

# Языки описания схем

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

## Блок К1

Кое-что ещё:

Протоколы передачи данных

Протокол UART (общее описание)

Лектор:

**Подымов Владислав Васильевич**

E-mail:

**valdus@yandex.ru**

# Вступление

Всё так складно в этом курсе рассказывалось, но чего-то недостаёт (?)

На занятиях возникало много несложных вспомогательных схем:

- ▶ последовательные и параллельные регистры
- ▶ мультиплексоры, демультимплексоры, дешифраторы
- ▶ таймеры, счётчики, делители частоты
- ▶ ...

Вспомогательные схемы использовались при построении более сложных, но всё равно “игрушечных” схем:

- ▶ Счётчики такие, счётчики сякие
- ▶ Арифметические устройства
- ▶ Стеки/очереди (*смотря кому что досталось*)
- ▶ ...

А как устроены серьёзные полезные “боевые” схемы?

# Вступление

Самая известная в мире схема — это **процессор** (он же **центральный процессор**, он же **микропроцессор**)

Менее известны, но более широко применяются маленькие специализированные процессоры с обширно встроенной периферией — **микроконтроллеры**<sup>1</sup>

Принципам проектирования процессоров посвящён курс “... проектирования архитектуры СБИС” следующего семестра

Но одними только процессорами мир схем не ограничивается

В оставшейся части курса обсудим нескольких несложных, но очень полезных схем, помогающих разработчикам и пользователям налаживать взаимодействие с самыми разнообразными устройствами

---

<sup>1</sup> Не будем углубляться в дискуссию о том, где именно кончаются процессоры и начинаются микроконтроллеры

# Вступление

Взаимодействие устройств между собой — это (прежде/чаще всего) пересылка *данных*, и “выглядеть” это может по-разному

**Например**, пересылка данных с одного *компьютера* на другой:

- ▶ 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

Взаимодействие устройств между собой — это (прежде/чаще всего) пересылка *данных*, и “выглядеть” это может по-разному

**Например**, пересылка данных с одного *компьютера* на другой:

- ▶ 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 курсе  
“... проектирования архитектуры СБИС”)
- ▶ 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 тоже 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 всё́м 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<sup>1</sup> универсального протокола:<sup>2</sup>

**UART** (Universal Asynchronous Receiver-Transmitter)

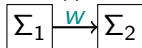
Начнём описание протокола немного издалека:

представим себе цифровые схемы  $\Sigma_1$ ,  $\Sigma_2$ ,

одна из которых ( $\Sigma_1$ ) хочет передать другой ( $\Sigma_2$ ) число  $x$

*Очевидный факт:*

для схемы  $\Sigma_1$ ,  $\Sigma_2$  должны быть соединены хотя бы одним проводом



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

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

---

<sup>1</sup> Имеется в виду “простота” почти во всех смыслах, какие только можно придумать. Оттенки “простоты” будут коротко обозначаться дальше.

<sup>2</sup> 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  $\Sigma_1$

*Больше простоты:*

считаем, что  $\Sigma_1$  и  $\Sigma_2$  — это синхронные схемы со сбросом

*Замечание-напоминание.* “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” разработчика схем

# UART: общее описание

*Вспоминаем физику:* схемам  $\Sigma_1$ ,  $\Sigma_2$  следует договориться о потенциале, обозначающем логический ноль

Иначе потенциал, выставляемый в  $w$  схемой  $\Sigma_1$ , может иметь некорректную логическую трактовку в  $\Sigma_2$  или даже вывести эту схему из строя

Как это сделать: например,

- ▶ подключить схемы к одному источнику питания, или
- ▶ добавить второй провод между схемами и послать в него
  - ▶ значение 0, и физически отключить  $\Sigma_2$  от своего источника питания
  - ▶ напряжение  $(GND - V)$ , где  $(GND + V)$  — текущее напряжение в  $w$ , поставив на концах провода преобразователь получившейся **дифференциальной пары** проводов в сигнал

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

# UART: общее описание

*Простота:* позволим схемам заранее договориться об особенном числе  $\nu$  — частоте протокола

При этом не будем заставлять тактовые сигналы схем осциллировать на частоте  $\nu$

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

*Больше простоты:*

позволим схемам заранее договориться о ширине передаваемого числа

Для определённости считаем, что передаётся число ширины 8

# UART: общее описание

Схема  $\Sigma_1$  может и не иметь намерения передать какое бы то ни было число

В этом случае заставим схему  $\Sigma_1$  выставить в  $w$  неизменное значение 1:



Представим себе, что у схемы  $\Sigma_1$  появилось намерение передать число через  $w$

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



# UART: общее описание

Предположим, что схема  $\Sigma_1$  имеет намерение передать число  $x$  (сообщение; ширины 8)

Тогда заставим схему  $\Sigma_1$  после начинающего бита последовательно выставить в  $w$  биты данных (data bits):

$x[0], x[1], \dots, x[7]$ , приблизительно по одному периоду  $T$  на разряд

После выставления всех разрядов заставим  $\Sigma_1$

снова выставить в  $w$  значение 1 (тишина) хотя бы на один период  $T$

передача сообщения  $x$



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  $\Sigma_1$ , работающей на частоте  $\mu$  (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  $\mu$  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  $\mu$ ”

Каждое устройство работает

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

- ▶  $\mu$  — номинальная частота
- ▶  $\varepsilon$  — 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 частоты — 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  $\Sigma_2$  могла (хоть насколько-нибудь) удостовериться в корректности сообщения, полученного от  $\Sigma_1$

## UART: вариации протокола

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

- ▶ Механизм проверки корректности передачи

В *ненадёжной* схеме  $\Sigma_1$  суммарная погрешность периодов передачи битов сообщения может оказаться слишком большой, и  $\Sigma_2$  может этого не заметить

Например:

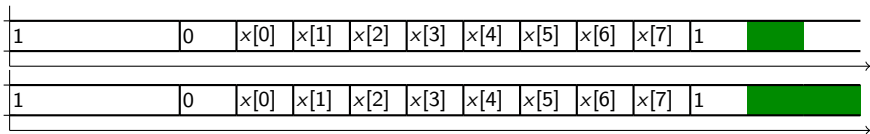
- ▶  $\Sigma_1$  передаёт одно сообщение (0 11111110 1),  
и передаёт каждый бит **два периода** вместо одного
- ▶  $\Sigma_2$  принимает два сообщения: (0 01111111 1)111111(0 01111111 1)

# 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  $\Sigma_1$  слать

**дополнительные завершающие биты** после каждого сообщения

Схема  $\Sigma_2$  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  $\Sigma_2$  некорректно распознает некоторые биты сообщения

Предположим, что помехи в среде настолько малы, что

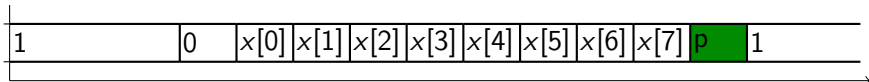
- ▶ в каждом сообщении искажается не более чем один бит, и
- ▶ начинающий и завершающий биты искажаются настолько редко, что можно считать их неискажаемыми

## 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  $\Sigma_1$  после разрядов пересылаемого числа послать

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

Если схема  $\Sigma_2$  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