Языки описания схем

(mk.cs.msu.ru → Лекционные курсы → Языки описания схем)

Блок K1

Кое-что ещё:
Протоколы передачи данных
Протокол UART (общее описания)

Лектор:
Подымов Владислав Васильевич

E-mail:
valdus@yandex.ru

ВМК МГУ, 2019, осенний семестр

Вступление

Очевидные факты, известные каждому программисту:

- Любая хоть сколько-нибудь полезная программа взаимодействует с *окружением* и обменивается с ним данными
- Любой хоть сколько-нибудь полезный компьютер взаимодействует с окружением и обменивается с ним данными

То же справедливо и для цифровых схем:

Любая хоть сколько-нибудь полезная аппаратура взаимодействует с окружением и обменивается с ним данными

Вступление

Пересылка данных с одного компьютера на другой —
это известное и не очень сложное дело:

- Взгляд пользователя:
  - Запускаешь программу отправки, в нужном месте вводишь данные, нажимаешь кнопку “Отправить”
  - Запускаешь программу приёма и ждёшь, пока она скажет “Данные приняты” и покажет их
- Взгляд “высокоуровневого” программиста
  - Изучаешь интерфейс отправки данных в высокоуровневом языке программирования, записываешь команду “Отправить данные”, собираешь, выполняешь
  - Изучаешь интерфейс приёма данных в <...>, записываешь команду “Принять и сохранить данные”, собираешь, выполняешь

Вступление

Пересылка данных с одного компьютера на другой —
это известное и не очень сложное дело:

- Взгляд “низкоуровневого” программиста
  - Изучаешь способ настройки и использования места,
    в которое требуется отправить данные
    (инициализация, открытие сессии,
    создание и наполнение буфера отправки, синхронизация, ...),
    настраиваешь, пишешь подходящий набор команд для посылки,
    собираешь, выполняешь
  - <...> из которого требуется принять данные <...>,
    собираешь, выполняешь

- Взгляд разработчика машинного кода: ?
  (*об этом будет пара слов в другом курсе*)

- Взгляд разработчика схемы: ?? — об этом можно поговорить сейчас

Вступление

Способы пересылки данных от одной аппаратуры к другой
заметно отличаются от способов пересылки между программами

Программа запускается

- в рамках операционной системы,
  “скрывающей” многие низкоуровневые программные детали, или
- как минимум, **на готовом процессоре**,
  скрывающем **схемные** детали выполнения кода

Цифровая схема существует сама по себе
без какого бы то ни было процессора (*процессор — это и есть схема*)

Скрыть детали пересылки данных одной схемой
может только другая схема —
и эту *другую схему* всё равно кто-то должен разработать

Обсудим такие *другие схемы*

Вступление

USB, PCI, IDE, SATA, Thunderbolt, Ethernet, DVI, HDMI,
LTP, COM, AHB, JTAG, UART, SPI, I2C, ..., ..., ...

Как это всё устроено, и есть ли в этом всём что-то общее?

Всё перечисленное — это (*в числе прочего*) протоколы передачи данных: своды правил, соглашений и требований, описывающих обмен информацией между произвольными устройствами

Эти протоколы существенно различаются, но при этом содержат схожие части *схемного уровня*:
описание того, как устройство A должно управлять цифровыми сигналами в реальных проводах в реальном времени, чтобы устройство B на другом конце проводов могло принять значение x, которое хочет послать A

Остановимся подробнее на схемных частях самых простых протоколов, *обозначив при этом характерные черты и многих других протоколов*

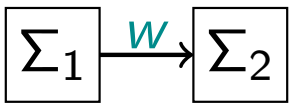
UART: общее описание

Начнём обсуждение с самого простого[1] универсального протокола:[2]

UART (Universal Asynchronous Receiver-Transmitter)

Начнём описание протокола немного издалека:
представим себе цифровые схемы Σ_1, Σ_2,
одна из которых (Σ_1) хочет передать другой (Σ_2) число x

Очевидный факт:
для схемы Σ_1, Σ_2 должны быть соединены хотя бы одним проводом



Универсальность: чем меньше проводов требуется для передачи данных, тем выше шанс, что в схему можно добавить столько лишних контактов

Ограничимся одним проводом w, и попробуем передать по нему число x

---

[1] Имеется в виду "простота" почти во всех смыслах, какие только можно придумать. Оттенки "простоты" будут коротко обозначаться дальше.

[2] Строго говоря, это семейство протоколов, устроенных примерно одинаково

UART: общее описание

Простота: разрешим схемам заранее договориться о чём угодно, кроме того, какое число хочет передать Σ_1

Больше простоты:
считаем, что Σ_1 и Σ_2 — это синхронные схемы со сбросом

Замечание. “Простота” синхронности в том,
что разработать истинно-асинхронную схему довольно непросто:
настолько, что методы разработки таких схем, хотя и существуют,
но ещё не вошли в список обязательных знаний
“обычного” разрабтчика схем

UART: общее описание

Вспоминаем физику: схемам Σ_1, Σ_2 следует договориться о потенциале, обозначающем логический ноль

Иначе потенциал, выставляемый в w схемой Σ_1,
может иметь некорректную логическую трактовку в Σ_2
или даже вывести эту схему из строя

Как это сделать: например,

- подключить схемы к одному источнику питания, или
- добавить второй провод между схемами,
  послать в него из Σ_1 значение 0 и
  физически отключить Σ_2 от своего источника питания[1]

Замечание. Такое "выравнивание" потенциалов есть во всех протоколах, содержащих схемную часть, и всегда неявно подразумевается

[1] Внезапно появился намёк на значение *высокого импеданса* (z).
Для простоты проигнорируем этот намёк.

UART: общее описание

Простота: позволим схемам заранее договориться
об особенном числе ν — частоте протокола

При этом не будем заставлять тактовые сигналы схем
осциллировать на частоте ν

$T = \frac{1}{\nu}$ — период протокола

Больше простоты:
позволим схемам заранее договориться о **ширине** передаваемого числа

Для определённости считаем, что передаётся число ширины 8

UART: общее описание

Схема Σ_1 может и не иметь намерения передать какое бы то ни было число

В этом случае заставим схему Σ_1 выставить в w неизменное значение 1:

Представим себе, что у схемы Σ_1 появилось намерение передать число через w

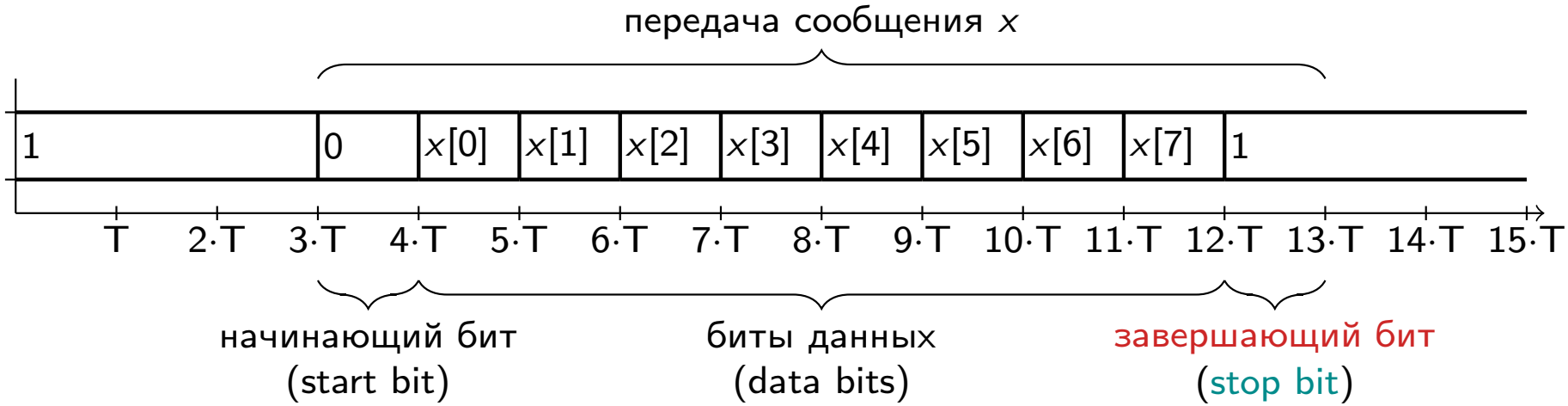
Для обозначения этого намерения заставим схему изменить значение в w на 0 и продержать его *приблизительно* один период T:

UART: общее описание

Предположим, что схема Σ_1 имеет намерение
передать число (сообщение) x ширины 8

Тогда заставим схему Σ_1 после начинающего бита
последовательно выставить в w биты данных (data bits):
$x[0], x[1], \ldots, x[7]$, *приблизительно* по одному периоду T на разряд

После выставления всех разрядов заставим Σ_1
снова выставить в w значение 1 (тишина) хотя бы на один период T



В результате получилось
описание передачи одного сообщения по протоколу UART

UART: погрешности

Почему в описании протокола про период говорилось “приблизительно”?

В реализации протокола UART обязательно содержатся **погрешности**, из-за которых фактический период пересылки каждого бита может (**и даже обязательно будет**) отличаться от T

Погрешность дискретизации

Предположим, что сообщение посылается схемой Σ_1, работающей на частоте μ (с периодом $\tau = \frac{1}{\mu}$)

Тогда время T^* выставления каждого значения в w обязательно кратно периоду этой схемы: $T^* = k \cdot \tau$, где $k \in \{1, 2, 3, \dots\}$

Если частота μ не делится нацело на частоту протокола, то при **любом** выборе k верно $T^* \neq T$

Обычно для передачи битов сообщения выбирается период T^*, *наиболее близкий* к периоду T, хотя и отличающийся от T

UART: погрешности

Почему в описании протокола про период говорилось “приблизительно”?

В реализации протокола UART обязательно содержатся **погрешности**, из-за которых фактический период пересылки каждого бита может (**и даже обязательно будет**) отличаться от T

Физическая погрешность

Никакое реальное устройство не работает “на заданной частоте μ”

Каждое устройство работает
“на частоте, колеблющейся в рамках заданного интервала $[\mu - \varepsilon, \mu + \varepsilon]$”:

- μ — номинальная частота
- ε — **погрешность**

UART: погрешности

Почему в описании протокола про период говорилось “приблизительно”?

В реализации протокола UART обязательно содержатся **погрешности**,
из-за которых фактический период пересылки каждого бита
может (**и даже обязательно будет**) отличаться от T

Погрешность коммерциализации

Для любого *коммерческого* устройства
важна его итоговая себестоимость:
чем она выше, тем выше цена продажи,
и тем меньше покупателей приобретут устройство

Высокие надёжность, скорость и точность элементов устройства $\Rightarrow$
высокая себестоимость

Низкие надёжность, скорость и точность элементов устройства $\Rightarrow$
низкая себестоимость **И** высокие погрешности всех видов

UART: вариации протокола

UART — это **семейство** протоколов, и каждый конкретный протокол задаётся, *в числе прочих*, следующими параметрами:

- Количество битов данных
  - 8 — самое популярное значение
- Порядок пересылки битов данных
  - Самый популярный порядок — от младшего бита к старшему
- Логическое значение, обозначающее тишину
  - “Концептуально”, отсутствие чего бы то ни было — это 0
  - Значение 1 более популярно в UART по историческим причинам:
    в телефонных линиях 0 (низкое напряжение)
    мог означать как “тишину”, так и оборванный провод,
    а 1 (высокое напряжение) — “соединение есть, и в нём тишина”
- Частота протокола
  - Наиболее популярные часто́ты — 9600Гц, 38400Гц, 115200Гц

UART: вариации протокола

UART — это **семейство** протоколов, и каждый конкретный протокол задаётся, *в числе прочих*, следующими параметрами:

- Механизм проверки корректности передачи

Протоколы UART применяются и для передачи данных между **ненадёжными устройствами** через **ненадёжную среду**

Можно модифицировать протокол так, чтобы схема Σ_2 могла (*хоть насколько-нибудь*) удостовериться в корректности сообщения, полученного от Σ_1

UART: вариации протокола

UART — это **семейство** протоколов, и каждый конкретный протокол задаётся, *в числе прочих*, следующими параметрами:

- Механизм проверки корректности передачи

В *ненадёжной* схеме Σ_1 суммарная погрешность периодов передачи битов сообщения может оказаться слишком большой, и Σ_2 может этого не заметить

Например:

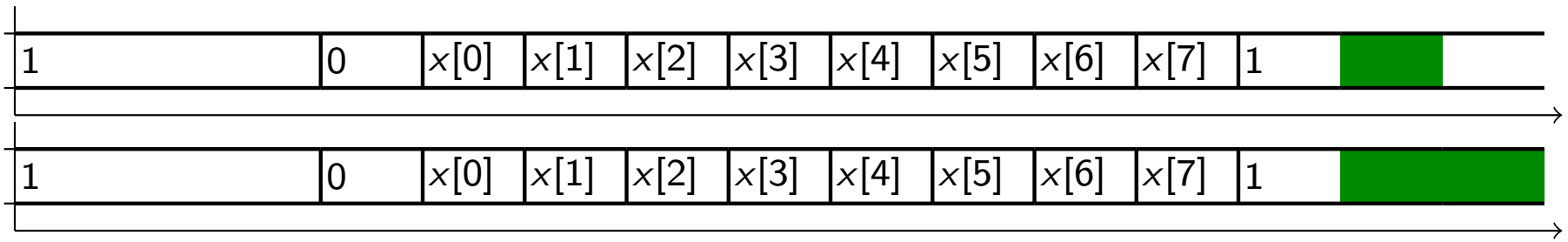
- Σ_1 передаёт одно сообщение (0 11111110 1),
  и передаёт каждый бит **два периода** вместо одного
- Σ_2 принимает два сообщения: (0 01111111 1)111111(0 01111111 1)

UART: вариации протокола

UART — это **семейство** протоколов, и каждый конкретный протокол задаётся, *в числе прочих*, следующими параметрами:

- Механизм проверки корректности передачи

Дополнительные завершающие биты:



Можно заставить схему Σ_1 слать
дополнительные завершающие биты после каждого сообщения

Схема Σ_2 может пометить сообщение как **некорректное**,
если на месте любого из завершающих битов обнаружено значение 0

Количество дополнительных завершающих битов — параметр протокола, определяемый наибольшей возможной погрешностью, накапливаемой при передаче одного сообщения

UART: вариации протокола

UART — это **семейство** протоколов, и каждый конкретный протокол задаётся, *в числе прочих*, следующими параметрами:

- Механизм проверки корректности передачи

При передаче сообщения через *ненадёжную среду*
уровни напряжений могут “зашумляться” помехами настолько,
что схема Σ_2 некорректно распознает некоторые биты сообщения

Предположим, что помехи в среде настолько малы, что

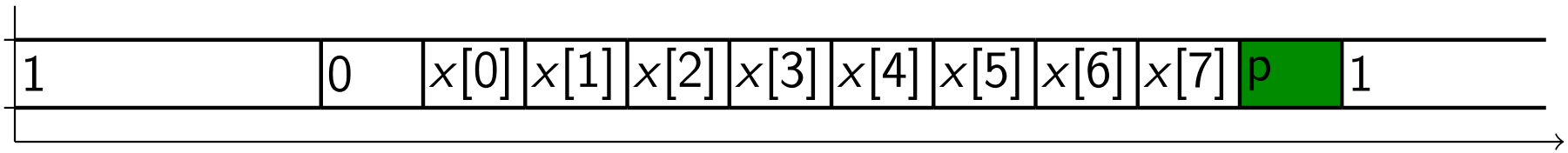
- в каждом сообщении искажается не более чем один бит, и
- начинающий и завершающий биты искажаются настолько редко, что можно считать их неискажаемыми

UART: вариации протокола

UART — это **семейство** протоколов, и каждый конкретный протокол задаётся, *в числе прочих*, следующими параметрами:

- Механизм проверки корректности передачи

Проверяющий бит:



Заставим схему Σ_1 после разрядов пересылаемого числа послать

- бит проверки чётности: $p = x[0] \oplus x[1] \oplus \cdots \oplus x[7]$ — или
- бит проверки нечётности: $p = 1 \oplus x[0] \oplus x[1] \oplus \cdots \oplus x[7]$

Если схема Σ_2 при приёме битов данных накапливает их сумму и накопленный итог не соответствует биту p, то сообщение можно пометить как **некорректное**: искажён либо бит данных, либо проверяющий бит