Языки описания схем

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

Блок K1

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

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

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

На занятиях возникало много несложных вспомогательных схем:

- последовательные и параллельные регистры
- мультиплексоры, демультиплексоры, 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*)
- Совсем миниатюрные управляющие устройства
- ...

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**[1]

Основам 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 разнообразными устройствами

1 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 в аппаратуре
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 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 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*

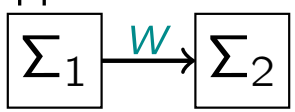
UART: общее описание

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

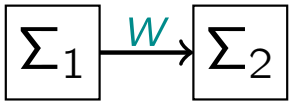
Очевидный факт: чтобы передача данных была возможна,
схемы Σ_1, Σ_2 должны быть соединены хотя бы одним проводом



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



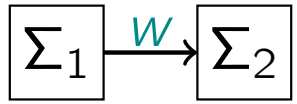
Универсальность: чем меньше проводов требуется для передачи данных, тем выше шанс, что в схеме найдётся столько портов

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

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

Чтобы передача данных была возможна, следует договориться об уровнях напряжений в w, обозначающих 0 и 1 — иначе напряжение, которое Σ_1 считает логическим значением, может неверно трактоваться в Σ_2 или даже вывести эту схему из строя

UART: общее описание



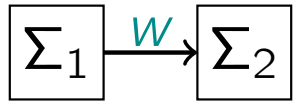
Если Σ_1 и Σ_2 подключены к одному источнику питания (одной паре напряжений GND, Vcc), то договорённость о напряжениях в w соблюдена автоматически

А иначе *типовая* договорённость состоит в том, что «виртуальный» провод w соответствует дифференциальной паре «реальных» проводов:

- Σ_1 выставляет в паре только напряжения вида $GND_1 + V$ и $GND_1 - V$ относительно своего ноля GND_1
- Σ_2 при чтении добавляет к своему нолю GND_2 полуразность значений в паре: $\frac{(GND_1+V)-(GND_1-V)}{2} = V$
- Остаётся только договориться об интервалах напряжений 0 и 1, для каждой схемы — относительно своего потенциала ноля

Принципы работы с дифференциальной парой выходят за рамки курса: будем считать такую пару одним «виртуальным» проводом, передающим 0/1 независимо от точного набора договорённостей

UART: общее описание



Больше простоты:

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

«Простота» здесь состоит в том, что принципы разработки истинно-асинхронных схем (есть, но) «необычны» и непросты, так что не входят в список знаний «обычных» разработчиков схем

Ещё больше простоты:

позволим схемам заранее договориться об особенном числе ν

Будем называть это число частотой протокола, но это не частота схем: не будем заставлять тактовые сигналы схем осциллировать на частоте ν

Периодом протокола будем называть число $T = \frac{1}{\nu}$

ЕЩЁ больше простоты:

позволим схемам заранее договориться о ширине передаваемого числа

Для определённости считаем, что передаётся число ширины 8

UART: общее описание

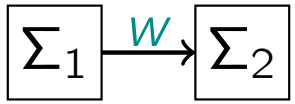
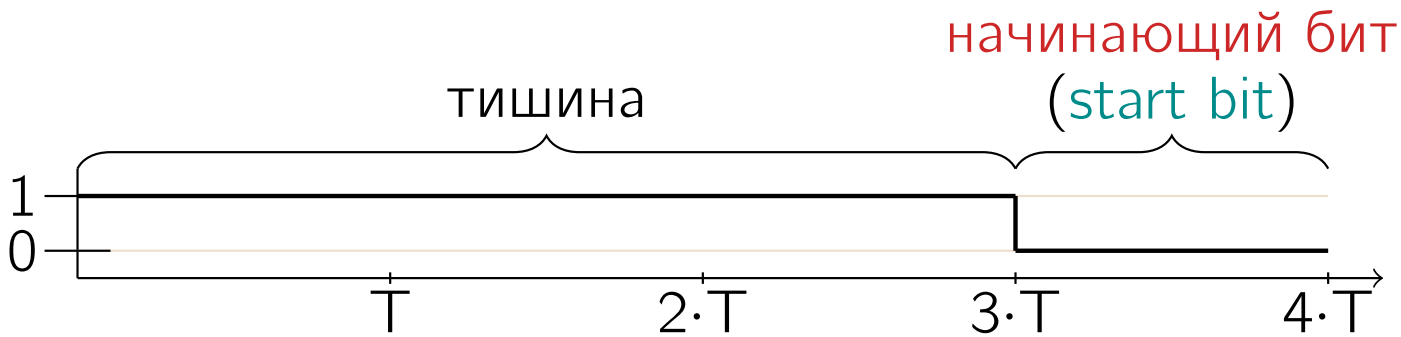


Схема Σ_1 может и не иметь намерения что-либо передавать

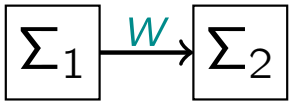
В этом случае заставим Σ_1 выставить в w неизменное значение 1:

Пусть теперь у схемы Σ_1 появилось намерение
передать число через w

Для обозначения этого намерения заставим Σ_1 изменить значение в w на 0 и продержать его *приблизительно* один период протокола:



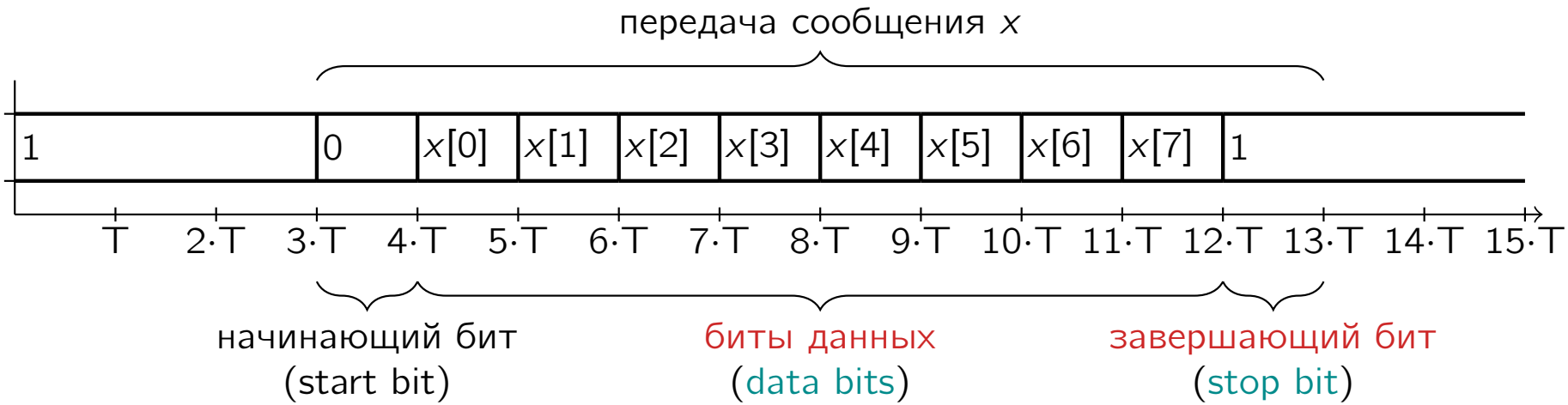
UART: общее описание



Пусть, для ясности, Σ_1 намеревается передать число x

Тогда заставим Σ_1 после начинающего бита
последовательно выставить в w значения $x[0], x[1], \ldots, x[7]$,
приблизительно по одному периоду T на каждое $x[i]$

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 x через w 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

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 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

Погрешность коммерциализации

Для любого *коммерческого* устройства
важна его итоговая стоимость:
чем она выше, тем меньше покупателей приобретут устройство

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

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

UART: вариации протокола

UART — это, строго говоря, **семейство** протоколов, и конкретный протокол задаётся, *в числе прочих*, такими параметрами:

- Количество битов данных
  - 8 — самое популярное значение
- Порядок пересылки битов данных
  - Самый популярный порядок — от младшего бита к старшему
- Логическое значение, обозначающее тишину (противоположное активному уровню)
  - «Концептуально», отсутствие чего бы то ни было — это 0
  - Значение 1 более популярно в UART по историческим причинам: такое значение тишины использовалось в телефонных линиях, чтобы различать тишину и разрыв провода
- Частота протокола
  - Наиболее популярные часто́ты — 9600Гц, 38400Гц, 115200Гц

UART: вариации протокола

UART — это, строго говоря, **семейство** протоколов, и конкретный протокол задаётся, *в числе прочих*, такими параметрами:

- Механизм проверки корректности передачи

UART применяется и для передачи данных между ненадёжными устройствами с погрешностями через ненадёжную среду

Если это не учитывать, то сообщение может быть принято некорректно (принято не то, что отправлялось) по самым разным причинам:

- Замещение (искажение; замена значения на противоположное) бита данных *очевидно* приводит к некорректному приёму
- Искажение начинающего бита: отправляется 0 (1 00000000 11), а принимается (10 00000001 1) — число 1
- Отправляется 254: (0 11111110 1111), но из-за погрешностей — по два периода на бит,
  а принимается (0 01111111 1) 111111 (0 01111111 1) — дважды 128
- ...

UART: вариации протокола

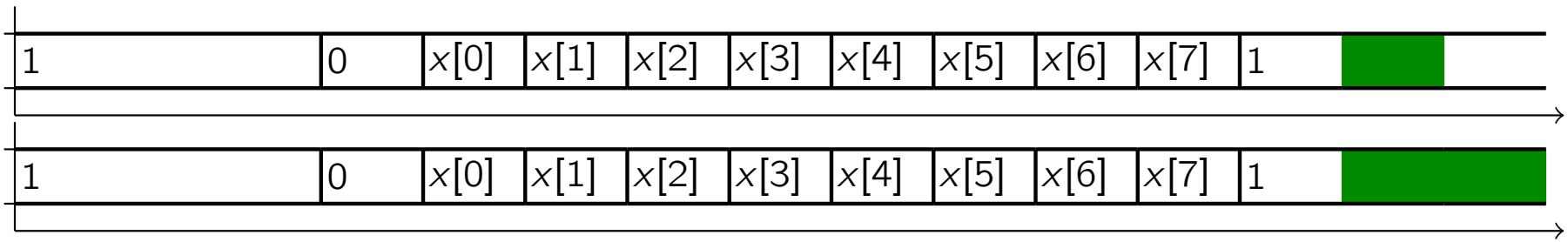
UART — это, строго говоря, **семейство** протоколов, и конкретный протокол задаётся, *в числе прочих*, такими параметрами:

- Механизм проверки корректности передачи

В UART содержатся только простые средства обнаружения ошибок

- в передаче битов данных и
- связанных с накоплением погрешностей в периодах передачи

Добавочные завершающие биты:



Можно заставить схему Σ_1 передавать добавочные завершающие биты в конце сообщения, чтобы Σ_2, обнаружив 0 в завершающих битах, признала сообщение некорректно принятым

Количество дополнительных завершающих битов — это ещё один параметр протокола

UART: вариации протокола

UART — это, строго говоря, **семейство** протоколов, и конкретный протокол задаётся, *в числе прочих*, такими параметрами:

- Механизм проверки корректности передачи

Чем сильнее помехи («уровень шума») в проводе, тем выше вероятность, что каждый конкретный бит будет искажён (принят как противоположный)

Приемлемый уровень шума может выражаться, в частности, в предположении о том, сколько битов может исказиться при передаче одного сообщения

В UART содержатся средства обнаружения искажения битов в предположении о том, что

- в сообщении искажается не более чем один бит и
- начинающий бит не искажается

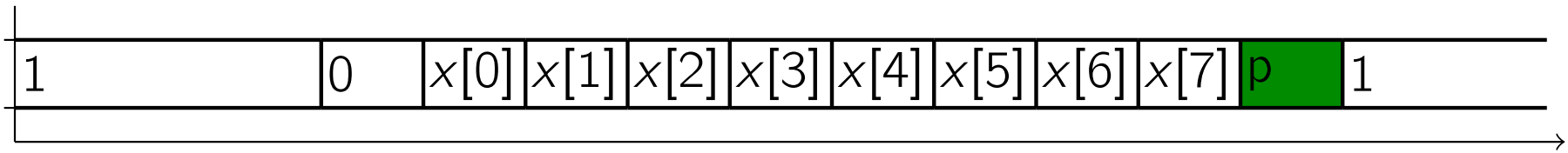
Искажение завершающего бита обнаружить легко (на заданном такте принят 0), а для искажения битов данных ...

UART: вариации протокола

UART — это, строго говоря, **семейство** протоколов, и конкретный протокол задаётся, *в числе прочих*, такими параметрами:

- Механизм проверки корректности передачи

Проверяющий бит:



Можно заставить 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 признаётся некорректно принятым, вероятно из-за искажения битов

Наличие и способ проверки чётности — это ещё один параметр протокола