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 данные и записать их в нужное место”, 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 (как инициализировать это место / открыть сессию / организовать буфер отправки / ...), 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, в которой “скрыты” многие низкоуровневые детали, или
- как минимум, **на готовом процессоре**, внутри которого реализованы нетривиальные детали, “скрытые” от машинного 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

Виды устройств и информации и способы её передачи в этих протоколах существенно различаются, но во всех этих протоколах есть общая неизменная черта: **обязательная часть каждого из перечисленных протоколов — описание того, как устройство A должно управлять логическими уровнями напряжений в реальных проводах в реальном времени, чтобы устройство B на другом конце проводов могло принять значение x, которое хочет послать A**

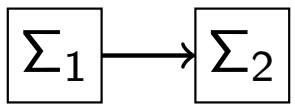
Остановимся подробнее на самых простых протоколах и их реализации, *обозначив при этом характерные черты и многих других протоколов*

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 передачи данных схемы должны быть соединены хотя бы одним проводом



Универсальность: чем меньше проводов требуется для передачи данных, тем выше шанс, что в схеме найдётся столько лишних контактов

Ограничимся одним соединяющим проводом, и попробуем передать число x по нему как можно более простым способом

---

[1] Этот протокол является "простым" 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 обсуждение физики в начале курса: схемам Σ_1, Σ_2 следует договориться о потенциале, обозначающем логический ноль

В противном случае потенциал, выставляемый в проводе 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 принимающую подсхему схемы Σ_2 от своего источника питания[1]

Замечание. Такое "выравнивание" потенциалов характерно для реализации пересылки данных не только по UART, но и по многим другим протоколам

[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

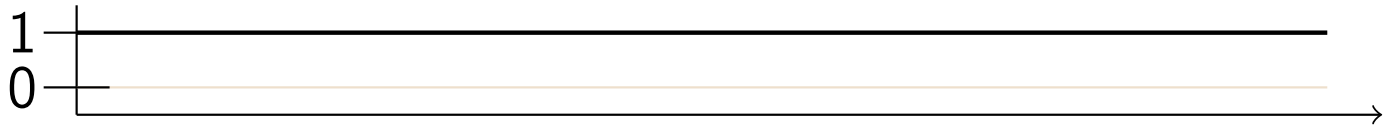
Больше простоты: позволим схемам заранее договориться о **ширине** передаваемого числа

Для определённости считаем, что передаётся число ширины 8

UART: общее описание

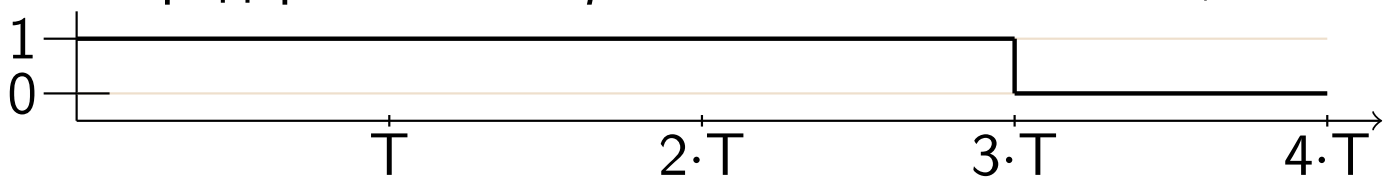
Схема Σ_1 может и не иметь намерения передать какое бы то ни было число

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



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

Для обозначения этого намерения заставим схему изменить значение в проводе на 0 и продержать его *приблизительно* столько, каков период T:

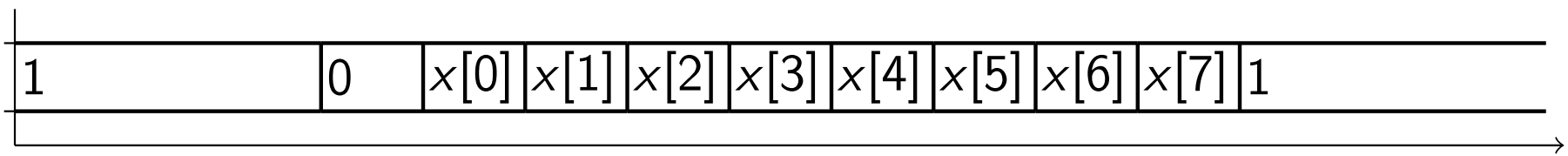


UART: общее описание

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

Тогда заставим схему Σ_1 после обозначения намерения о передаче последовательно выставить в соединяющем проводе разряды $x[0], x[1], \ldots, x[7]$, *приблизительно* по одному периоду T на разряд

После выставления всех разрядов заставим Σ_1 снова выставить в проводе значение 1 (“нет передачи”)



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 конкретного значения может отличаться от T

Физическая погрешность, которая всегда есть в реальном мире

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 конкретного значения может отличаться от T

Погрешность дискретизации

Предположим, что физической погрешности не существует, и что передача по протоколу UART реализована **синхронной** схемой Σ_1, работающей на частоте μ (и с периодом $\tau = \frac{1}{\mu}$)

Тогда время T^* выставления каждого значения в проводе 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 значений выбирается доступный период T^*, *наиболее близкий* к периоду T

Итог: время выставления каждого значения может отличаться от периода протокола, и *обычно* принадлежит интервалу $(T - \tau, T + \tau)$

UART: погрешности

Почему в описании протокола про период говорилось “приблизительно”?

В реализации протокола UART обязательно содержатся **погрешности**, из-за которых фактический период пересылки каждого конкретного значения может 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 во всех проявлениях (в том числе физические и дискретизации)

UART: вариации протокола

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

- Количество пересылаемых разрядов
  - 8 — самое популярное значение
- Порядок пересылки разрядов
  - Самый популярный порядок — от младшего разряда к старшему
- Логическое значение, обозначающее отсутствие пересылки
  - “Концептуально”, отсутствие чего бы то ни было — это 0
  - Значение 1, чаще использующееся в протоколах UART, сложилось 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 следующими параметрами:

- Механизм проверки корректности передачи

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

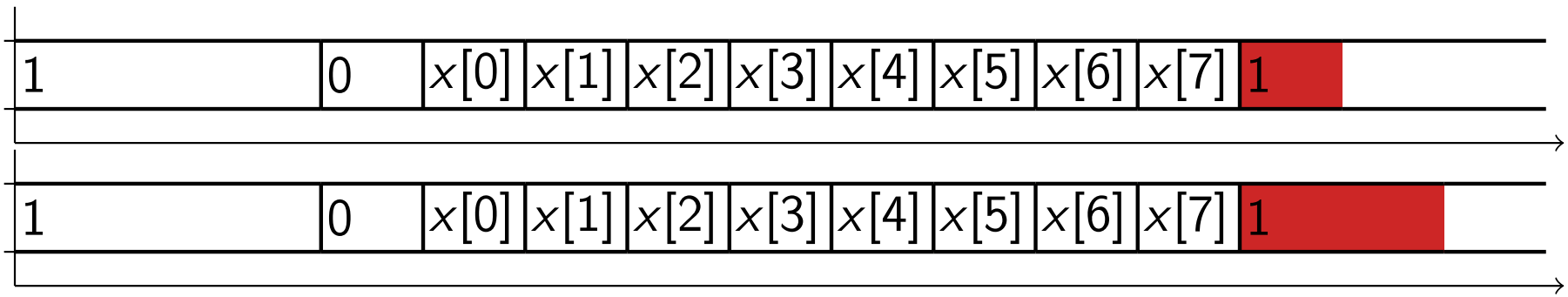
Можно модифицировать протокол так, чтобы схема Σ_2 могла (*хоть насколько-нибудь*) удостовериться в том, что правильно приняла число, переданное схемой Σ_1

UART: вариации протокола

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

- Механизм проверки корректности передачи

Обязательные завершающие разряды:



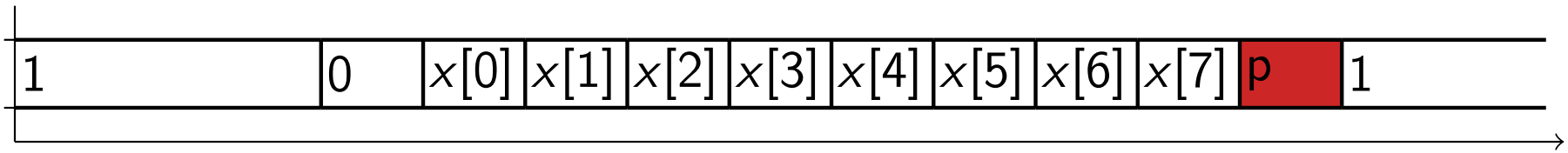
Проверив, что после восьми принятых разрядов сообщения некоторое время обязательно выдаётся значение 1, Σ_2 *повысит уверенность* в том, что Σ_1 не передавала сообщение “слишком медленно”, или обнаружит некорректность пересылки

UART: вариации протокола

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

- Механизм проверки корректности передачи

Проверка чётности:



Заставим схему Σ_1 после разрядов пересылаемого числа послать

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

Тогда Σ_2, приняв дополнительный разряд и сравнив его с суммой предыдущих разрядов, *повысит уверенность* в том, что в среде пересылки не было помех, **исказивших** один из пересылаемых разрядов, или обнаружит некорректность пересылки