командой передать сообщение оконечному устройству с адресом B
2) передача сообщения с данными от оконечного устройства с адресом A оконечному устройству с адресом B.

Задача построения магистральных каналов информационного обмена с использованием кольца с арбитражем FC:
• Выбор порядка расположения оконечных устройств в кольце с арбитражем
– от порядка следования устройств зависит длительность передачи сообщений
• Назначение адресов оконечным устройствам
– влияет на арбитраж, если нет централизованного управления
• Построение расписания обменов
!!! Слайд 488 – почему такой странный порядок?

Построение расписания обменов:
• Для схемы с централизованным управлением. Эта задача возникает, если используется протокол FC-AE-1553 и формулируется аналогично задаче для канала MIL STD-1553B.
• Для схемы с децентрализованным управлением. В этом случае должно составляться расписание выставления заявок на арбитраж для каждого оконечного устройства.

28. (Лекция 8) Задача совместного планирования вычислений и обмена по каналу с централизованным управлением. Подходы к решению этой задачи. Жадный алгоритм совместного планирования, в т.ч. решение проблемы зависимости длительности передачи сообщений от привязки задачи- отправителя и задачи-получателя к абонентам канала.

Цель совместного планирования вычислений и обменов:

Построение расписания выполнения задач на вычислительных модулях в составе ИУС РВ и расписания передачи сообщений между ними по каналу с централизованным управлением с соблюдением ограничений:
– реального времени
– совместимости расписаний
– ограничений, связанных со спецификой аппаратных и программных средств ИУС РВ

Расписание выполнения задач:
• Требования реального времени
• На одном и том же вычислительном модуле в каждый момент времени может выполняться только одна задача
• Каждая задача запланирована на допустимый вычислительный модуль

Расписание передачи сообщений:
• Требования реального времени
• В каждый момент времени может передаваться только одно сообщение
• Технологические ограничения на обмен данными

Задача: По заданным наборам задач и сообщений построить корректные совместимые расписания выполнения задач и передачи сообщений, содержащие максимальное количество задач и сообщений

Подходы:
• 3 подхода:
– Сначала построить расписание сообщений, затем расписание задач
– Сначала построить расписание задач, затем расписание сообщений
– Одновременное построение обоих расписаний
• Для первых двух подходов существуют частные задачи, имеющие полное расписание, которое нельзя построить в рамках этих подходов

Жадный алгоритм:
Проблема: Изменяющаяся длительность передачи сообщений
Решение: Считать длительность передачи сообщения максимально возможной и корректировать её в процессе работы алгоритма

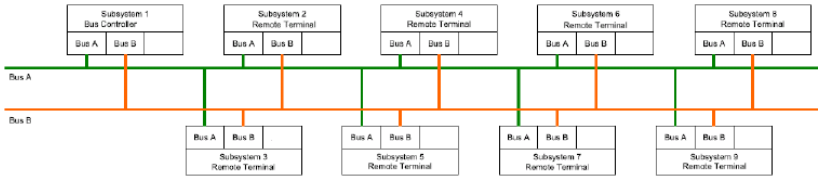
!!! Слайд 499 – совершенно не понимаю картинки на этом слайде.
!!! не ясно, что отвечать на «Жадный алгоритм совместного планирования, в т.ч. решение проблемы зависимости длительности передачи сообщений от привязки задачи- отправителя и задачи-получателя к абонентам канала»

29. (Лекция 9) Недостатки каналов точка-точка при использовании в ИУС РВ. Подход к устранению этих недостатков при помощи мультиплексных каналов, недостатки этого подхода. Организация сети ИУС РВ на основе коммутаторов. Преимущества и недостатки такой организации. Устранение недостатков за счёт поддержки виртуальных каналов.

Бортовые сети – обеспечение связи между бортовыми подсистемами
- надёжная доставка
- соблюдение требований реального времени

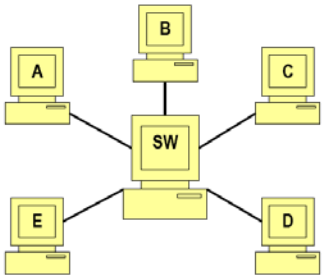
Недостатки каналов точка-точка:
- Много кабелей
- Пропускная способность: недоиспользование, нехватка, сложность наращивания
- Сложно закладывать резерв
- Проблемы с передачей данных по сложному маршруту
- Низкая адаптивность (невозможна реконфигурация)

Интеграция каналов, мультиплексирование трафика:



- Много каналов точка-точка → общая шина для многих потоков данных
- Проблема коллизий при доступе к шине
    - синхронизация доступа (нужно единое время)
    - либо централизованное управление (накладные расходы…)
- Последовательная обработка запросов => задержки
- Нет устойчивости к «генерации» в канале при выходе абонента из строя

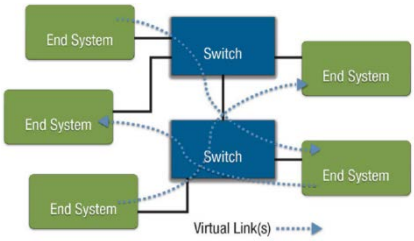
Коммутатор, параллелизм обмена:



- Дуплексные (двунаправленные) каналы
- Частичный параллелизм обмена
    - A→B || B→C || C→D – нет конфликта
    - A→B, C→B – конфликт на линии SW→B; как делить линию?
- Неустойчивость к «генерации»
- Проблема мультиплексирования потоков данных при отправке

- Оставшиеся конфликты – как лечить?
    - синхронизация доступа (нужно единое время)
    - централизованное управление (накладные расходы …)
    - верхние оценки задержек (а если между A и B большой поток?)

Виртуальные каналы:



- Разделение пропускной способности, разграничение потоков данных
    - => пригодность для сложного трафика из множества потоков данных
- Гарантированные верхние границы задержек
- Резервирование, гибкость реконфигурации
- Реализация: согласованные действия отправителя и коммутаторов
- Устойчивость к «генерации»
    - коммутатор сбрасывает слишком частые кадры

Протоколы коммутируемых сетей с поддержкой ВК (Виртуальных Каналов):
- AFDX (на базе 100 Мбит Ethernet)
- FC-RT (на базе Fibre Channel)
- Программно-конфигурируемые сети

30. (Лекция 9) Сети на основе стандарта AFDX: архитектура, стек протоколов, маршрутизация потоков данных. Параметры виртуальных каналов AFDX. Формирование трафика AFDX на оконечной системе, контроль трафика на коммутаторе.

Стандарт AFDX:
- Avionics Full-Duplex Ethernet (AFDX) – стандарт построения бортовых сетей на основе протокола Ethernet
- Основан на протоколе Ethernet
- Полнодуплексная передача данных
- Теоретическая пропускная способность – до 100 Мбит/с на одном физическом соединении

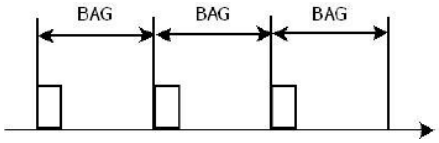
Архитектура сети AFDX:

Компоненты:
- Абоненты (бортовые подсистемы, отправители и получатели данных)
- Оконечные системы – интерфейс между абонентами и сетью
- Коммутаторы и физические соединения

Дублирование сети для увеличения надежности передачи:
- Кадры передаются одновременно в обе сети
- При диагностировании ошибки (например, несовпадение контрольной суммы) в одной сети данные берутся из другой сети
- На оконечной системе производится сброс кадра в случае, если кадр уже пришел из другой сети

Стек протоколов AFDX:
- Канальный уровень
    - Ethernet
    - Виртуальные каналы
        - Одна оконечная система – отправитель; одна или более оконечная система – получатель
        - Маршрут следования кадров виртуального канала прописан статически в коммутаторах
    - Маршрутизация
- Сетевой уровень
    - IP (без маршрутизации)
- Транспортный уровень
    - UDP

Параметры виртуальных каналов AFDX:
- Для каждого виртуального канала вводятся следующие параметры:
    - *BAG* – Bandwidth Allocation Gap – минимальный интервал времени вежду началами выдачи последовательных кадров на одном виртуальном канале (1-128 мс, является степенью двойки)
        - *BAG* используется для достижения максимальной выделенной пропускной способности (кадры можно пускать друг за другом и не подряд)
        - В дальнейшем рассматривается, когда кадры посылаются без промежутков между соседними *BAG*-интервалами



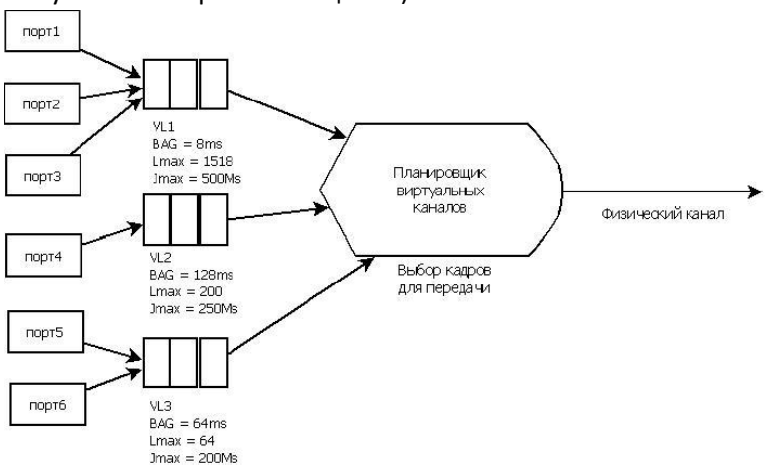
    - *Lmax* – максимальный размер кадра (<=1518байт)
    - *Jmax* – максимально допустимое отклонение между кадрами от *BAG*

Пропускная способность виртуальных каналов:
- Вычисление:
    - *Bandwidth = Lmax / BAG*
        - *BAG* = 32 мс
        - *Lmax* = 200 байт
        - *Bandwidth* = 200 байт / (32 / 1000) сек = 6250 байт/сек
    - Ограничение на зарезервированную пропускную способность на физическом канале (проводе):

$\sum_{VL=1..n} \frac{L_{VL,max}}{BAG_{VL}}$ <=100 Мбит/с

Управление виртуальными каналами:

Сообщения на входе в AFDX бьются на кадры, которые в рамках одной оконечной системы приходят на планировщик виртуальных каналов, где он выбирает кадры для передачи (происходит мультиплексирование по времени). (т.к. расписание передачи кадров чётко прописано, то коммутаторы смогут отмаршрутизировать, а получатели собрать сообщения)



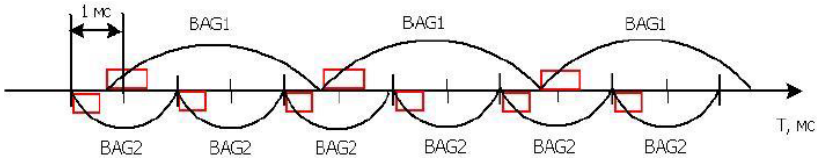
Формирование трафика:

При формировании трафика, на отправителе – мультиплексирование по времени.

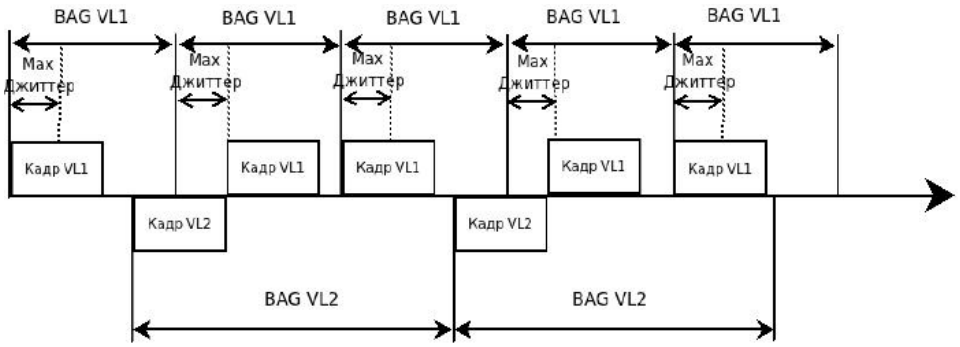
При мультиплексировании определяется значение джиттера

- Пример для нулевого джиттера:

  Кратность BAG-ов, известность всех L_{max}, отсутствие джиттера *готовности кадров* в каждом ВК => исключено взаимное влияние ВК, отсутствует джиттер *выдачи кадров* в каждом ВК



- Пример для не нулевого джиттера (такого пытаются избегать):



Коммутатор:

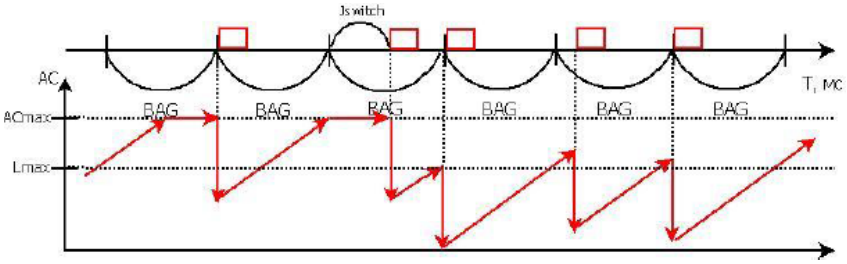
<u>Функции коммутатора</u>:

- Маршрутизация кадров по пути следования виртуальных каналов (пути виртуальных каналов конфигурируются статически)
- Фильтрация трафика (контроль целостности кадра, контроль следования кадра по виртуальному каналу)
- Контроль трафика
  - размер кадра (не должен превышать Lmax)
  - BAG, Jmax
  - нарушение => сброс кадра

<u>Контроль трафика на коммутаторе</u>:

- Производится на входном порту коммутатора
- Используется алгоритм, основанный на вычислении кредита
- AC – кредит, растет с течением времени до значения ACmax
- При приходе кадра AC уменьшается на размер кадра; если кредита не хватает – кадр сбрасывается
- Кредит соответствует количеству байт, которые пропускает канал
  - За время *BAG* кредит увеличивается на *Lmax*

- ACmax – соответствует количеству байт, которое позволяет пропустить 2 кадра за (*BAG – Jmax)*



31. (Лекция 9) Задачи проектирования сети AFDX. Оценка длительности передачи кадра через сеть AFDX. Профиль Fibre Channel реального времени, его сходства и отличия от протокола AFDX.

Задачи проектирования сети AFDX:

- Дано: потоки данных, требования к их передаче в реальном времени
    - размер сообщения
    - частота передачи
    - макс. допустимый джиттер (end-to-end)
    - макс. допустимая задержка (end-to-end)
- Требуется:
    - построить систему виртуальных каналов:
        - маршруты
        - параметры (BAG, Lmax)
    - рассчитать конфигурационные параметры сетевых устройств – коммутаторов, абонентов (в т.ч. Jmax)

Оценка длительности передачи кадра:

- Необходима для оценки длительности передачи сообщения
- Актуальность: требования реального времени – длительность не должна превышать заданных значений
- Длительность вычисляется с момента поступления кадра для выдачи в канал до момента поступления кадра на оконечную систему-получатель

Длительность передачи кадра:

- максимальный джиттер на отправителе

    Мультиплексирование:

    - При мультиплексировании может возникать джиттер
    - Максимальная задержка – при максимальном джиттере
    - Максимальное значение джиттера – при ожидании кадров всех других виртуальных каналов

    Вычисление максимального джиттера на отправителе: $J_{max} \leq J_T + \frac{\sum_{i \in VLs} L_{i,max}}{R}$

        - *VLs* – множество виртуальных каналов, формируемых на оконечной системе-отправителе
        - *R* – скорость выдачи данных на канал (100 Мбит/сек)
        - J_T – технический джиттер (время обработки кадра), в AFDX равен 40 мкс

- длительность передачи по каналам

    Длительность передачи кадров по каналам: $t_{links} = n * \frac{L_{max}}{R}$

        - *R* – скорость выдачи данных на канал (100 Мбит/сек)
        - *n* – количество каналов передачи данных на пути следования кадра

- задержки на выходных портах коммутаторов

    Оценка задержки кадра на коммутаторе:

    В очереди перед кадром: кадры с других ВК, идущие на тот же выходной порт

    - Сколько этих кадров?
        - интервал накопления M: макс. интервал после предыдущего кадра данного ВК
        - M = BAG + накопленный макс. джиттер
        - число кадров с другого ВК, набежавших за интервал M: см. формулы анализа времени отклика
        - какие-то из набежавших кадров уже выданы в выходной порт к моменту прихода «нашего» кадра (=> их выдачи не нужно ожидать)

Оценка длительности и джиттера передачи сообщения:
$Dur_{max}(msg) = \mu + \delta + \Delta$
μ - const (разбиение и сборка сообщения)
δ - время выдачи сообщения в канал (выдача последнего кадра)
Δ - длительность передачи кадра
$J(msg) = Dur_{max}(msg) - Dur_{min}(msg)$

Профиль реального времени Fiber Channel:
- Базовая схема обмена данными позаимствована из AFDX
    - виртуальные каналы
    - 100% резервирование
- В каждом коммутаторе - несколько таблиц ВК
    - переключение в реальном времени между конфигурациями сети
    - нет избыточного резервирования пропускной способности для поддержки нескольких наборов ВК (втч. в разных режимах ИУС РВ)
    - поддержка статического набора возможных миграций задач
- Управление джиттером отправки целых сообщений
- Система приоритетов → поддержка нерегулярных (апериодических) сообщений
    - низкий приоритет -> нет помех для сообщений из ВК
    - высокий приоритет -> доставка без задержек (ценой задержки сообщений из ВК)
- Встроенные средства синхронизации времени

32. (Лекция 9) Перспективы применения программно-конфигурируемых сетей (ПКС) в ИУС РВ. Выбор между активным и пассивным режимом. Функциональность приложения управления трафиком для контроллера ПКС в ИУС РВ. Ниша для применения ПКС в ИУС РВ.

ПКС – Программно-Конфигурируемые сети:
Бортовая сеть управляется приложением функционирующем на контроллере
- Синхронизация времени: например, протокол PTP
- Пассивный режим:
    - правила формируются заранее и загружаются на коммутаторы до старта системы
    - большую часть времени работы ИУС РВ коммутаторы функционируют автономно (без обращения к контроллеру)
- Активный режим:
    - контроллер постоянно осуществляет мониторинг обмена данными
    - обнаружены новые потоки данных -> новые правила формируются и загружаются в коммутаторы
- Первый шаг внедрения ПКС на борт: схема выделения пропускной способности
    - воспроизведение схемы на основе алгоритма текущего ведра (по образцу AFDX)?

Активный vs Пассивный режим:
• Отказоустойчивость / резервирование
    - активный режим: контроллер и управляющая сеть обязательно должны быть продублированы
    - пассивный режим: более высокая устойчивость к сбоям контроллера и управляющей сети
• Реконфигурируемость
    - потоки данных в ИУС РВ являются предсказуемыми
    - правила для заранее определенных режимов могут быть сформированы заранее
    - правила для динамических режимов (в т.ч. при отказах оборудования) могут быть сгенерированы контроллером и в пассивном режиме
→ предпочтителен пассивный режим

Функциональность приложения для контроллера ПКС в ИУС РВ:
• Построение маршрутов передачи сообщений между абонентами сети с обеспечением требуемого качества обслуживания, в т.ч. ограниченных задержек передачи данных и ограниченного джиттера
• Динамическая адаптация маршрутов в случае сбоев сети или миграции задач

• Формирование правил для коммутаторов, в т.ч.:
- правил проверки трафика на соответствие требованиям качества обслуживания
- правил маршрутизации
- правил разделения пропускной способности сети между потоками данных

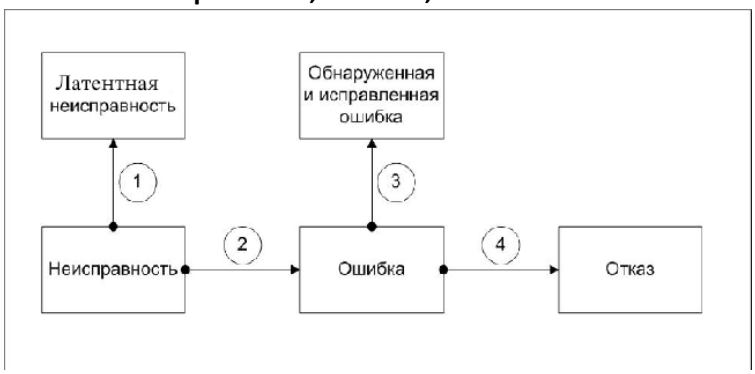
Востребованность ПКС в ВИУС РВ (Ниша ПКС):
• Отказоустойчивые системы с миграцией задач
- сценарии миграции задач при множественных отказах не могут быть просчитаны заранее (комбинаторный взрыв числа сценариев)
- AFDX: резервирование виртуальных каналов под сбойные режимы непродуктивно расходует пропускную способность сети
- FC-RT: ограниченный объем памяти коммутаторов под таблицы ВК для сбойных режимов
• Динамическое формирование и смена режимов ИУС РВ
– реконфигурируемость – «изюминка» ПКС
• Динамическое подключение оборудования и ПО
- подключаемые в режиме plug-and-play устройства (датчики, устройства связи и т.п.)
- ПО загружается с подключаемого устройства и выполняется на бортовом вычислителе с архитектурой ИМА в отдельном разделе
- ВК для обмена бортового вычислителя с подключенным устройством автоматически настраиваются контроллером ПКС
- пример: многофункциональный беспилотник со сменными датчиками

33. (Лекция 10) Понятия неисправности, ошибки и отказа; связь между ними. Классификация неисправностей. Шаги противодействия неисправностям. Общие принципы построения отказоустойчивых систем.

Цена ошибок в ИУС РВ – млн-млрд.

Избежать неисправностей – невозможно. Главное – минимизировать вероятность отказа.

Понятия неисправности, ошибки, отказа:



Классификация неисправностей 1:
• Активность: латентные / активные
• Постоянство: проходящие / постоянные
• Источник: внешнее воздействие / ошибка разработки
• Распространение последствий: обнаруживаются и локализуются / проникают в другие подсистемы
• Одиночность: одиночные / групповые
• Взаимосвязанность: независимые / связанные

Классификация неисправностей 2:
• По локализации
- Программные
- Аппаратные

• По этапу возникновения
    - Проектирование/разработка
        => все изделия, созданные по проекту
    - Производство
        Дефект серии => все изделия серии
        Дефект при производстве конкретного изделия
    – Эксплуатация
        => одиночные изделия, с учётом особенностей эксплуатации

Шаги противодействия неисправности:
• Обнаружение
• Ограничение распространения
• Маскировка
• Диагностика
• Восстановление
• Возобновление штатного функционирования

Борьба с серийными неисправностями:
• Возникают на этапе проектирования, разработки или серийного производства
    - На ранних этапах их и следует обнаруживать…
• Затрагивают всю серию компонентов
    - Отзыв серии и всех использующих её систем…
• Борьба: использование проектного или реализационного разнообразия
    - Аппаратура: различные архитектуры, производители и элементная база
    - ПО: различные языки, алгоритмы/подходы, команды разработчиков

Специфика ИУС РВ:
• Жесткие условия эксплуатации
- Внешние воздействия вызывают неисправности оборудования
• Недопустимость прекращения функционирования при возникновении ошибки
- Ошибка = реализация неисправности
• Невозможность оперативного ремонта (или ремонта вообще) (самолёт, спутник)
• Реакция на ошибки должна укладываться в директивные сроки

Принципы построения отказоустойчивых систем:
• Недопустимость единственной точки отказа
• Поддержка локализации отказа (обнаружения отказавшего компонента)
• Нераспространение последствий отказа далее по системе
- Защита аппаратуры
- Блокировка распространения некорректных результатов вычислений
• Постепенная деградация
- По мере отказа подсистем, вначале отключаются второстепенные функции
- Функции, критические для выживания/восстановления системы поддерживаются «до последнего»
- Защитный режим: поддержка существования + обеспечение удалённого доступа для диагностики и обслуживания

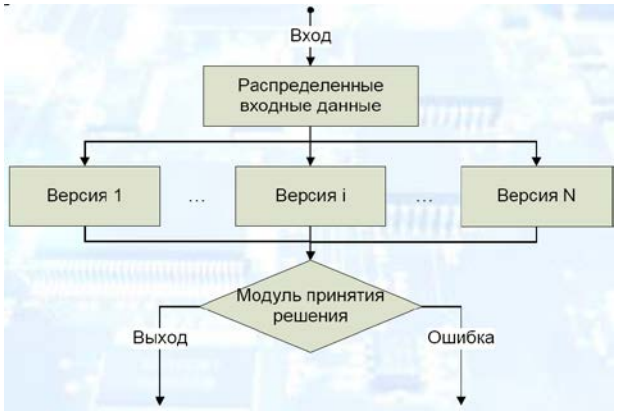
Механизмы обеспечения отказоустойчивости (МОО):
- аппаратные
- программные
- аппаратно-программные

34. (Лекция 10) Аппаратные, программные и программно-аппаратные методы обеспечения отказоустойчивости ИУС РВ. Бортовая ИУС космического челнока как пример использования МОО.

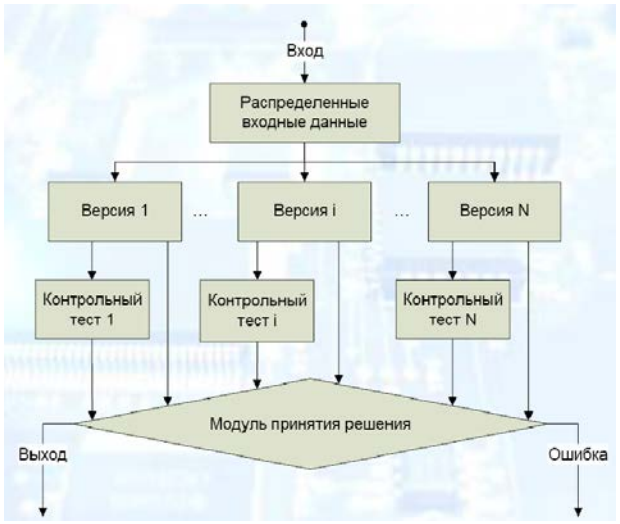
Механизмы обеспечения отказоустойчивости (МОО):
- аппаратные
- программные
- аппаратно-программные

Аппаратное резервирование:
• По разнообразию компонентов
  - Использование идентичных компонентов
    Борьба с дефектами изделия, в т.ч. возникающими в ходе эксплуатации
  - Использование различных компонентов
    Функционально идентичны
    Различная элементная база, производитель, проект и т.п.
    Борьба с дефектами проекта, серии (не только изделия)
• По уровню
  - Система/подсистема
  - Отдельные компоненты
• Активное: основные и резервные компоненты функционируют одновременно
  - Синхронизация данных
  - Минимальное время переключения на резервный компонент
  - Использование результатов:
    Игнорирование результатов резервных компонентов
    Голосование (нет выделенного основного компонента)
  - Повышенное энергопотребление
• Пассивное: резервный компонент включается при выходе основного из строя
  - Затраты времени на инициализацию
  - Проблема записи состояния основного компонента (нужно внешнее запоминающее устройство)
  - Экономия энергии
• Жаргон: «горячий» и «холодный» резерв

Программные МОО:

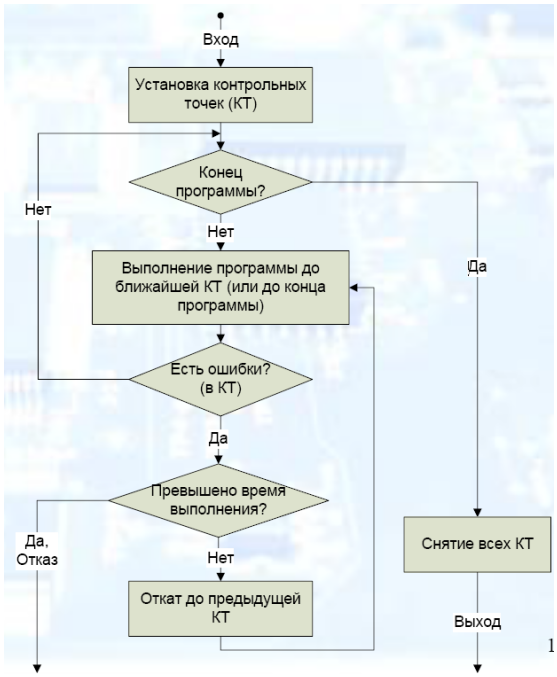


N-версное программирование:
- Версии ПО функционально эквивалентны
- Различаются: алгоритмы, методы разработки, языки программирования, группы разработчиков



N-самотестируемое программирование:

- Версии ПО запускаются последовательно (до первого результата, прошедшего контрольный тест) или параллельно (голосование среди успешных результатов)

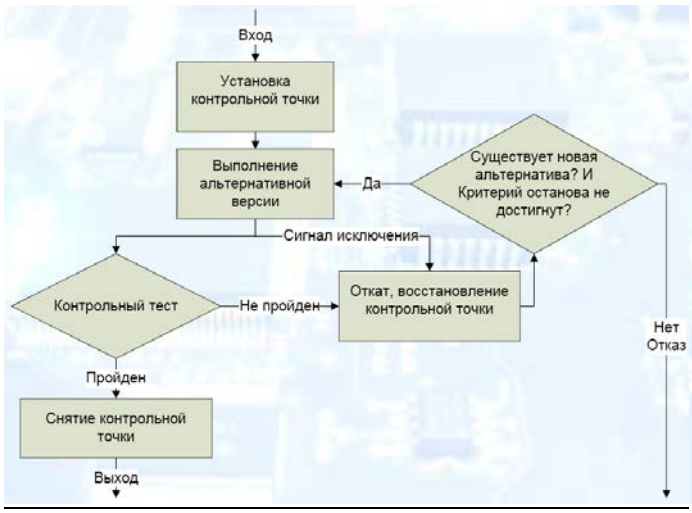


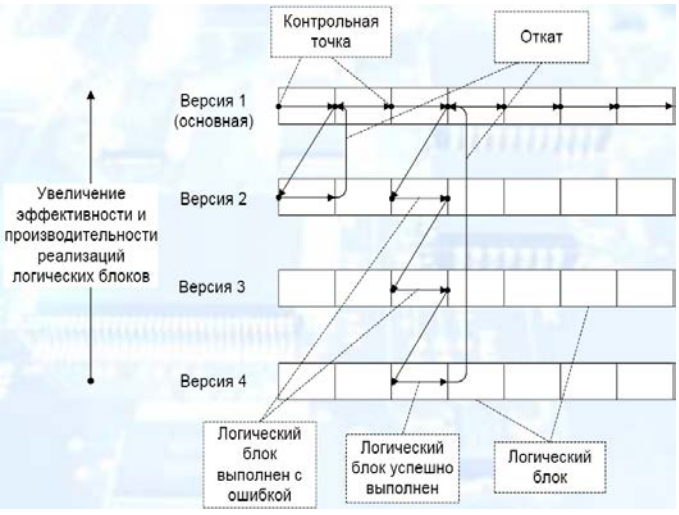
Контрольные точки:

• Борьба только со случайными «однократными» неисправностями (инверсия бита памяти и т.п.)

• Не помогает от «постоянных» неисправностей, в т.ч. в ПО

• Затраты ресурсов на сохранение состояния в КТ, сложность механизмов поддержки сохранения

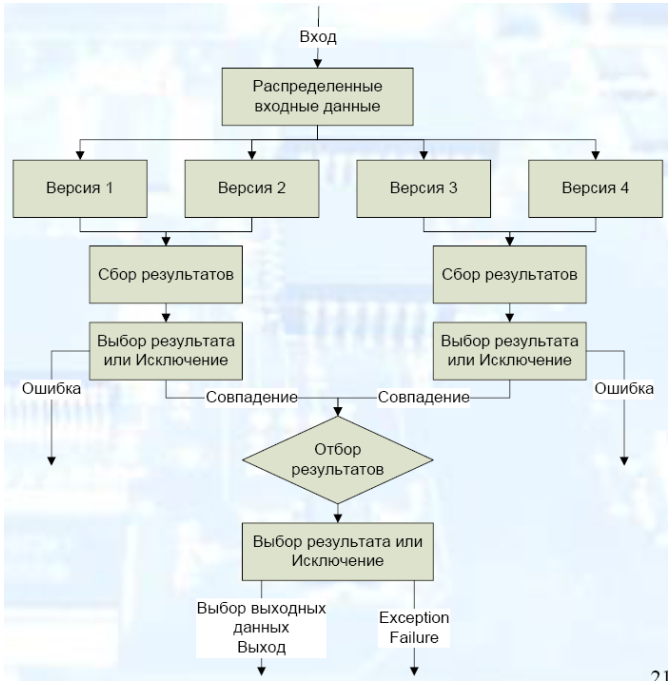




Восстановление блоками:

- Сочетает N-версионное программирование и контрольные точки
- Перебор блоков по убыванию «скорости»
- WCET оценивается по наихудшему пути…

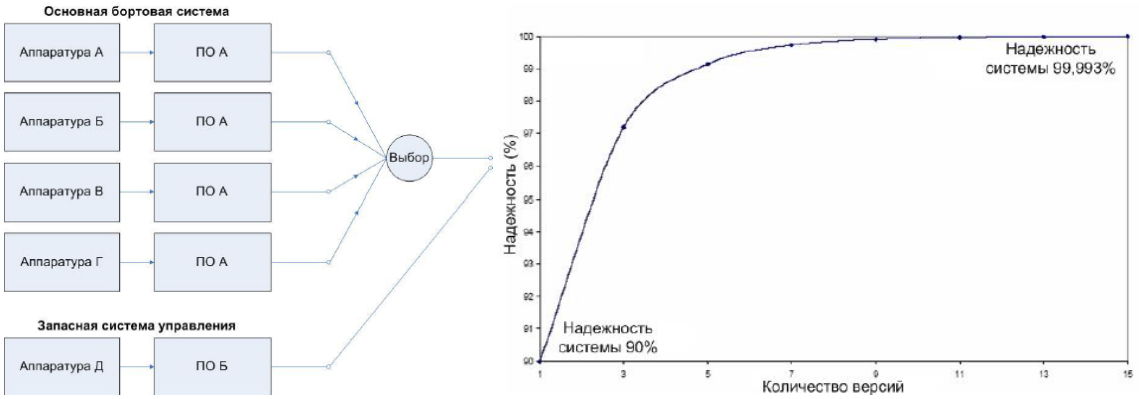
Программно-аппаратные ММО:



N-самоконтролируемое программирование:
- Борьба с аппаратными и программными ошибками
- ПО в каждой паре: одна версия + контрольный тест, или более одной версии + алгоритм выбора

Активное резервирование + N-версионное программирование:
- На каждом аппаратном компоненте выполняется своя версия программы
- Все компоненты активны
- Борьба с программными и аппаратными неисправностями

Пример ММО на космическом челноке:



(за пределами билета):

Принципы проектирования безопасных и отказоустойчивых систем *(Real-Time Systems: Design Principles for Distributed Embedded Applications):*

1. Требования отказоустойчивости и безопасности должны быть неотъемлемой частью спецификации системы, а для некоторых систем – основой для процесса проектирования.

2. Ожидаемые виды отказов и частоты их возникновения должны быть определены в начале процесса проектирования.

3. Должны быть определены области локализации неисправностей в системе (fault containment regions). Последствия неисправности в одной из таких областей не должны распространяться на другие области.

4. Понятие времени и состояния системы должны быть строго определены. В противном случае затруднительно отличить первичную неисправность от ее последствий.

5. Необходимо четко определить интерфейсы, чтобы скрыть внутреннее устройство компонентов системы.

6. Необходимо обеспечить независимость возникновения неисправностей в различных компонентах.

7. Компонент системы должен считать себя корректно функционирующим, пока два или более других компонента не примут противоположную точку зрения.

8. Механизмы обеспечения отказоустойчивости должны быть устроены так, чтобы не усложнять анализ поведения системы. МОО должны быть отделены от основной функциональности системы.

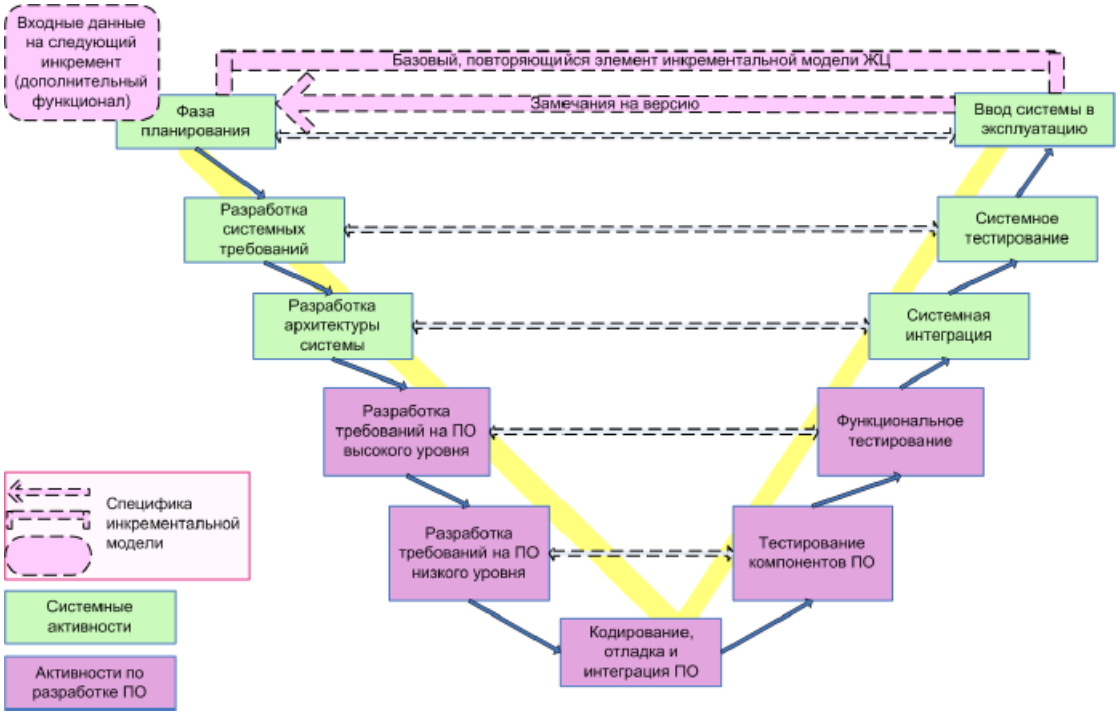
9. Возможность диагностирования должна быть заложена в систему на этапе проектирования. В частности, должна обеспечиваться возможность выявления скрытых неисправностей, не проявляющихся в поведении системы.
10. Интерфейс с оператором должен быть интуитивно понятным и устойчивым к ошибкам. Безопасность должна обеспечиваться несмотря на ошибки человека-оператора.
11. Любая аномалия функционирования системы должна быть зарегистрирована. Некоторые аномалии не обнаруживаются на уровне интерфейсов между компонентами. Регистрация должна фиксировать в т.ч. внутренние аномалии, в противном случае они могут быть замаскированы в результате работы МОО.
12. Система должна реализовывать стратегию «никогда не сдавайся», гарантируя заданный минимальный уровень обслуживания. Полный выход из строя крайне нежелателен.

35. (Лекция 11а) Требования к средствам тестирования ИУС РВ. Архитектура стенда тестирования ИУС. Задачи, требующие работы с натурными устройствами ИУС на стенде. Аппаратная база стенда. Примеры стендов, построенных по разработанной архитектуре. Процесс совместного применения стендов для отработки бортовых ИУС РВ.

Функции ИУС:
- Контроль состояния управляемого объекта
- Управление движением объекта или его частей
- Отслеживание положения объекта или его частей в пространстве
- Обмен данными с внешними системами
- Управление специализированными приборами (прикладной нагрузкой)
- Обмен данными с оператором
    - отображение данных
    - ввод данных

Неоднородность ИУС (по каналам, по данным, по устройствам). Проблема унаследованных устройств.

Жизненный цикл ПО ИУС:
Фаза планирования
    Разработка системных требований
        Разработка архитектуры системы
            Разработка требований на ПО высокого уровня
                Разработка требований на ПО низкого уровня
                    Кодирование, отладка и интеграция ПО
                Тестирование компонент ПО
            Функциональное тестирование
        Системная интеграция
    Системное тестирование
Ввод системы в эксплуатацию



Требования к средствам тестирования ИУС:

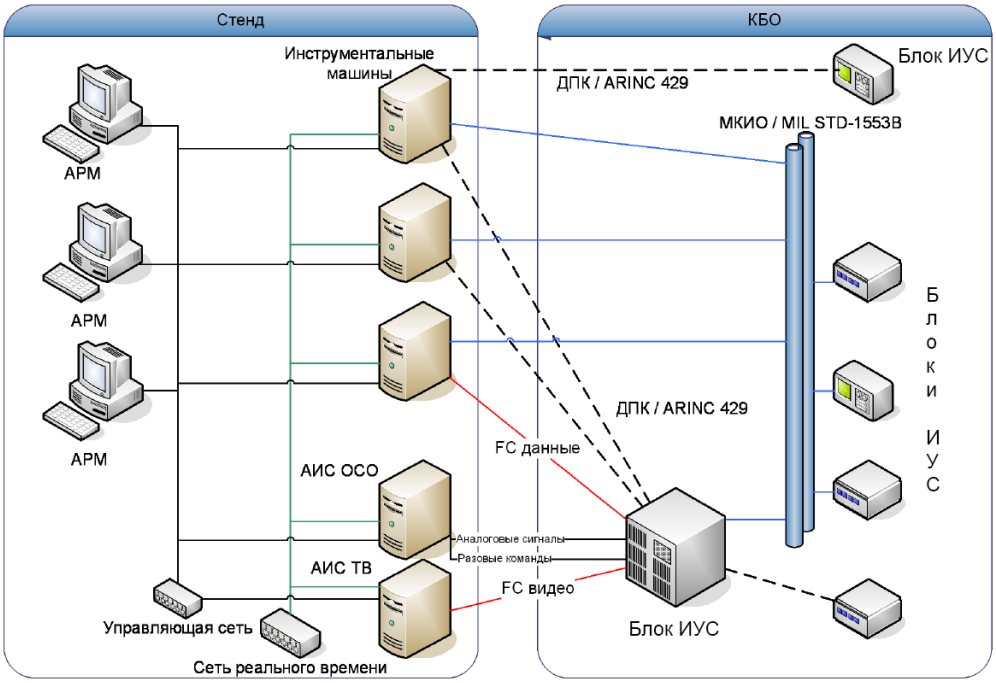
- Поддержка тестирования ПО на целевом вычислителе без инструментирования аппаратуры или ПО вычислителя
- Поддержка обмена данными через все типы каналов, используемых в ИУС
  - выдача в каналы тестовых данных и прием ответных данных для последующего анализа
  - мониторинг обмена по каналам бортовых интерфейсов и обеспечение доступа тестовых сценариев к результатам мониторинга
  - поддержка формирования сбойного трафика
- Поддержка тестирования временных характеристик функционирования целевой системы
  - формирование и выдача тестовых данных в режиме реального времени
  - измерение задержек поступления ответных данных
- Поддержка многомашинных конфигураций
  - одновременный обмен по десяткам (сотням) каналов
- Поддержка автоматического и интерактивного тестирования
  - интерактивное: для проверки индикаторов и пультов управления
- Поддержка пакетного режима выполнения тестов
- Поддержка оперативного управления тестированием
  - выбор порядка выполнения тестов
  - задание значений тестовых данных
- Поддержка оперативного отображения хода тестирования
  - значения тестовых данных и ответных данных от тестируемой системы
  - протоколы тестирования
- Поддержка прослеживаемости требований и формирования отчётов по результатам тестирования
  - задание соответствия требований тестовым сценариям
  - формирование матрицы прослеживаемости требований
  - формирование отчёта о прохождении тестов и выполненности требований по результатам тестирования
- Интеграция со средствами поддержки разработки ПО ИУС
  - средства управления версиями (хранение ПО и тестов в едином репозитории)
  - средства управления требованиями
  - база данных бортовых интерфейсов (форматы информационных сообщений)
- Единый подход к тестированию для различных фаз жизненного цикла ИУС

Комплекс средств тестирования ИУС:

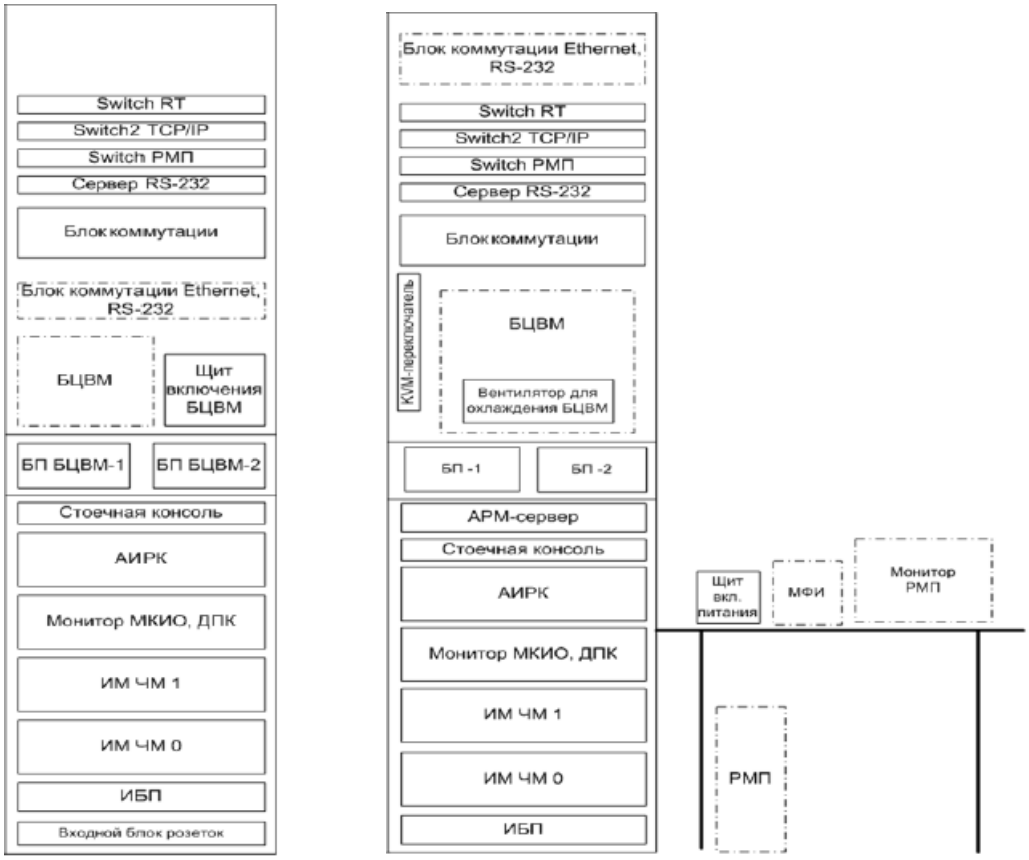
- Разработан в Лаборатории вычислительных комплексов ВМК МГУ
- Предназначен для тестирования устройств ИУС через каналы бортовых интерфейсов (КБИ)

• Функционирует на ПК под управлением ОС Linux, в состав которых входят адаптеры КБИ
• Поддерживает распределенное выполнение тестовых сценариев
• Удовлетворяет перечисленным выше требованиям
• Положен в основу семейства стендов тестирования, отработки и интеграции ИУС

Архитектура стенда тестирования ИУС:

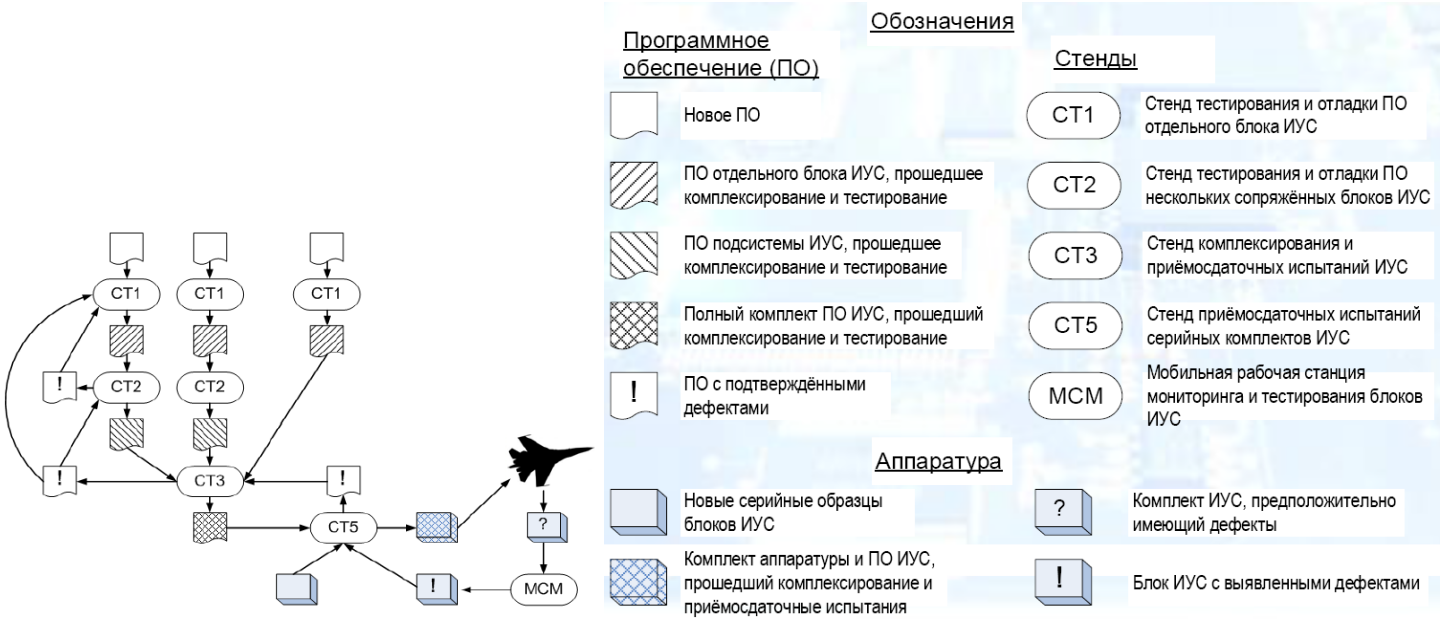


Задачи, требующие работы с натурными устройствами ИУС:
• интеграция аппаратуры и ПО, отладка ПО ИУС на целевой платформе
• интеграция компонентов ПО ИУС, в т.ч. компонентов, поступающих от предприятий-соразработчиков
• интеграция подсистем ИУС, а также ИУС в целом как многокомпонентной аппаратно-программной системы
• функциональное и квалификационное тестирование ПО ИУС
• приемосдаточные испытания серийно выпускаемых комплектов ИУС
• диагностика блоков ИУС, по которым поступили рекламации (жалобы заказчика)
• диагностика блоков ИУС в составе объекта



Стенд отработки ПО БЦВМ и Стенд отработки БЦВМ+МФИ

Технологический цикл применения семейства стендов отработки ИУС:



Аппаратная база стенда:

- Промышленные компьютеры (PICMG)
- Платы-расширители шин (ISA, PCI, PCI Express)
- Процессорные платы
- Адаптеры бортовых интерфейсов

Пример коммутатора:

- *Программируемый коммутатор Fibre Channel*
- *Оптический сигнал преобразуется в электрический*
- *Коммутация электрического сигнала при помощи ПЛИС*

Программные средства функционального тестирования в составе стенда:

Средства АРМ (Автоматизированного рабочего места) Средства ИМ (!!! Инструментальной машины?)

36. (Лекция 11а) Основные понятия языка описания тестов (ЯОТ), используемого на стенде. Тестовые компоненты, интерфейсы, сообщения, битовые поля, тестовые случаи, тестовые шаги. Типовая организация тестового шага. Взаимодействие с пользователем при интерактивном тестировании. Протокол тестирования, его содержание и назначение. Процедура подготовки и проведения тестирования.

Язык описания тестов (ЯОТ):

- Поддерживает тестирование ИУС через каналы бортовых интерфейсов (MIL-STD-1553B, ARINC 429, FC-AE-ASM, CAN, AFDX и т.п.) с использованием следующих основных элементов:
  - Сообщения и слова КБИ (каналы бортовых интерфейсов)
  - Битовые поля сообщений/слов
- Предоставляет операторы для:
  - Определения структуры слов/сообщений (состав битовых полей, их имена и расположение)
  - Задания и обработки значений битовых полей
  - Автоматической проверки тестируемых условий
  - Взаимодействия с пользователем
  - Задания структуры тестового сценария

Основные понятия ЯОТ:

- **Тестовые компоненты (ТСК)** – это средство компоновки объектов ФТ (!!! Функционального тестирования?) в составе стенда. ТСК определяет состав и последовательность выполнения тестов.

  С каждым ТСК связаны:
  - имя (определяет тип ТСК)
  - описание сообщений КБИ с указанием их структуры
  - описание тестов

  Структура ТСК:
  - Заголовок ТСК определяет состав интерфейсов, состав и структуру сообщений КБИ, состав тестов и их привязку к требованиям
  - Тело ТСК определяет состав и логику выполнения тестовых случаев и тестовых шагов

- **Интерфейсы** – это объект средств ФТ, предоставляющий программный интерфейс для взаимодействия ТСК с натурными блоками ИУС, а также друг с другом
  - Интерфейсы обеспечивают доступ ТСК к адаптерам КБИ
  - Через интерфейсы обеспечивается передача и прием сообщений

- **Сообщения бортовых интерфейсов**

  Сообщение MIL STD-1553B - это объект средств ФТ, соответствующий сообщению бортового интерфейса MIL STD-1553B.

С каждым сообщением связаны: имя, интерфейс MILS, адрес, подадрес.

С отдельными словами в составе сообщения можно работать как с элементами массива. (сообщения для других типов интерфейсов устроены аналогично)

- **Битовые поля сообщений**

Битовое поле – это объект средств ФТ, соответствующий группе последовательных битов в составе слова сообщения.

Бит – это битовое поле, соответствующее 1 биту.

С каждым битовым полем связаны:

- имя
- позиция в сообщении (номер первого слова, номер начального бита, кол-во битов)
- признак использования знакового или беззнакового поля
- коэффициент масштабирования (ЦМР)

С битовым полем можно работать как с числовой переменной.

- **Тестовые шаги (test step)** - фрагмент теста, проверяющий элементарную функцию тестируемого устройства.

Тестовый шаг может проверять правильность приема/передачи отдельного сообщения или слова, или отдельного битового поля в составе сообщения.

Типичная структура тестового шага:

- Установка тестовых условий (входных значений)
- Ожидание результата
- Проверка ожидаемого результата (+ нумерация)
- Операторы if и RTS_WHEN

- **Тестовые случаи (test case)** – последовательность связанных по смыслу тестовых шагов, соответствующих отдельным разделам или отдельным требованиям системной спецификации на тестируемое устройство.

“Минималистический” тестовый случай может состоять из одного тестового шага.

При выполнении тестового случая увеличивается нумерация тестов на соответствующем уровне.

Протокол тестирования. Содержание:

- Информация о смене (назначении) номеров тестов
- Информация о смене выполняемых тестов
- Результаты тестирования: «Тест прошел» / «Тест не прошел» с указанием номера теста
- Результаты всех команд взаимодействия с пользователем
- Комментарии с описаниями ожидаемого поведения теста и другая дополнительная информация, выдаваемая командой RTS_COMMENT
- Информация о текущих значениях объектов ФТ (в т.ч. битовых полей и битов), выдаваемая командой RTS_DISPLAY

Назначение: Протоколы тестирования служат основой для формирования отчетов по тестированию.

Задаются такие параметры как (например, для системы, разработанной в лаборатории мгу):
Входные и выходные сообщения
Битовые поля и биты
Тесты
Соответствия требований идентификаторов тестам
Временная задержка/отправка данных
Взаимодействие с пользователем (Пользователь выполняет все действия через интерфейс СОВУ)

Примеры для системы разработанной лабораторией в мгу:
Сообщения MIL STD-1553B:
RTS_OUTPUT_MESSAGE (<имя>,
<интерфейс MILS>,
<адрес>,
<подадрес>,

<кол-во слов в сообщении>,
<период в мс>)
RTS_INPUT_MESSAGE (<имя>,
<интерфейс MILS>,
<адрес>,
<подадрес>,
<кол-во слов в сообщении>,
<период в мс>)
RTS_OUTPUT_ / RTS_INPUT_TERM_MESSAGE (<имя>,
<интерфейс MILS>,
<подадрес>,
<кол-во слов в сообщении>)

Битовые поля и биты:
RTS_BITFIELD (<имя поля>,
<имя сообщения> [<номер слова>],
<начальный бит>,
<кол-во битов>)
RTS_BOOLEAN (<имя поля>,
<имя сообщения> [<номер слова>],
<позиция бита>)

Задание иерархического идентификатора теста:
Идентификатор теста задается в формате: <имя 1>.<имя 2>....<имя n>
Например: «Су35.СОЛС.Наземный_контроль.17»
RTS_TESTNAME (“<имя теста>”)
RTS_TESTNUM (<номер теста>)
RTS_TEST_SHOW ()
RTS_TEST_PASSED ()
RTS_TEST_FAILED ()

Задание соответствия требований идентификаторам тестов:
RTS_DOCUMENT (“<requirement specification document name>”,
RTS_REQUIREMENT (<requirement identifier>, “<requirement name>”),
RTS_REQUIREMENT (<requirement identifier>, “<requirement name>”),
…)
RTS_TESTSPEC (“<test name (or name template)>“,
<requirement identifier>, <requirement identifier>, ...)
Пример:
Заголовок:
RTS_DOCUMENT (“Board computer system initialization”)
RTS_REQUIREMENT (BCS.INIT.START, “Starting up MIL STD-1553B subscribers”))
RTS_REQUIREMENT (BCS.INIT.WORK, “Starting BCS operation main loop”))
RTS_TESTSPEC (“3.2.{1-4}“, BCS.INIT.START)
RTS_TESTSPEC (“3.2.{1-4}“, BCS.INIT.WORK, BCS.SKV.WORK)
Тело:
RTS_TESTNAME(3.2.2)
// логика тестового шага

Временная задержка, отправка данных:
// Задержать выполнение теста на заданное время
RTS_DELAY (<duration (ms)>)
// Выдать сообщение в канал (для активных абонентов)
RTS_EXECUTE (<Message/word name>)

Взаимодействие с пользователем:
// Отобразить значение объекта и занести это значение в протокол
RTS_DISPLAY (<object name>)
// Отобразить и записать в протокол текстовое сообщение
RTS_COMMENT (<message text>)
// Приостановить выполнение теста и ждать команды пользователя на продолжение
RTS_PAUSE ()
// Запросить у пользователя заключение (тест пройден / не пройден) и завершить тест в соответствии с полученным заключением
RTS_CONFIRM ()
// Запросить у пользователя заключение и занести его в протокол
RTS_REQUEST ()
// Запросить у пользователя текстовый комментарий и занести его в протокол
RTS_EXPLAIN ()

Оператор RTS_WHEN

```
RTS_WHEN ( “W46A185_13 == 0” , 15) { // ожидание 15 мс
        RTS_THEN():
                RTS_COMMENT ( “ERROR Circuit- Command Port (W46A185_13 = 0)” );
                RTS_TEST_FAILED();
                break;
        RTS_OTHERWIZE():
                RTS_TEST_PASSED();
                break;
}
```

Работа с протоколами тестирования:
Заголовок:

```
        RTS_LOG_FILE(LOG1, “log1.log”)
        // Файл «log1.log» будет помещен в подкаталог «ПротоколыТестирования» каталога результатов эксперимента
```

Тело:

```
        RTS_LOG(LOG1)
        RTS_CLEAR_LOG(LOG1)
```

Подготовка и выполнение тестирования:
Создание проекта
Редактирование кода ТСК
Привязка ТСК к инструментальным машинам
Привязка интерфейсов к адаптерам КБИ
Параметры эксперимента
Оперативное управление экспериментом. Инструменты:
- Ожидание ввода данных
- Ожидание выбора теста
- Циклическое выполнение теста
- Задержка циклического выполнения
- Выбор внеочередного теста

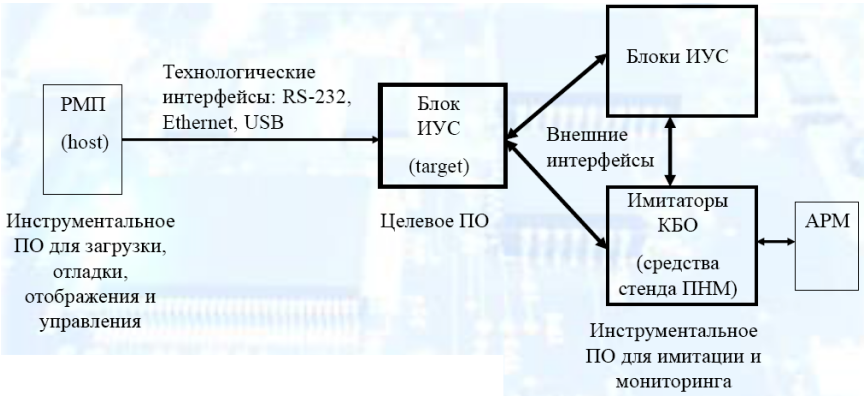
Обработка результатов эксперимента:
- Формирование отчётов по результатам тестирования
- Анализ временной диаграммы событий эксперимента

Перспективки:
• Автоматизация проверки выполнения расписаний обмена

• *Автоматизация тестирования индикаторов*
  – *интеграция с «блоком нажатия кнопок»*
  – *перехват изображения*
  – *сравнение с эталоном*
• *Автоматизация генерации тестов по описанию требований*
• *Интеграция тестов с отладочными средствами БЦВМ*
• *Новые внедрения*
  – *существующие: Су-35, Т-50/ПМИ, МС-21*

37. (Лекция 11б) Уровни информационного обмена по каналам в ИУС РВ. Способы подключения монитора к каналам различной топологии. Задачи мониторинга на различных уровнях: физическом, канальном, логическом. Средства мониторинга обмена по каналам в ИУС РВ на перечисленных уровнях.

Виды интерфейсов ИУС РВ:
- Технологические интерфейсы
- Внутренние (локальные) интерфейсы
- Бортовые интерфейсы (внешние)



Внутри-блочные интерфейсы:
- ME32 - 60 МБ/с
- PCI32 33 МГц - 60-80 МБ/с
- PCI Express x4 - 4x250 МБ/с

Внешние интерфейсы блоков:
- МКИО (MIL-STD-1553B) - 80-90 КБ/с
- ДПК (ARINC 429) - 7.12 КБ/с
- FC-AE-ASM - 100 МБ/с
- AFDX - 10-12 МБ/с
- ARINC 818 (видео) - 70.1 МБ/с
- CAN (500 КГц) - 24.4 КБ/с
- Разовые команды (РК)

Необходимы специализированные инструментальные средства мониторинга и анализа информационных обменов.

Монитор – особый тип абонента: пассивный и без собственного адреса.

- Уровни информационного обмена (мониторинг информационного обмена):
  - Между блоками (канал)
  - Между модулями в составе блока (внутренняя шина)
  - Между функциональными задачами в рамках модуля (разделяемая память, очереди сообщений)

Мониторинг каналов/шин (задачи мониторинга различных уровней):

- По уровням протокола:
  - физический: проверка наличия и характеристик сигнала

    Способ анализа – осциллограф:
    - Физические характеристики сигнала
    - Правильность формирования слова/кадра, повторяемого по каналу (можно «непосредственно» увидеть)
    - …и даже значения данных (в повторяемых кадрах)
  - канальный:
    - проверка соответствия информационных слов и кадров стандарту канала
    - проверка ограничений реального времени на передачу слов/кадров
    - анализ активности абонентов канала

    Анализ канальных протоколов:
    - Виды представления данных
      - последовательность обменов
        - быстро обновляется, большой объём
        - фильтрация, поиск
        - постоянный мониторинг + запуск регистрации по событию
      - статистика обменов
        - по абонентам
        - по потокам данных (виртуальные каналы, метки слов, пары отправитель-получатель)
        - статистика ошибок

    Инструментальные средства анализа шин VME/PCI: (Silicon Control Inc, Curtiss-Wright Electronics systems / VMETRO, LeCroy Inc, Tektronix Inc, Гранит-ВТ)

    Способы визуализации результатов: таблица обменов, временная диаграмма.

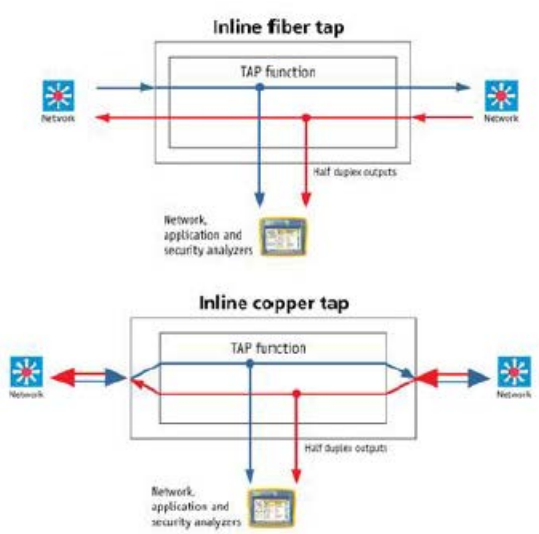
    (!!! Правильно ли я отнёс это к логическому уровню протокола?)
  - логический:
    - проверка соответствия параметров, передаваемых в полезной нагрузке, протоколу информационного взаимодействия

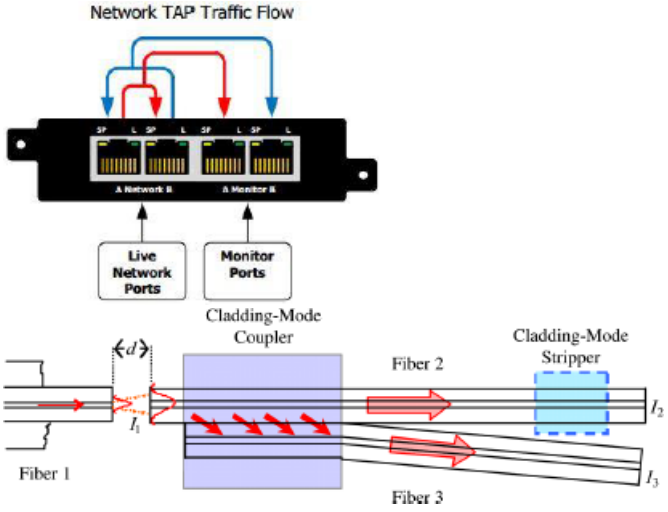
Варианты анализа:

- Оперативный: выявление и анализ проблем «на лету»
  - Автоматически проверяемые условия корректности
  - Визуальный анализ
- Анализ результатов регистрации
  - Большой объём
  - Начало регистрации «по событию»

Способы подключения монитора к каналам различной топологии:

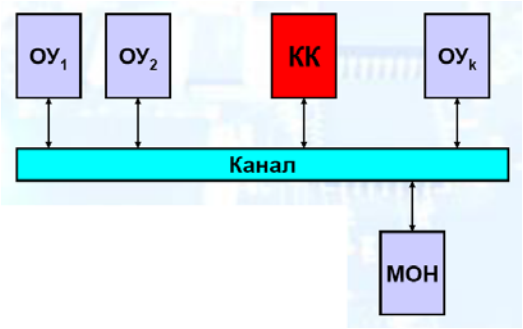
Точка-точка:



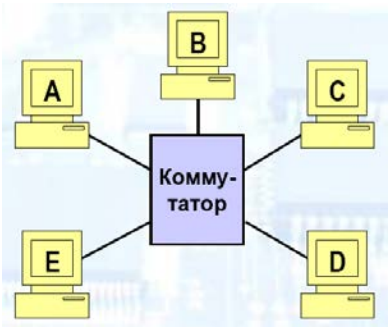


Часть оптического сигнала не отражается и уходит в другое волокно

Канал с общей шиной:



Коммутируемая среда:



Для коммутируемой среды есть 2 подхода:

- Слушаем выбранные линии как каналы «точка-точка»
  - – Коммутационная панель
  - – Каждый канал выведен с разрывом
  - – Варианты замыкания: напрямую или через разветвитель
  - – Громоздко…
- Порт мониторинга на коммутаторе
  - – «сливаются» только корректные кадры, воспринятые коммутатором
  - – Нарушение временных характеристик потока
  - – Возможное превышение пропускной способности порта
  - – Компактное решение

Архитектура средств мониторинга:

!!! ДПК – диспетчерский пункт круга?

Поддерживаемые протоколы (!!! Кто поддерживает?):
• MIL STD-1553B
• ARINC 429
• CAN
• Fibre Channel (оптика/данные)
• ARINC 818 (оптика/видео)

Анализатор позволяет следить за каналом (в частности, троированным), выдавать информацию о сообщении, статистику обмена, фильтровать обмен,

Анализ передаваемых данных:
• Проверяемые условия:
  – Обновление данных
  – Гладкость: $|A_i - A_{i-1}| < d$ (!!! Что это?)
  – Пороговое сравнение с эталоном
  – Сравнение по маске с эталоном или другим параметром
• Анализ графиков изменения параметров
• Интеграция с БД протоколов информационного взаимодействия

38. (Лекция 11б) Мониторинг межзадачного обмена в ИУС РВ. Инструментирование ПО ИУС РВ для выполнения мониторинга, негативное влияние инструментирования на точность наблюдений. Виды представления информации: снимки, трасса. Примеры системной информации, доступной для мониторинга.

!!! Инструментирование ПО ИУС РВ для выполнения мониторинга – что именно нужно ответить на этот вопрос?
!!! негативное влияние инструментирования на точность наблюдений – что именно нужно ответить на этот вопрос из билета

Инструментирование - возможность отслеживания или установления количественных параметров уровня производительности программного продукта, а также возможность диагностировать ошибки и записывать информацию для отслеживания причин их возникновения.

Мониторинг межзадачного обмена:
• Агент отладки
• **Запись трассы** + сброс в технологический порт
• «**Снимки**» интерфейсных переменных по внешнему запросу
• Мониторинг системной информации
• Показатели реального времени искажаются!

Мониторинг переменных:
• Инструмент мониторинга переменных позволяет получить доступ к значениям переменных и содержимому ячеек памяти в защищённых разделах памяти целевого вычислителя.
• Поддерживаются глобальные и статические переменные как пользовательских процессов, так и процесса ядра.
• Компонент позволяет получать и отображать значения, а также сохранять их в файле трассы.

Мониторинг системной информации:
• сведений о распределении памяти
• информации о каналах
• информации об условной переменной
• содержимого памяти
• информации о канале межмодульной передачи данных
• информации об очереди сообщений
• информации о мьютексе
• информации о порте с очередью сообщений

• информации о расписаниях и окнах
• информации о сегментах
• информации о семафоре
• сведений о состоянии потока управления
• информации о таймере
• статистики по использованию окон

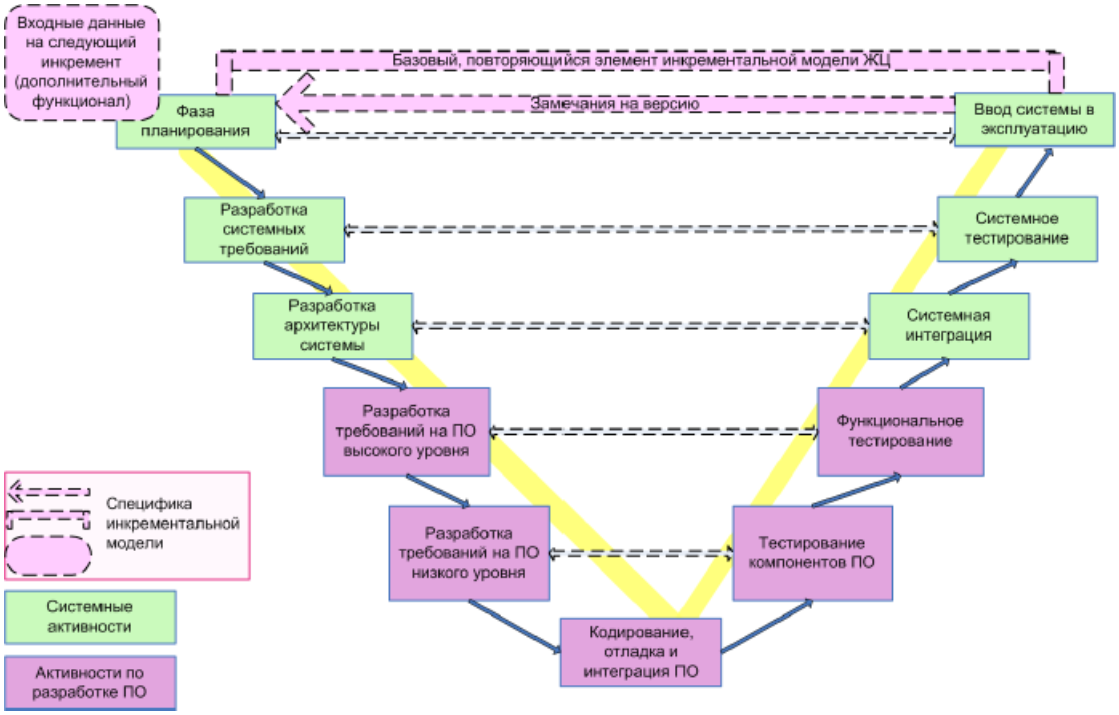
39. (Лекция 12) V-образный жизненный цикл ПО ИУС РВ. Основные процессы жизненного цикла по стандарту DO-178B.

Комплекс бортового оборудования:
средства отображения информации, органы управления
управление и контроль двигателей
среда передачи данных между компонентами КБО
системы навигации и связи, РЛС (радиолокация)
информационно-вычислительная система
<- сложная, неоднородная, распределённая вычислительная система

Специфика бортовых ИУС РВ:
- Высокая сложность
- Функционирование в реальном времени
    - Вычисления
    - Информационный обмен
- Требования
    - Функциональность => разнообразие
    - Надёжность
    - Реальное время
- Критичность
- Неоднородность
    - Каналы: точка-точка, шина, коммутатор; 12 kbps, 1 Mbps, 1 Gbps
    - Устройства: датчики, индикаторы, вычислители, органы управления, исполнительные устройства
    - Данные: аналоговые, цифровые; числовые массивы, видеопотоки

Бортовые ИУС РВ и их ПО требуют систематического подхода к проектированию, реализации и тестированию

Жизненный цикл ПО ИУС:
Фаза планирования
    Разработка системных требований
        Разработка архитектуры системы
            Разработка требований на ПО высокого уровня
                Разработка требований на ПО низкого уровня
                    Кодирование, отладка и интеграция ПО
                Тестирование компонент ПО
            Функциональное тестирование
        Системная интеграция
    Системное тестирование
Ввод системы в эксплуатацию



Процессы ЖЦ по стандарту DO-178B:

- Процесс планирования и управления проектом

Основные цели:

- определение основной методологии разработки
- написание планов разработки
- определение инструментов разработки
- определение и проведение необходимых тренингов
- определение методов мониторинга проекта
- выпуск необходимых шаблонов.

Основные артефакты:

- планы (разработки, тестирования ПО, конфигурационного управления, управления качеством и т.д.)
- отчеты о рецензировании планов
- графики
- стандарты
- шаблоны артефактов (документов) проекта
- протоколы совещаний
- официальные письма
- рабочая переписка (e-mail)
- метрики и индикаторы, отражающие состояние процессов и прогресс
- презентации
- списки мероприятий.

- Процесс разработки ПО

Подпроцессы:

- Разработка требований
- Проектирование
- Кодирование, отладка и интеграция

Для разработки требований и проектирования:

Основные цели:

Создание и контроль требований верхнего и нижнего уровня на разработку программного обеспечения.
Контроль полноты и непротиворечивости требований, прослеживаемости и тестируемости.

Основные артефакты:

- Спецификации требований в текстовом формате.

Если требования разрабатываются с использованием инструментальных средств, то модули в формате инструментального средства.

- Формальные спецификации (данные, приготовленные для автоматической генерации кода) в формате их редакторов:
    - формальные спецификации в формате HTML или текстовом формате
    - отчеты о рецензировании формальных спецификаций
    - описание графической информации в необходимом объеме
- Результаты автоматического контроля формальных спецификаций (матрица соответствия элементов формальных спецификаций и требований к ним с отчетом о рецензировании)
- сгенерированный код.

Для подпроцесса кодирования и интеграции:

Основные цели:

Разработка программного кода в соответствии со спецификациями, получение загрузочных файлов.

В процессе кодирования исходный код составляется на основе архитектуры ПО и требований нижнего уровня.

Формирование исполняемого кода, загрузка исполняемого объектного кода на целевой вычислитель для интеграции ПО и аппаратных средств.

Основные артефакты:

- код, разработанный вручную
- служебные файлы, необходимые для компиляции и сборки кода
- матрица покрытия требований кодом
- код, сгенерированный автоматическими средствами генерации.

Объектные файлы.

Загрузочный файл

- Интегральные процессы
    - Управление конфигурацией (конфигурационное управление, включая управление изменениями)

Основные цели:

Интегральный процесс, целью которого является:

- определение и идентификация продукта
- контроль выпуска и изменения версий в течение жизненного цикла
- контроль корректности и полноты версии,
- версионный контроль всех артефактов проекта

- определение конфигурационной структуры, обеспечение однозначной конфигурационной идентификации артефактов проекта.
- установление и проведение процесса управления изменениями.

Основные артефакты:

- поставки (файлы-архивы) и сопровождающие файлы-отчеты по каждому из процессов
- описание версии ПО
- сопровождающая документация
- носитель версии (оптический диск), если это оговорено - с необходимыми атрибутами (наклейками или другой идентифицирующей информацией)
- конфигурационный перечень артефактов проекта

- базы данных проблем - описания проблем (в формате, пригодном для импорта в средство управления изменениями (в текстовом, или ином удобном для чтения формате))
- протоколы совещаний по управлению конфигурацией

В случае наличия независимых баз данных у различных участников кооперации, предусмотреть наличие процесса и артефактов для поддержки синхронизации информации в этих базах данных.

- Верификация

Основные цели:

- интегральный процесс, целью которого является обнаружение и регистрация ошибок, которые могли появиться в процессе разработки ПО.

Эти цели достигаются через ряд мероприятий: написание тестовых процедур в соответствии с требованиями, проведение анализа кода и тестовых процедур, проведение формального тестирования, анализ полученных результатов, написание сообщений о проблемах, найденных в процессе различных верификационных активностях, создание и выпуск отчетов о проведенном тестировании.

Основные артефакты:

- отчет об автоматическом анализе кода, отчеты о рецензировании кода.
- тестовые процедуры и скрипты в формате редакторов тестов и текстовом формате, в случае использования инструментальных средств разработки требований и тестовых процедур – модули в формате инструментального средства
- отчеты о рецензировании тестов
- отчеты о рецензировании результатов тестирования
- матрица покрытия требований тестами с отчетом о её рецензировании
- матрица результатов тестирования с отчетом о её рецензировании
- реестр со списком всех верификационных документов для версии ПО.

- Управление качеством

Основные цели:

Интегральный процесс, целью которого является контроль качества продукта и контроль соблюдения процессов.

Основные артефакты:

- отчеты (протоколы) проверки качества продукции
- отчеты (протоколы) проверки процессов
- планы проверки

- Сертификационное взаимодействие (в данном курсе не рассматривается)

- Поставка ПО заказчику:

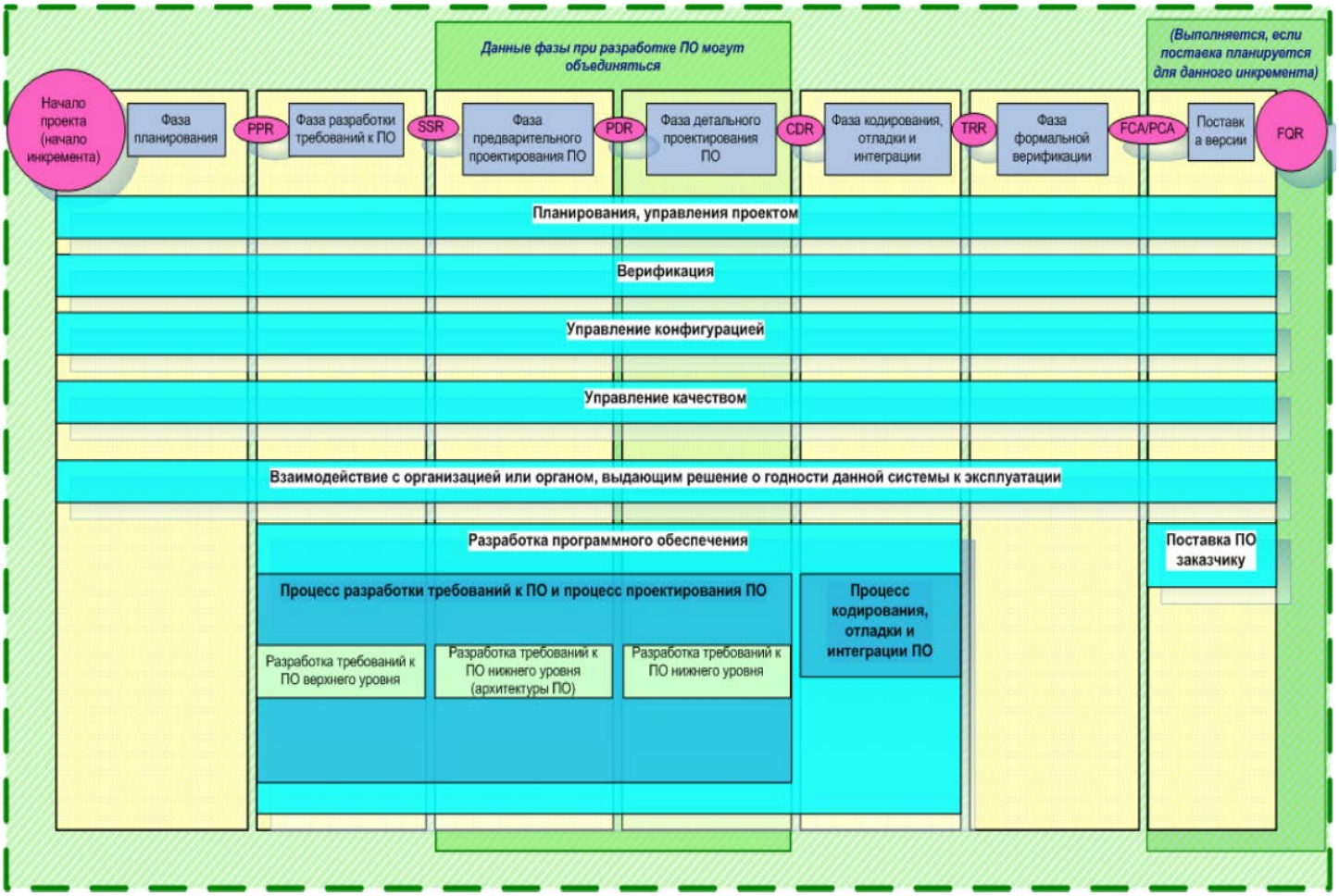
Основные цели:

Передача разработанного продукта заказчику.

Основные артефакты:

- описание версии ПО
- финальная версия поставляемого ПО
- заключение о годности версии ПО к натурным испытаниям на целевой системе.

Соотношение фаз и процессов ЖЦ (жизненного цикла) ПО:



40. (Лекция 12) Фазы жизненного цикла ПО ИУС РВ. Соотношение фаз и процессов жизненного цикла. Основные вехи жизненного цикла ПО по стандарту DO-178B и их место в рамках V-образного жизненного цикла.

Фазы ЖЦ:

- планирование

Основные активности внутри фазы:

- определение методов и инструментов для производства программного продукта в процессе написания планов разработки ПО
- написание стандартов разработки и создание необходимых шаблонов.

Применяемые инструментальные средства:

Инструменты, позволяющие:

- осуществлять планирование (в т.ч. создавать графики разработки)
- создавать планы разработки и тестирования
- создавать шаблоны разработки
- отслеживать ход выполнения проекта
- осуществлять коммуникацию, взаимодействие различных участников проекта.

- разработка требований к ПО

Основные активности внутри фазы:

- анализ входных требований
- написание требований верхнего и нижнего уровня на разработку ПО
- определение бортовых интерфейсов, структуры и протоколов среды обмена
- построение матрицы прослеживаемости между требованиями различного уровня
- написание формальных спецификаций
- начало написания тестовых процедур, соответствующих требованиям верхнего уровня.

Применяемые инструментальные средства:

Инструменты, позволяющие:

- создавать требования
- создавать связи между требованиями
- разрабатывать и поддерживать интерфейсы между различными компонентами ПО
- проводить необходимые обзоры артефактов
- создавать тестовые процедуры
- разрабатывать формальные спецификации

- проектирование (дизайн) ПО

Основные активности внутри фазы:

- определение архитектуры ПО, декомпозиция ПО, декомпозиция распределения памяти
- определение структуры ПО с учетом требований реального времени
- написание (детализация) тестовых процедур в соответствии с уточнениями и детализацией требований.
- построение предварительной матрицы покрытия требований тестовыми процедурами

Применяемые инструментальные средства:

Инструменты, позволяющие:

- создавать и поддерживать описание архитектуры ПО
- прослеживать связи между требованиями различного уровня
- задавать и прослеживать соответствие тестовых процедур и требований
- осуществлять необходимые обзоры артефактов
- создавать матрицу покрытия требований тестовыми процедурами

- кодирование отладка и интеграция

Основные активности внутри фазы:

- генерация кода из формальных спецификаций
- разработка ручного кода.
- интеграция всех разработанных программных компонент в единый модуль.
- загрузка программного модуля на целевой вычислитель.
- отладка разработанного кода и, параллельно, тестовых процедур.
- в конце фазы происходит «заморозка» программного кода и тестовых процедур.

Применяемые инструментальные средства:

Инструменты, позволяющие:

- генерировать код на языке высокого уровня из формальных спецификаций
- разрабатывать ручной код
- проводить компиляцию исходного кода
- проводить компоновку двоичного кода
- проводить загрузку на целевой вычислитель и интеграцию
- проводить отладку на целевых вычислителях
- выполнять интеграционные тесты с имитацией данных на входах целевых вычислителей
- проводить коррекцию программного кода и тестовых процедур в случае обнаружения ошибок
- разрабатывать и выпускать необходимую документацию

- верификация

Основные активности внутри фазы:

- проведение прогона разработанных тестовых процедур на «замороженной» версии ПО (тестирование).
- анализ результатов тестирования.
- создание сообщений о проблемах по результатам тестирования.
- финализация матрицы покрытия.

Применяемые инструментальные средства:

Инструменты и средства, позволяющие:

- осуществлять прогон тестируемого ПО на целевом вычислителе
- выполнять тесты с имитацией данных на входах целевых вычислителей
- собирать результаты тестирования, в т.ч. формировать протоколы тестирования
- проводить необходимые обзоры артефактов
- оформлять сообщения о проблемах
- модифицировать матрицу покрытия

- подготовка финальной поставки

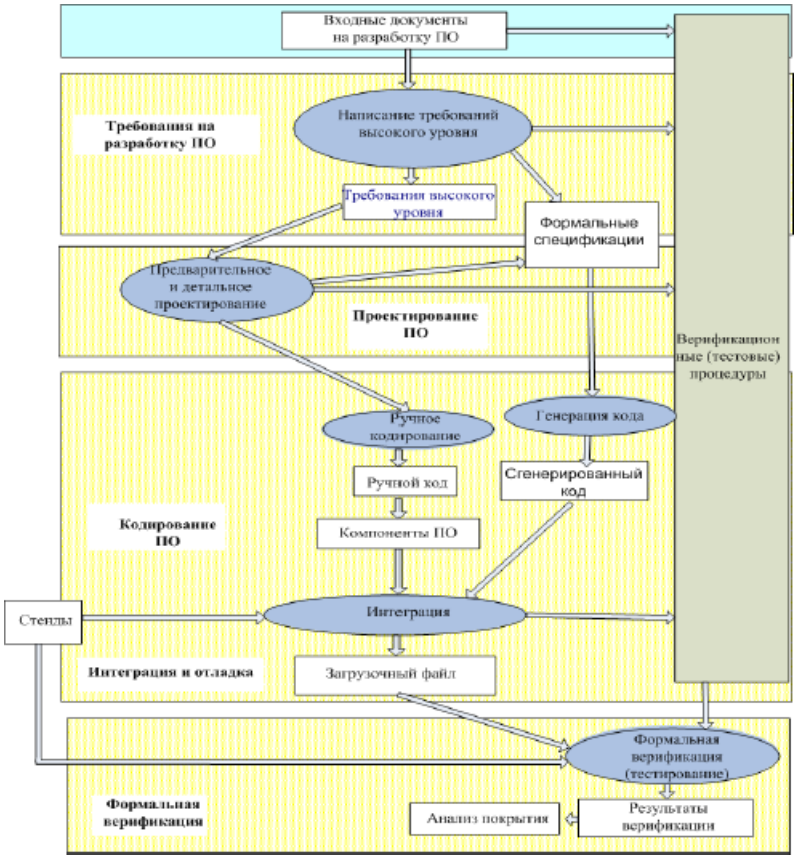
Основные активности внутри фазы:

- выпуск итоговых документов по составу версии

Применяемые инструментальные средства:

- инструменты, позволяющие создавать, модифицировать и выпускать итоговые документы проекта.

Активности по фазам ЖЦ ПО:

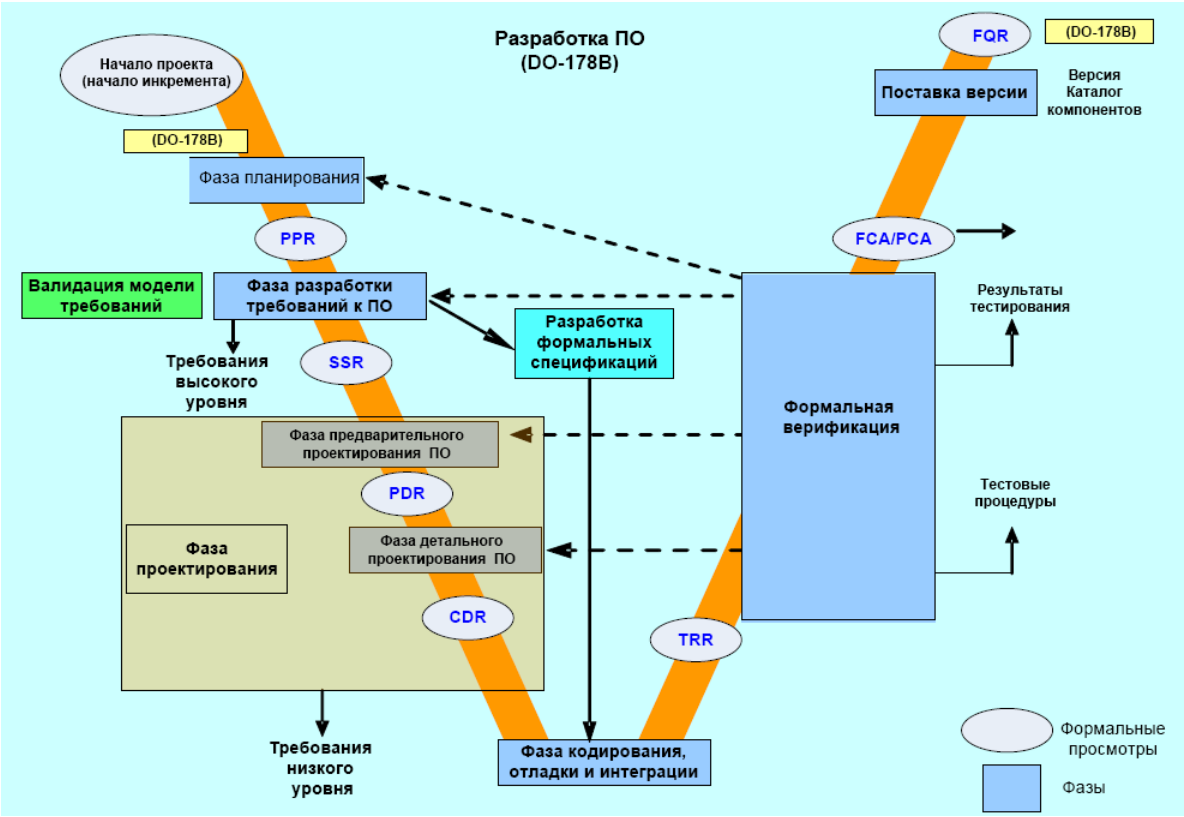


Вехи ЖЦ ПО:

Вехи		Аббревиатура	Описание
Русское название	**Английское название**		
Обзор планов	Plan Process Review	PPR	Одобрение, официальный выпуск планов разработки
Обзор спецификаций	Software Specification Review	SSR	Одобрение, официальный выпуск требований на разработку ПО и начало фазы предварительного дизайна
Предварительный обзор архитектуры	Preliminary Design Review	PDR	Одобрение, официальный выпуск предварительного дизайна (архитектуры ПО) и начало активностей по определению детального дизайна
Критический обзор архитектуры	Critical Design Review	CDR	Одобрение, официальный выпуск детального дизайна ПО
Обзор готовности тестов	Test Readiness Review	TRR	Начало выполнения формальной верификации (формального тестирования)

Функциональный конфигурационный аудит (Функциональный аудит программных конфигураций)	Functional Configuration Audit	FCA	Проверка того, что элементы конфигурации отвечают всем функциональным требованиям, в том числе требованиям по производительности. FCA представляет обзор функционирования элементов, не только для обеспечения соответствия спецификации, но и для выявления не заданных требованиями функциональных характеристик.
Физический конфигурационный аудит (Физический аудит Программных конфигураций)	Physical Configuration Audit	PCA	PCA обеспечивает гарантию того, что поставляемые результаты физически соответствуют перечисленным в документации пунктам и представлены в поставляемой версии.
Формальный квалификационный обзор	Formal Qualification Review	FQR	Успешное завершение этих аудитов может быть обязательным требованием для финального фиксирования версии.

Фазы и вехи на V-цикле:



41. (Лекция 12) Средства поддержки разработки требований, примеры требований к ИУС РВ. Средства версионного и конфигурационного контроля. Древовидная структура версий. Средства отслеживания проблем и изменений. Жизненный цикл сообщения о проблеме.

Средства поддержки разработки требований:

- Функциональность бортового ПО описывается десятками тысяч требований (системных и собственно, к ПО)
- Необходимая функциональность:
    - создание и хранение требований, отслеживание истории
    - связывание требований с версиями документов и ПО
    - прослеживаемость требований на:
        - Низкоуровневые требования
        - Формальные спецификации
        - Код
        - Тесты
- Примеры средств: IBM DOORS, Borland CaliberRM, SyBase PowerDesigner

Примеры требований к ВИУС РВ:

BCS.INIT.SELF - Встроенная самопроверка БЦВМ

После включения питания БЦВМ должна выполнять начальную встроенную самопроверку. Взаимодействие по каналам КБИ при этом не выполняется. Продолжительность встроенной самопроверки не менее 2 сек. и не более 8 сек. По результатам встроенной проверки зажигается светодиод исправности БЦВМ.

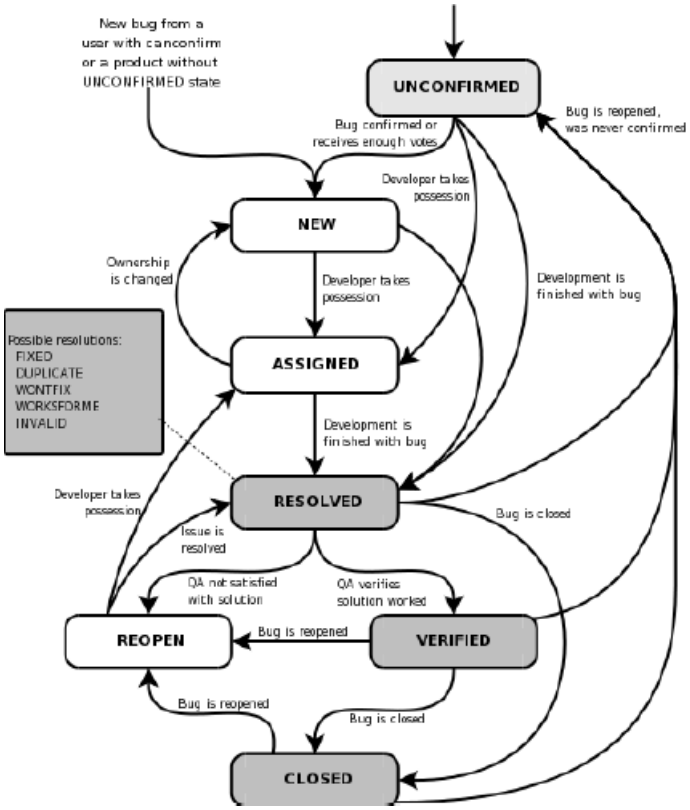
Средства версионного/конфигурационного контроля:

- При разработке ПО формируется множество документов, группируемых в версии
    - Промежуточные («инженерные») версии
    - Официальные версии
- Версия = вся совокупность документов
    - Требования, спецификации, код, тесты, …
    - Часть конфигурации борта
- Необходимая функциональность:
    - Версионирование совокупности документов (в т.ч. атомарность изменений)
    - Поддержка ветвей истории
    - Поддержка групповой разработки, в т.ч. разграничения доступа
    - Обновление документов в реальном времени
- Примеры: CVS, Subversion, git, IBM ClearCase

Средства отслеживания проблем и изменений:

- В процессе разработки ПО:
    - Дефекты и ошибки: зарегистрированы, проанализированы и устранены
    - Изменения: зарегистрированы, утверждены и реализованы
- Необходимая функциональность:
    - Поддержка структуры продукта и процесса
    - Настраиваемый формат сообщения о проблеме
    - Настраиваемый ЖЦ сообщения, поддержка согласования
    - Поддержка групповой разработки, втч разграничения доступа
    - Интеграция со средствами управления версиями
- Примеры: Bugzilla, Trac, IBM ClearQuest

Жизненный цикл сообщения о проблеме:



42. (Лекция 12) Средства поддержки сопряжения подсистем ПО ИУС РВ. Средства автоматизации проектирования индикационных форматов. Средства проектирования алгоритмов бортового ПО. Отладка ПО ИУС РВ на реальном блоке ИУС. Общие требования к построению технологической цепочки средств поддержки жизненного цикла ПО ИУС РВ.

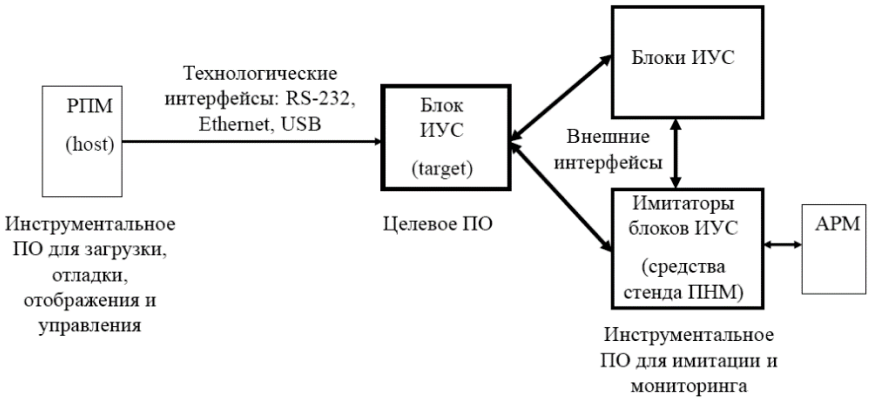
Средства поддержки сопряжения подсистем ПО:
- Средства автоматизации проектирования бортовых интерфейсов
    - Балансировка загрузки каналов
    - Формирование набора сообщений
    - Построение расписаний обмена (канал с централизованным управлением)
    - Построение системы виртуальных каналов (сеть на основе коммутаторов)
- Средства автоматизации интеграции ПО
    - Использование унифицированных структурных компонентов ПО
    - По управлению: расписание выполнения СКПО
    - По данным:
        - БД информационных связей СКПО
        - Автоматическое формирование описания интерфейсов ПО (буфера, каналы, сообщения) для конфигурирования ОСРВ

Средства автоматизации проектирования индикационных форматов:
- Индикационный формат = набор графических элементов + правила поведения
- Необходимая функциональность:
    - Редактирование в графической форме, WYSIWYG
    - Поддержка библиотеки элементов
    - Поддержка автономного тестирования
    - Генерация кода в формате для целевого устройства
- Примеры: SCADE Display, VAPS, САПР ИФ

Средства проектирования алгоритмов:
- Алгоритмы бортового ПО хорошо формализуются
    - Потоковая обработка данных
    - Конечный автомат
- Проектирование/описание на формальном уровне позволяет формализовать проверку алгоритма
- Необходимая функциональность:
    - Поддержка обоих видов формального представления
    - Графическое описание
    - Тестирование и пошаговая отладка на уровне модели
    - Верификация на основе формальных методов
    - Сертифицированный кодогенератор
- Примеры: Telelogic Rhapsody, SCADE Suite, Simulink

Технологический комплекс разработки программ:
- Поддержка целевой ОС и аппаратной архитектуры
- Поддержка редактирования/компиляции/компоновки программ
- Поддержка отладки
- В среде инструментального ПК
- В среде эмулятора целевой системы
- На целевой системе
- Поддержка отладки в реальном времени
- Мониторинг внешних каналов связи
- Мониторинг внутренних данных программы
- Мониторинг системных шин БЦВМ

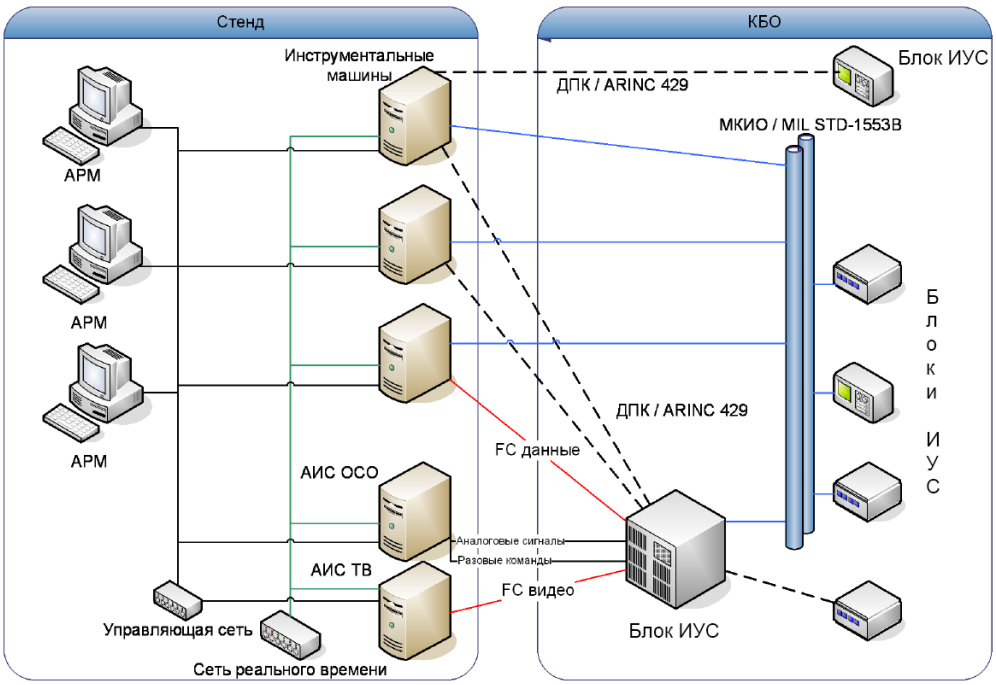
Отладка ПО ИУС РВ на реальном блоке:



Средства поддержки верификации и тестирования бортового ПО:

- Тестирование на целевой платформе
- Недопустимость инструментирования
- Тестирование через каналы бортовых интерфейсов
- Тестирование требований реального времени
- Интерактивное тестирование индикационных форматов
- Многоэтапное тестирование
- Сопровождение интеграции подсистем КБО
- Необходимая функциональность:
- Поддержка стандартов бортовых интерфейсов
- Многомашинные конфигурации
- Выполнение тестов в реальном времени
- Автоматическое и интерактивное тестирование
- Пакетный режим
- Формирование отчётов, прослеживаемость требований
- Примеры средств: Rational Test RealTime, VectorCast, средства разработки ЛВК

Архитектура стенда тестирования ИУС:



Принцип построения технологической цепочки разработки бортового ПО:

- Сквозная поддержка ЖЦ, включая активности на всех фазах
- Сопряжение «вход-выход» с обеспечением совместимости форматов данных
- Особое внимание на переходы между фазами

- Требуется фиксация выходных артефактов

Цепочка средств разработки бортового ПО: