Моделирование и формальная верификация микроархитектуры

Александр Готманов

Strategic CAD Labs
Intel Corporation (Москва)

Формальная верификация СБИС

- Структурная модель автомата с конечной памятью
  - Доказательство эквивалентности
  - Верификация моделей
  - Алгебраическое исполнение (STE)

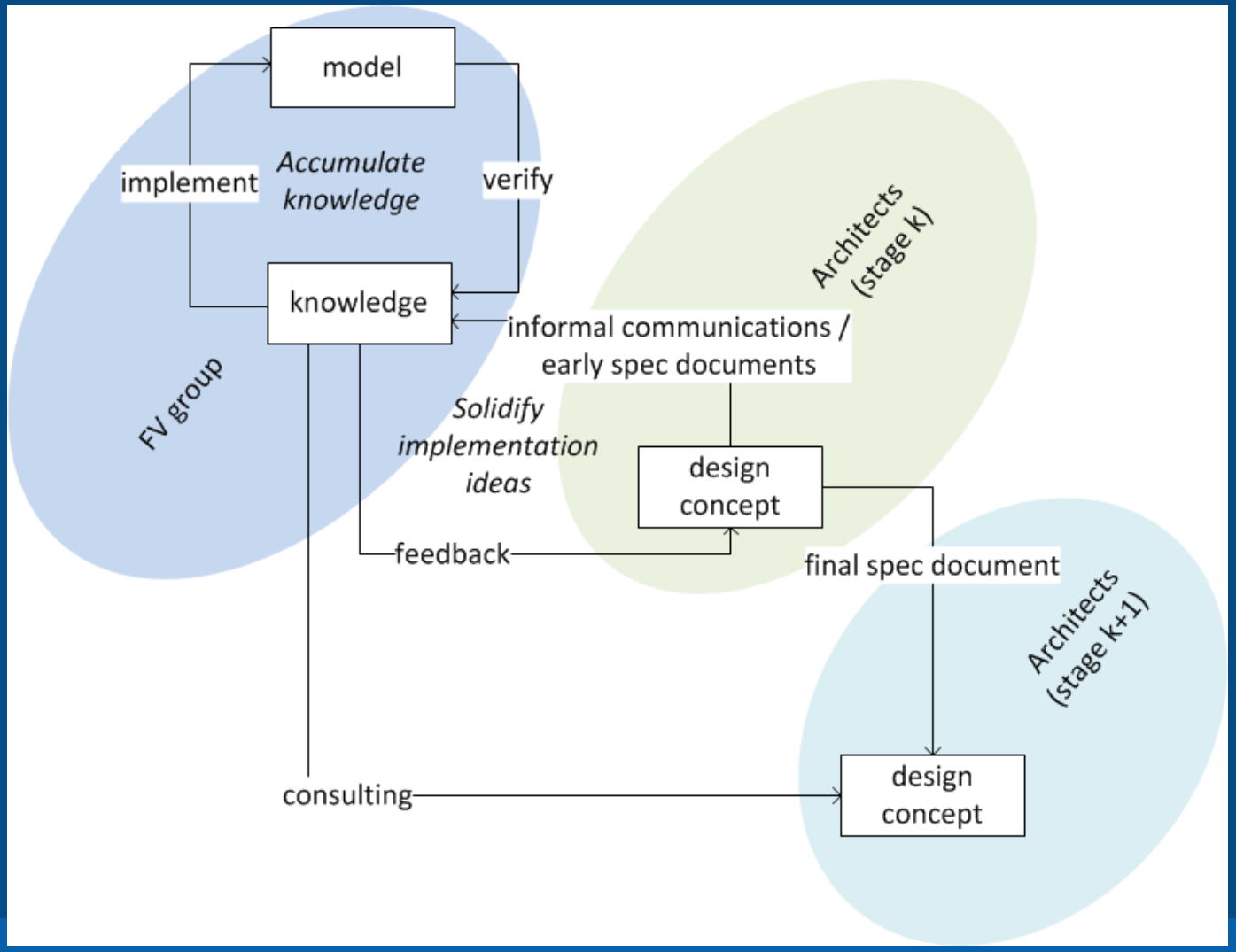
- Модель распределенного алгоритма
  - Верификация моделей
  - Дедуктивный вывод
  - Абстрагирование и усиление

Верификация архитектуры / 1

- Используется один из видов моделей распределенных алгоритмов
  - Murphi-подобные языки
  - TLA
  - xMAS
  - Специальные языки

- Особенности
  - Написание, поддержка и верификация модели – отдельная задача, требующая времени и умения
  - Модель создается на ранней фазе проектирования
  - Необходимо ясное *соглашение о сотрудничестве* с архитекторами: цели, протокол, терминология, распределение ролей

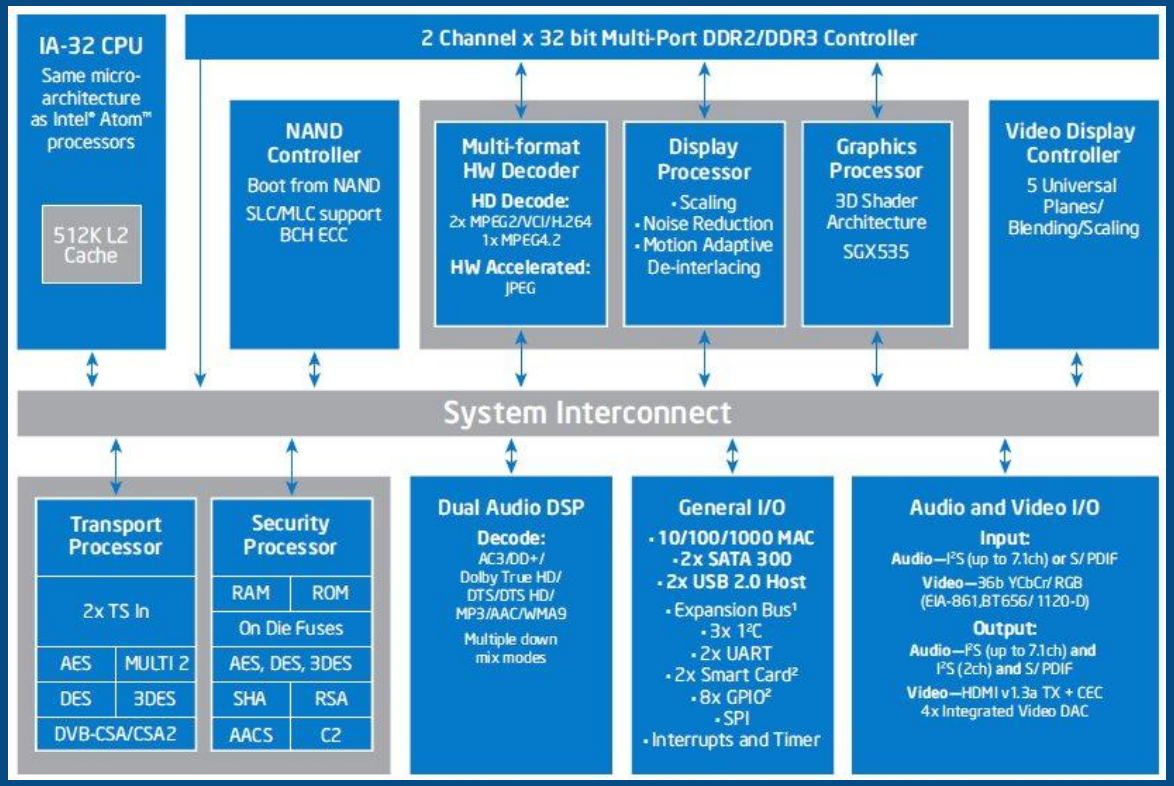
Верификация архитектуры / 2



Robert Beers “Pre-RTL Formal Verification: An Intel Experience”

Коммуникационная фабрика

Решает задачу взаимодействия устройств на кристалле
(“Транспортная логика”)



Ядра, кэши, IP блоки, память и т.д.

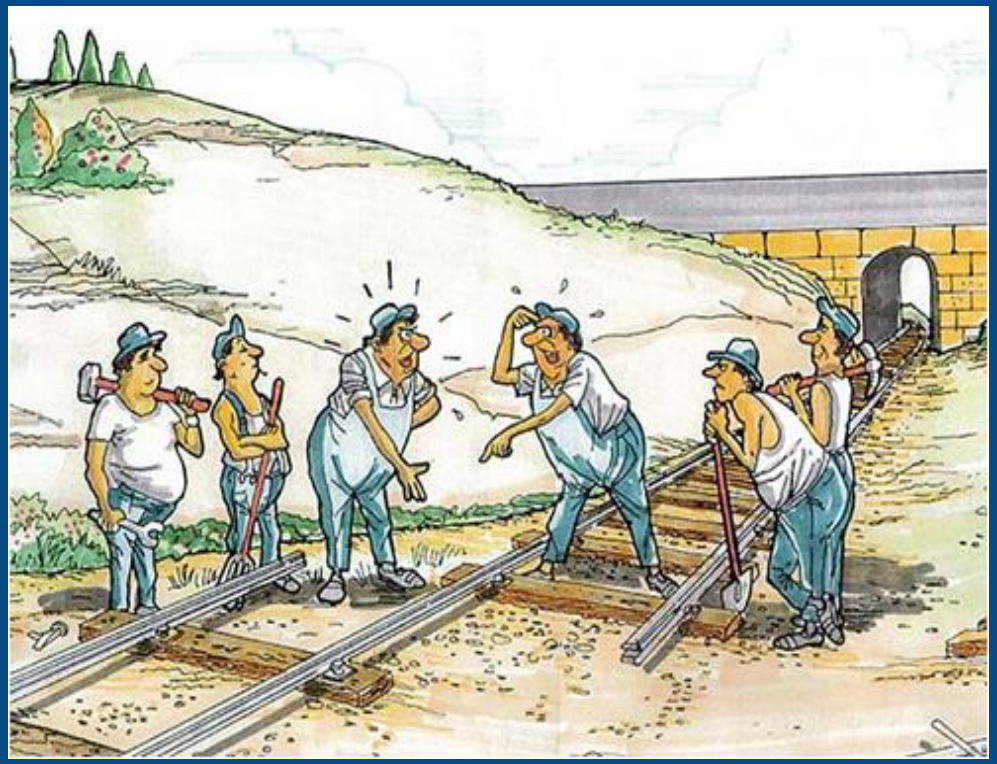
Разнообразие микроархитектур и топологий.

Intel® Atom® processor CE4100

Трудности проектирования

Причина: распределенное проектирование распределенного алгоритма

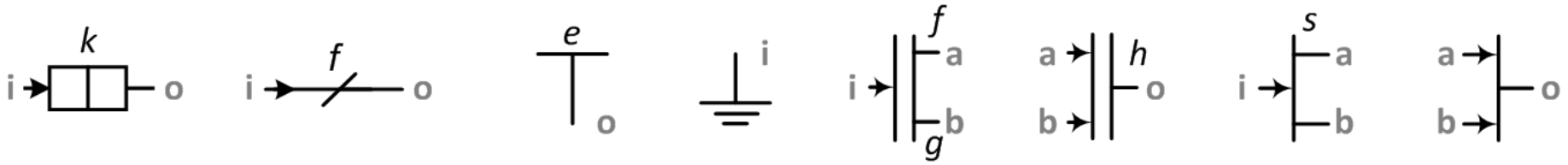
- Типы ошибок
  - разделение ресурсов
  - зависания
  - нарушения упорядочения
- Отсутствие ясных *априорных* требований к компонентам системы
- Позднее обнаружение проблем
  - на этапе сборки системы
  - RTL, тестовый чип



Как обнаруживать ошибки в начале проектирования?

Язык моделирования xMAS

Модель = синхронная *сеть* передачи *пакетов*, построенная из простых *примитивов*

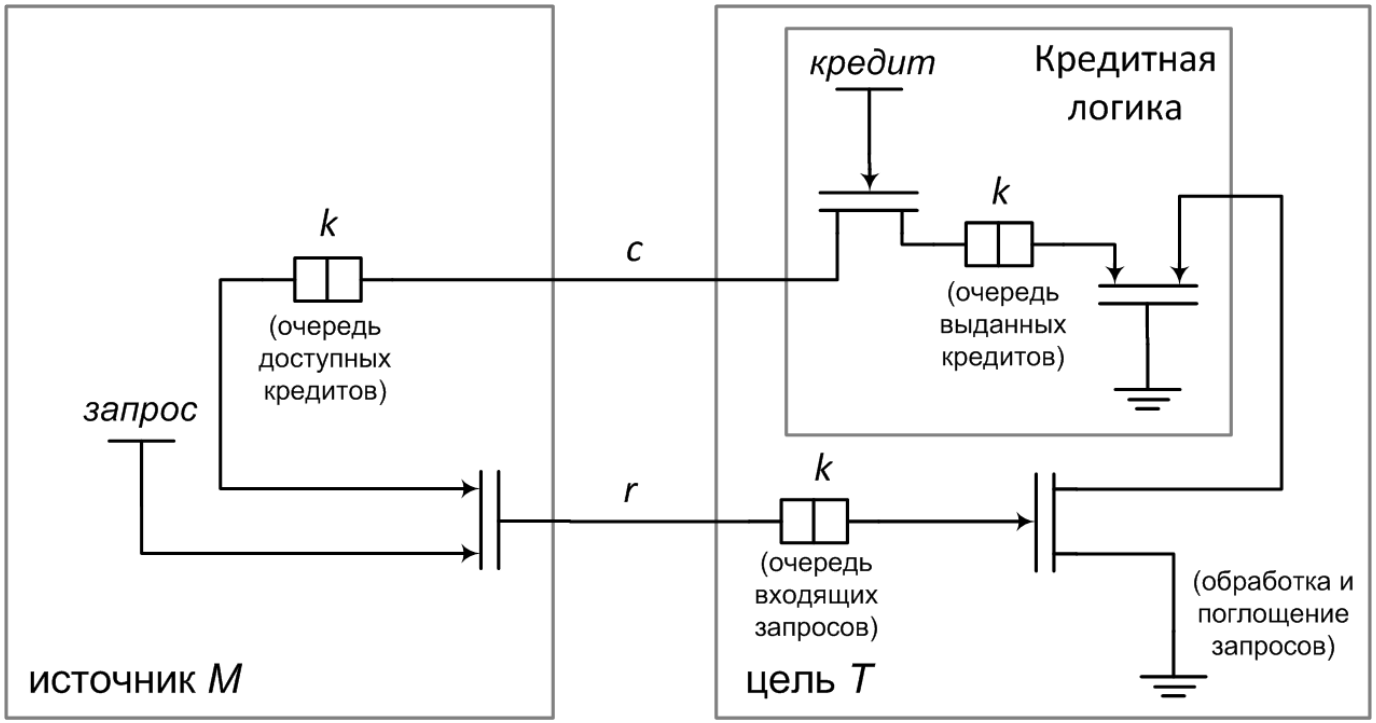


очередь (queue) | преобразование (function) | исток (source) | сток (sink) | форк (fork) | барьер (join) | ветвление (switch) | слияние (merge)

Примитивы соединяются *каналами*

Ориентация на моделирование транспортной логики

Модель управления потоком

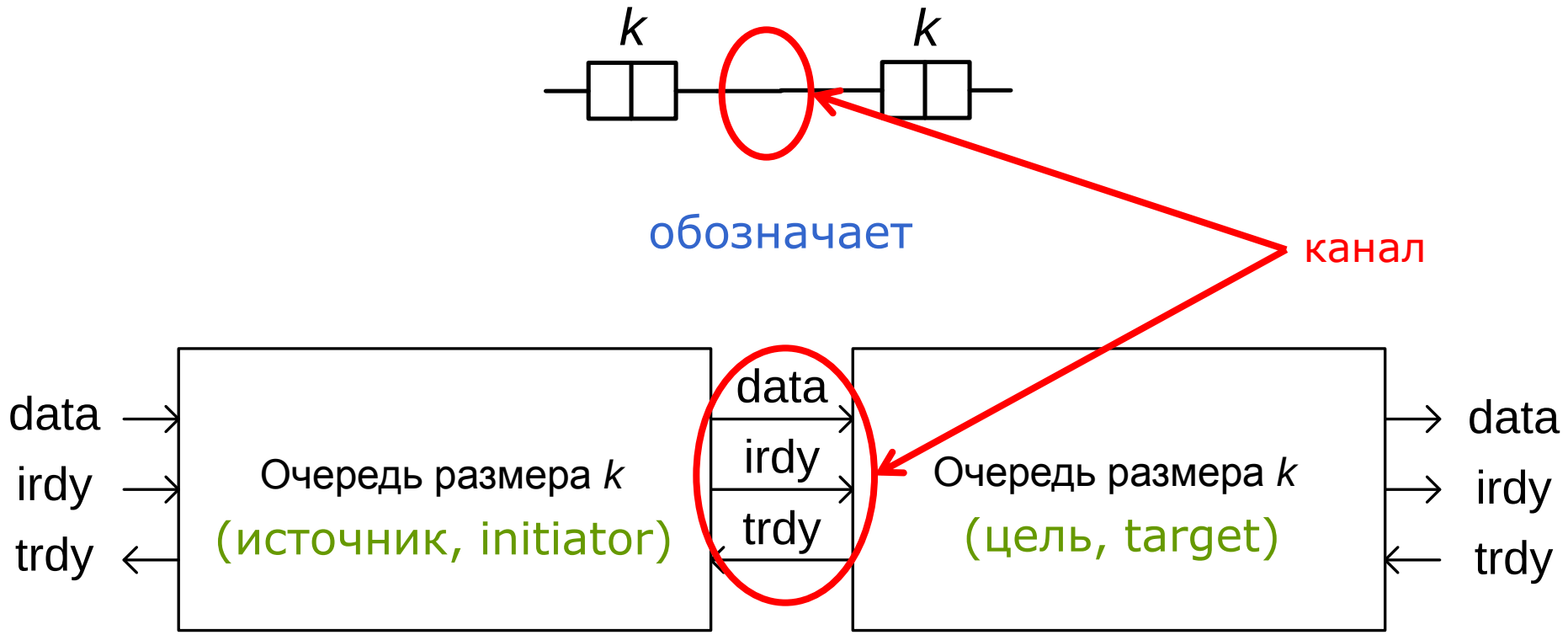


Посылка запроса возможна только при наличии доступного кредита, что гарантирует место в очереди входящих запросов.

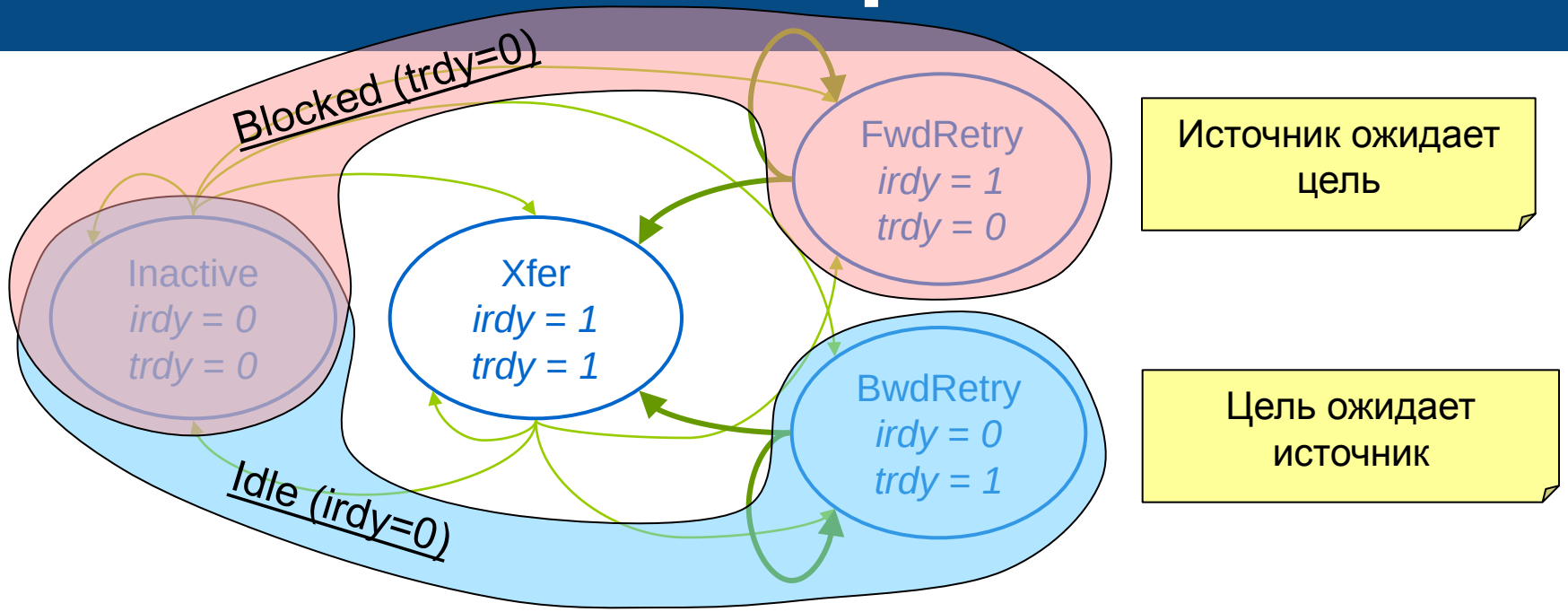
Семантика

Примитив = синхронный модуль (ср. модель Verilog с одним тактовым сигналом)

Канал реализует простой протокол типа “запрос-ответ”



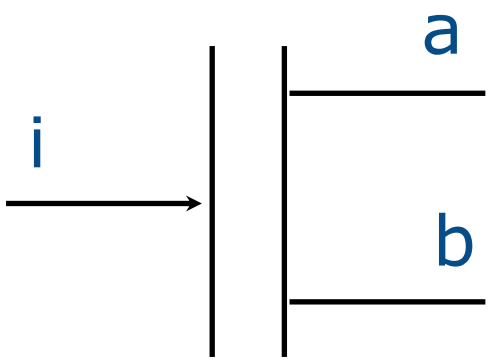
Устойчивость протокола



- Каждый канал устойчив
  - Примитивы построены так, чтобы обеспечить устойчивость
- Устойчивость исключает “сложные” сценарии зависаний на канале

Состояние retry сохраняется до момента передачи.

Синхронные уравнения



Уравнения

a.irdy **:=** i.irdy **and** b.trdy

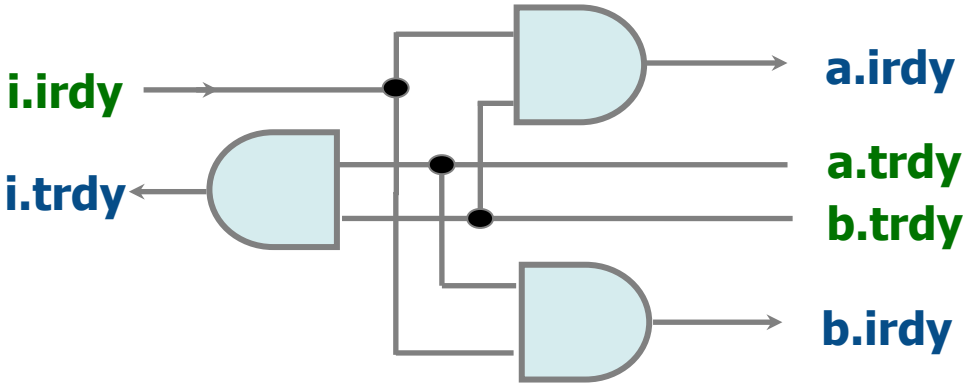
b.irdy **:=** i.irdy **and** a.trdy

i.trdy **:=** a.trdy **and** b.trdy

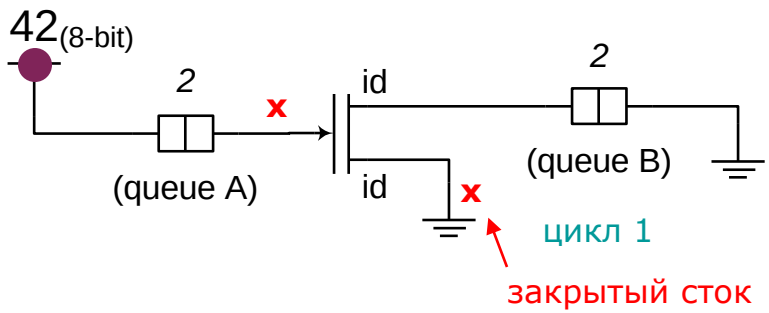
a.data := i.data

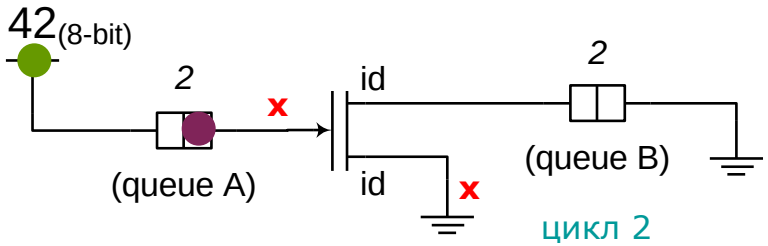
b.data := i.data

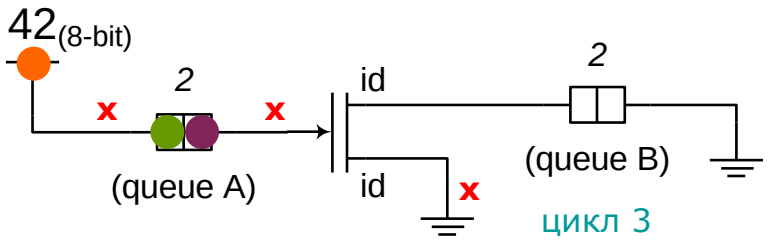
Логическая сеть

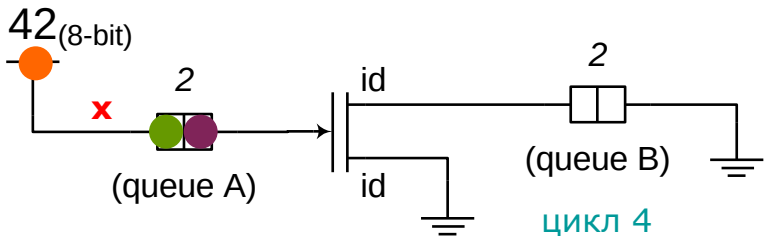


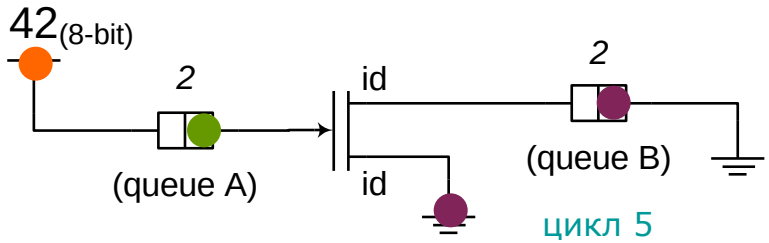
Пример

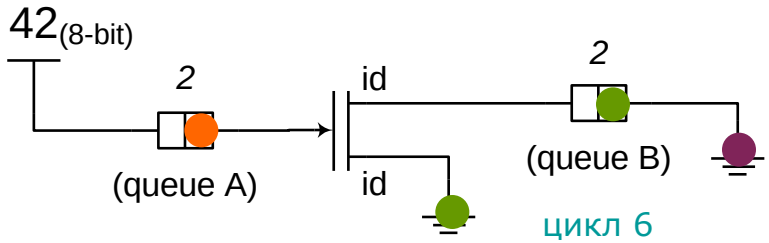

42(8-bit)
2
x
id
2
(queue A)
id
(queue B)
x
цикл 1
закрытый сток


42(8-bit)
2
x
id
2
(queue A)
id
(queue B)
x
цикл 2


42(8-bit)
x
2
x
id
2
(queue A)
id
(queue B)
x
цикл 3


42(8-bit)
2
x
id
2
(queue A)
id
(queue B)
цикл 4

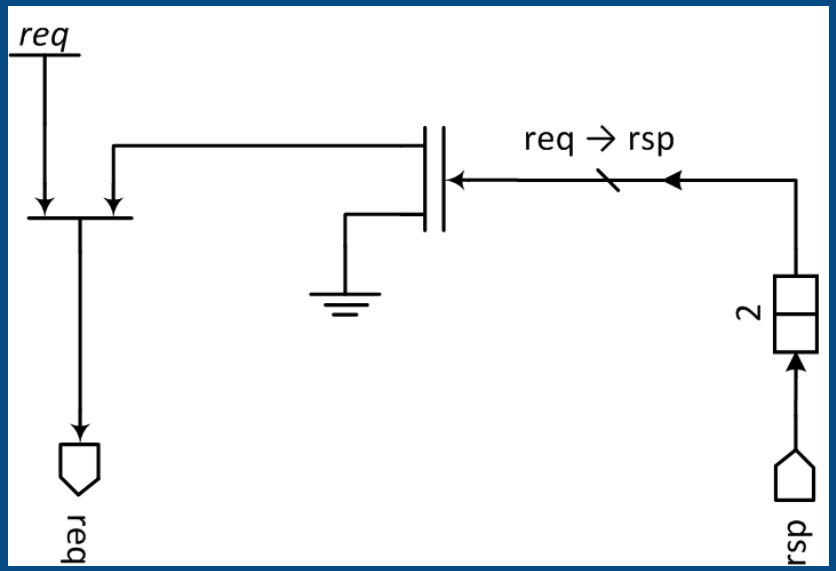

42(8-bit)
2
id
2
(queue A)
id
(queue B)
цикл 5


42(8-bit)
2
id
2
(queue A)
id
(queue B)
цикл 6

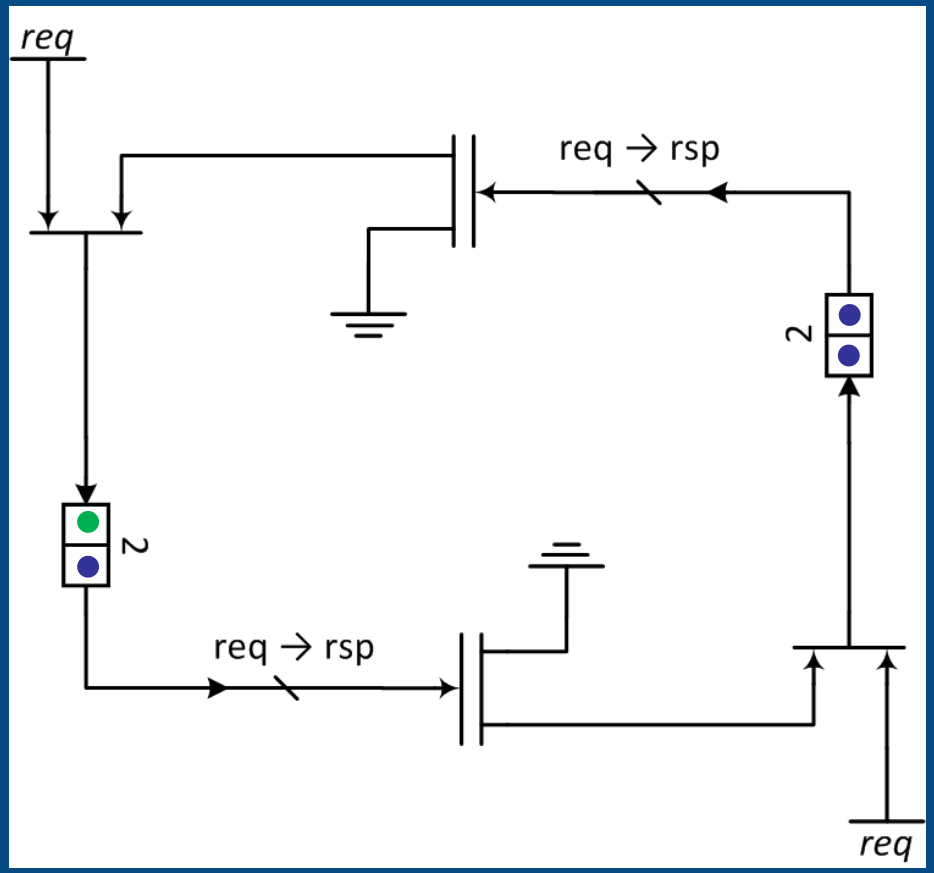
Разнообразие моделей

- Компоненты
  - FSM (автоматы)
  - Логика упорядочения
  - Таблицы
  - Конвейеры
  - Переключатели пакетов
  - Кэши
  - …
- Топологии
  - Шины
  - Кольца
  - Решетки
- Агенты
  - Протокольные зависимости
  - IP блоки, ядра
  - Контроллеры памяти

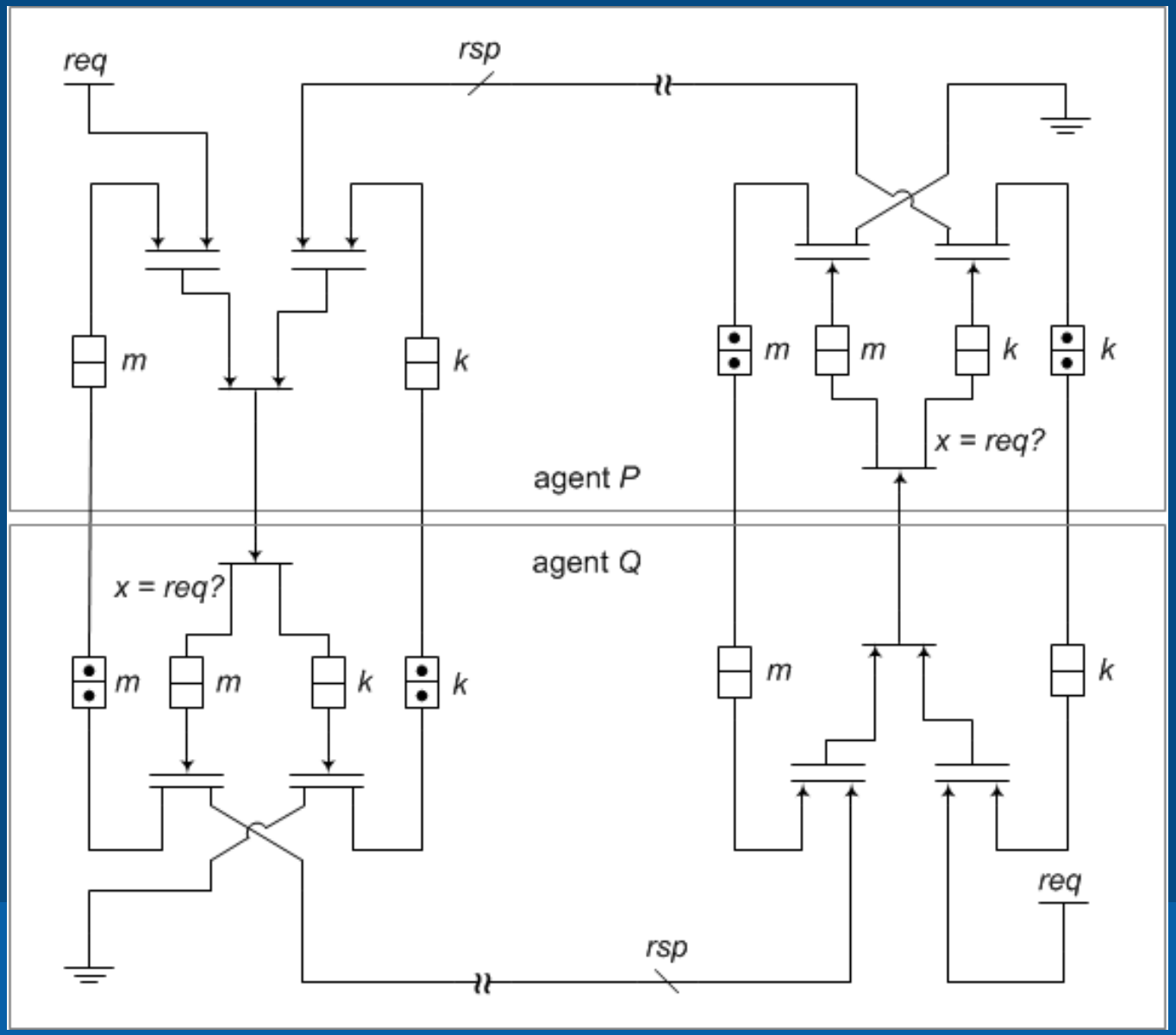
Модель взаимодействия агентов / 1



Модель взаимодействия агентов / 1



Модель взаимодействия агентов / 2



Методы верификации

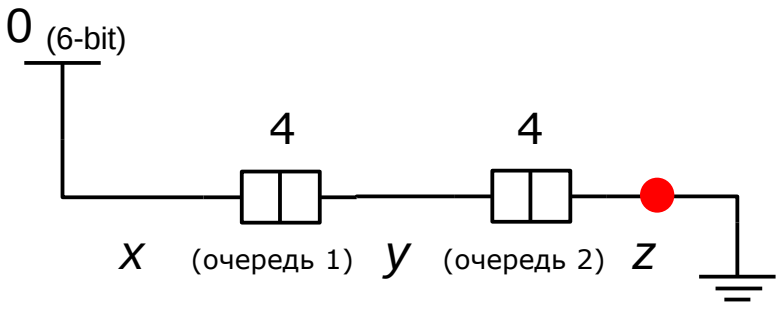
- Структурный анализ модели позволяет существенно улучшить сходимость

- Типы свойств
  - Допустимые значения данных на канале (safety)
  - Линейные инварианты (safety)
  - Отсутствие зависаний (liveness)

Свойства данных / 1

Данные на канале обладают некоторым свойством $P(x)$

— например, все входящие пакеты имеют правильный адрес назначения



Пример:

$$\mathbf{G}\,(\mathrm{z.irdy} \rightarrow (\mathrm{z.data} = 0))$$

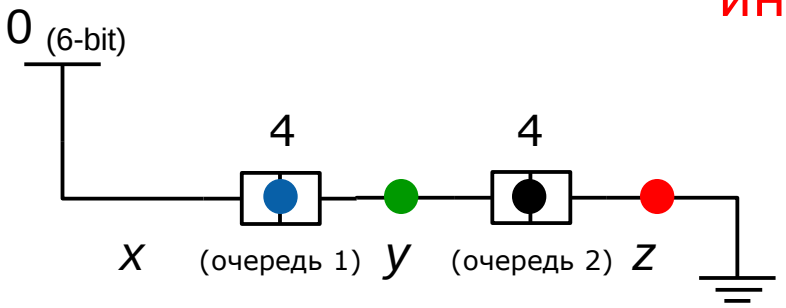
— Доказательство может представлять серьезную трудность для стандартных алгоритмов верификации

— На практике модели содержат десятки/сотни очередей и примитивов

Обозначения LTL, **F** = “однажды”, **G** = “всегда”

Свойства данных / 2

Структурный анализ модели позволяет усилить инвариант



Исходное свойство:

$$\mathbf{G}\,(\mathrm{z.irdy} \rightarrow (\mathrm{z.data} = 0))$$

Шаг 1: $\mathbf{G}\,(\mathrm{used_j} \rightarrow (\mathrm{mem_j} = 0))$

Всякий пакет в очереди 2 имеет значение 0

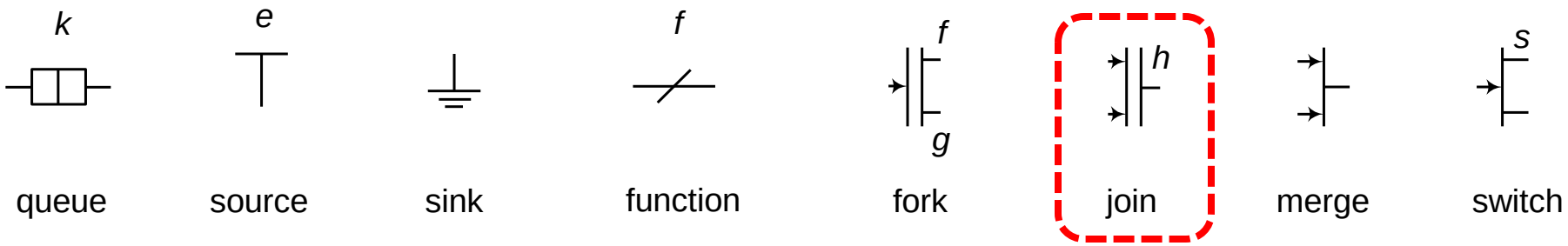
Шаг 2: $\mathbf{G}\,(\mathrm{y.irdy} \rightarrow (\mathrm{y.data} = 0))$

Шаг 3: $\mathbf{G}\,(\mathrm{used_j} \rightarrow (\mathrm{mem_j} = 0))$

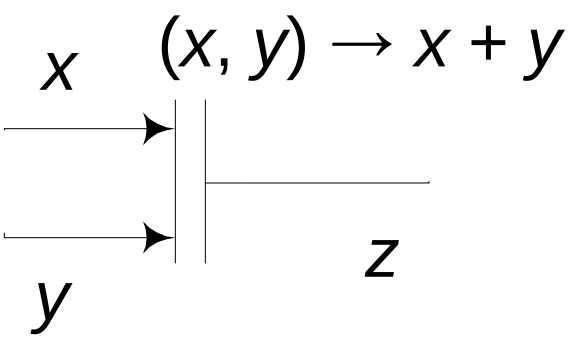
(То же для очереди 1)

Совокупность построенных инвариантов индуктивна.

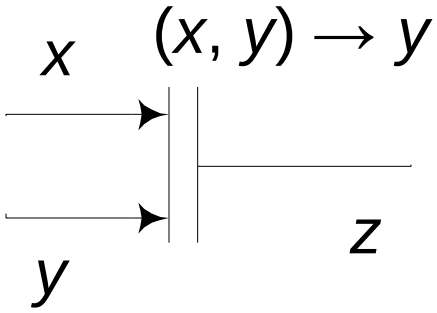
Применимость метода



Сложный случай



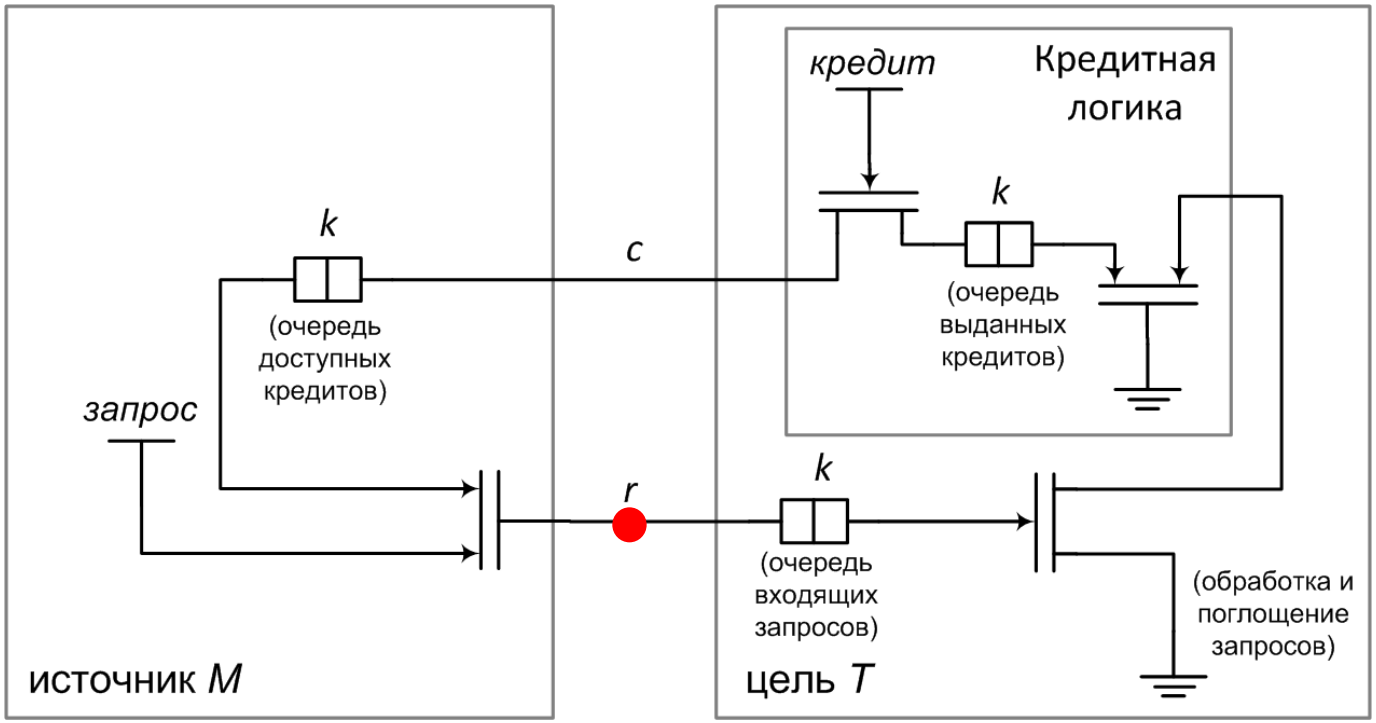
Простой случай



G (*z*.irdy ⇒ (*z*.data = 42))

(used for synchronization)

Линейные инварианты / 1



Посылка запроса возможна только при наличии доступного кредита, что гарантирует место в очереди входящих запросов.

Свойство неблокирующей передачи: $\mathbf{G}\,(\mathrm{r.irdy} \rightarrow \mathrm{r.trdy})$

Линейные инварианты / 2

Запишем уравнения сохранения пакетов в окрестности каждого примитива (ср. уравнения Кирхгофа для тока)

$$\lambda_e = \lambda_f = \lambda_r$$

$$\lambda_u = \lambda_t = \lambda_v$$

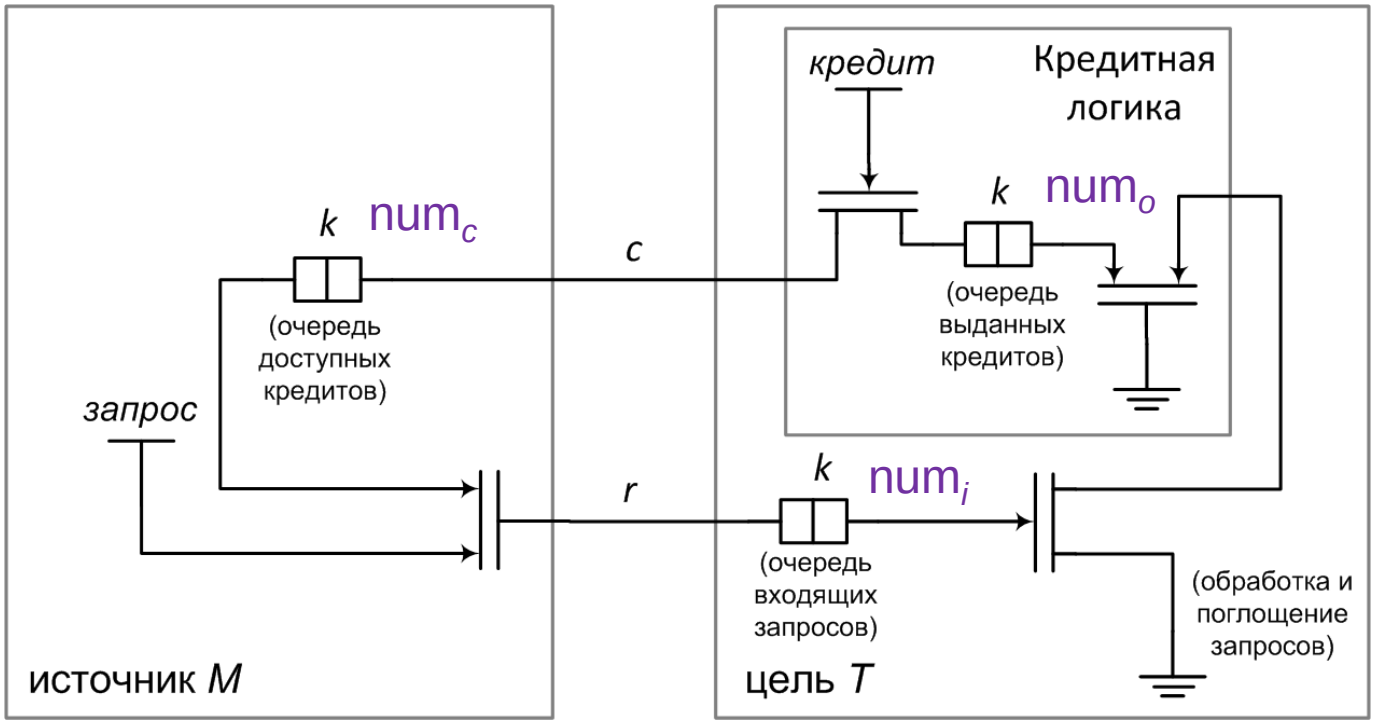
$$\lambda_s = \lambda_w = \lambda_z.$$

$$\lambda_p = \lambda_n = \lambda_s.$$

$$\lambda_r = \mathrm{num}_i + \lambda_p$$

$$\lambda_t = \mathrm{num}_c + \lambda_e$$

$$\lambda_v = \mathrm{num}_o + \lambda_w$$



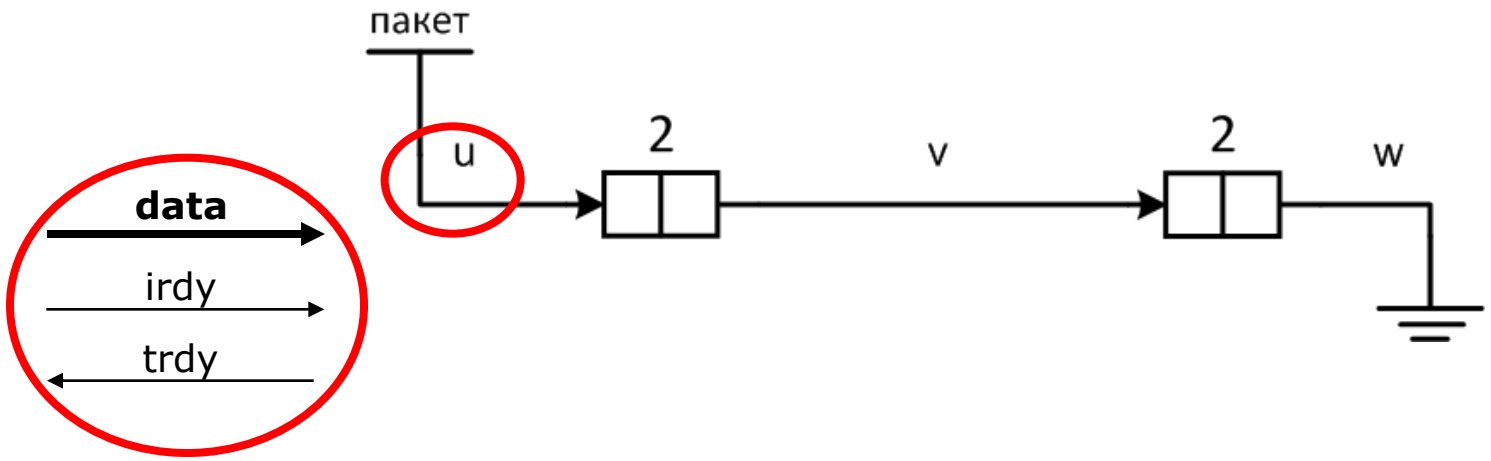
Индуктивное уравнение баланса: $\mathbf{G}\,(\mathrm{num_i} + \mathrm{num_c} = \mathrm{num_o})$

Линейные инварианты / 3

- Практическая реализация
  - Подсчет разных сортов пакетов
  - Вывод неравенств
  - Автоматический вывод 10-100 инвариантов

- Существенное ускорение сходимости алгоритмов верификации
  - Пр. 1: проверка маршрутизации на решетке 6x6 (~5мин)
  - Пр. 2: проверка неблокирующих свойств в коммуникационной фабрике (~5сек вместо 12ч)

Зависание



- Зависание определяется для отдельного канала
- $\mathbf{Dead}(u)$ = "на канале *u* появляется пакет, но выход *u* всегда остается закрытым"
- $\mathbf{Dead}(u) = \mathbf{F}(\mathrm{u.irdy} \cdot \mathbf{G}\,\neg \mathrm{u.trdy})$
- $\mathbf{Live}(u) = \neg\mathbf{Dead}(u) = \mathbf{G}(\mathrm{u.irdy} \rightarrow \mathbf{F}\,\mathrm{u.trdy})$

Ограничения справедливости

- Ограничения для справедливого стока:

- **Fair** = конъюнкция всех ограничений

- Требуется доказать: $\mathbf{Fair} \rightarrow \mathbf{Live}(u)$

Методы решения задачи

- Верификация моделей
  - Поиск циклического пути в пространстве состояний, обладающего определенными свойствами
  - Плохо масштабируется; не применима к моделям xMAS с 10-ми очередей
- Док
  - Тр
    ф

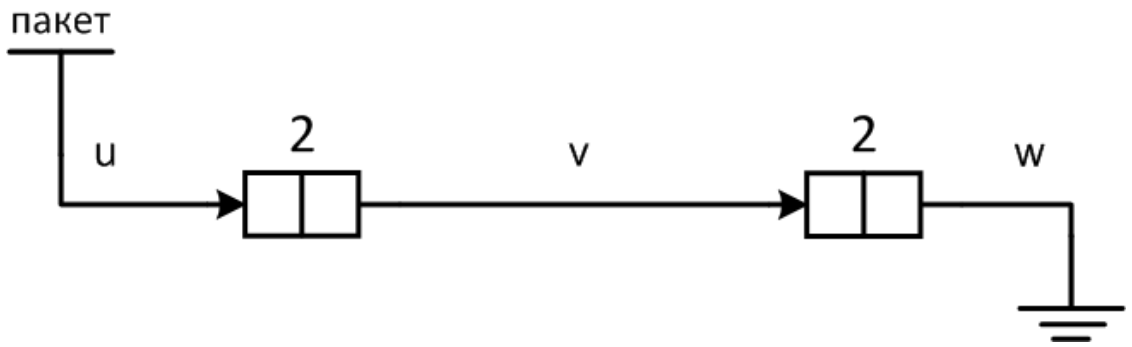
Пример
Модель обмена сообщениями с упорядочением
75 примитивов
(24 очереди)

Верификация моделей (ABC)
Ограниченное док-во,
29 тактов за 1 час.

во

Структурный анализ
Полное док-во за 4 сек.

Стационарные переменные



$\mathbf{Idle}(u) = \mathbf{FG}(\neg u.irdy)$ "u стремится к состоянию idle $(u.irdy = 0)$"

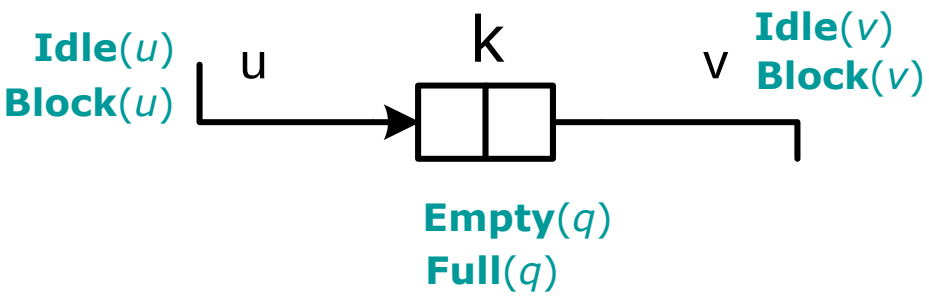
$\mathbf{Block}(u) = \mathbf{FG}(\neg u.trdy)$ "u стремится к состоянию block $(u.trdy = 0)$"

$\mathbf{Dead}(u) = \neg\mathbf{Idle}(u) \cdot \mathbf{Block}(u)$

$\mathbf{Fair} = \neg\mathbf{Block}(w)$

Состояние модели на бесконечности задается значениями стационарных переменных.

Стационарные уравнения



$$\begin{cases} u.trdy = (q.num < k) \\ v.irdy = (q.num > 0) \\ \dots \end{cases}$$

$\mathbf{Full}(q) = \mathbf{FG}(\mathrm{q.num} = k)$ "q стремится к состоянию full $(q.num = k)$"

$\mathbf{Empty}(q) = \mathbf{FG}(\mathrm{q.num} = 0)$ "q стремится к состоянию empty $(q.num = 0)$"

$\mathbf{Block}(u) = \mathbf{Full}(q)$

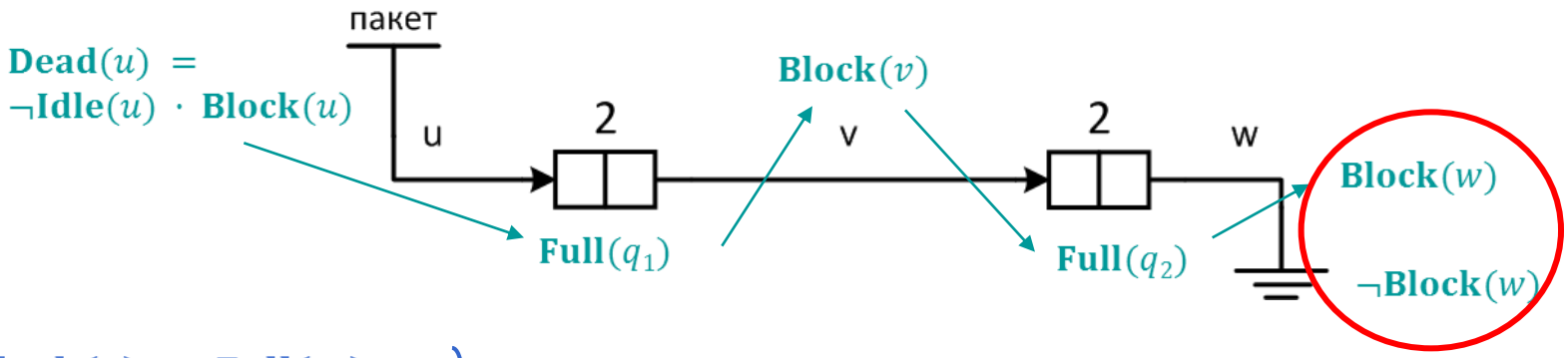
"u стремится к block $\Leftrightarrow$ q стремится к full"

$\mathbf{Full}(q) \rightarrow \mathbf{Block}(v)$

"если q стремится к full, то v стремится к block"

Каждый примитив ограничивает возможные значения стационарных переменных в своей окрестности.

Ход доказательства



$$\left.\begin{array}{l} \mathbf{Block}(u) = \mathbf{Full}(q_1) \\ \mathbf{Full}(q_1) \rightarrow \mathbf{Block}(v) \\ \mathbf{Block}(v) = \mathbf{Full}(q_2) \\ \mathbf{Full}(q_2) \rightarrow \mathbf{Block}(w) \end{array}\right\} \mathbf{Struct}_M$$

(стационарные уравнения)

$$\mathbf{Fair} = \neg\mathbf{Block}(w)$$

(ограничения справедливости)

$$\mathbf{Dead}(u) \cdot \mathbf{Struct}_M \cdot \mathbf{Fair} = 0$$ как формула исчисления высказываний

Не существует выполнения модели, в котором было бы выполнено $\mathbf{Dead}(u)$.

Для доказательства пригоден любой SAT-солвер (алгоритм проверки выполнимости формул).

Обзор метода

$\mathbf{Dead}(u)$ как $\neg\mathbf{Idle} \cdot \mathbf{Block}$

+

$\mathbf{Struct}_M$ стационарные уравнения, отражающие семантику примитивов

+

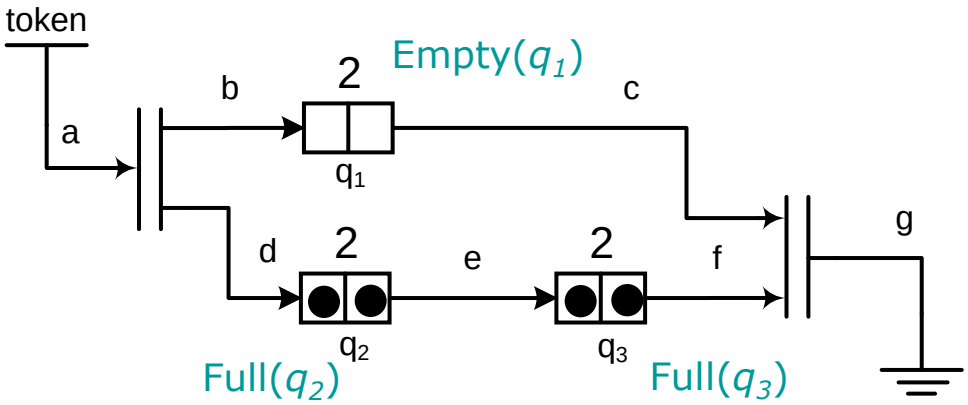
$\mathbf{Fair}$ ка

- **UNSAT** = Зависа
- **SAT** = Контрпри

Для применения на практике также требуются

- Дополнительные инварианты, ограничивающие значения стационарных переменных (порождаются автоматически)
- Дополнительные переменные для описания зависимости от данных

Добавление инвариантов



Dead(a)
Fair = ¬Block(g)

Система $\mathbf{Dead}(a) \cdot \mathbf{Struct}_M \cdot \mathbf{Fair}$ совместна

- Решение – достижимое исполнение с пустой очередью q_1 и полными очередями q_2, q_3

- Это решение противоречит инварианту:

$$q_1.num = q_2.num + q_3.num$$

$$(q1.num = 0) \rightarrow ((q2.num = 0) \cdot (q3.num = 0))$$

Эксперименты (доказательство)

Модель	**NPrims / NQueues**	**Время (ABC)**	**Время (структурный анализ)**
Две очереди	4 / 2	<1c	<1c
SMF = модель обмена сообщениями	57 / 20	N/A	2c
SMF с упорядочением	75 / 24	N/A	4c
IOF = “южный мост”	493 / 97	N/A	90c
CMF = “северный мост”	710 / 88	N/A	690c

Заключение

- xMAS – язык, ориентированный на моделирование транспортной логики
- Графическая нотация
- Структурный анализ для быстрой верификации свойств safety
- Автоматизированное доказательство отсутствия зависаний для моделей с 100-ми очередей