Глава 8. Протоколы, еще круче. IIC.

Ежелев Г. И.

13 февраля 2026 г.

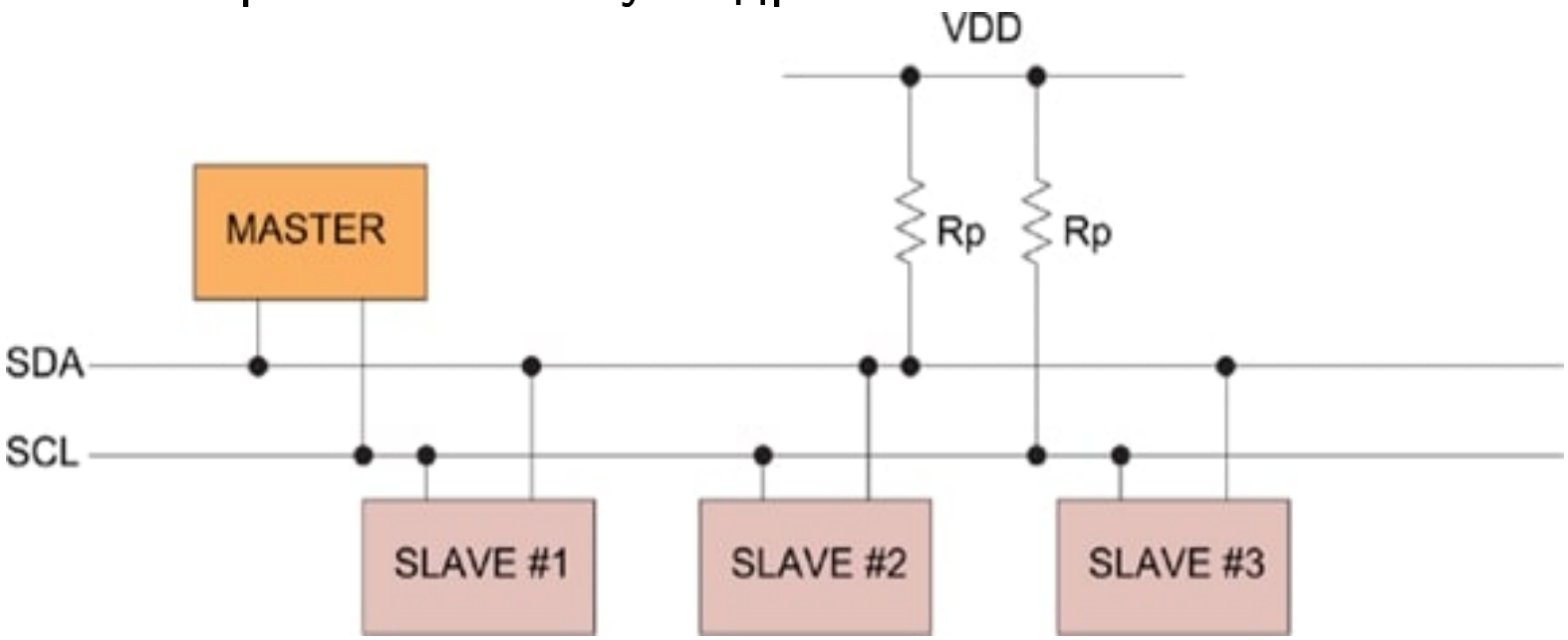
Протокол IIC

IIC - **Inter-Integrated Circuit** - межинтегральная схема, читается ай-ту-си.

- Этот протокол позволяет подключить по одной шине сразу несколько устройств.
- Всегда одно главное устройство Master, которое управляет процессом передачи данных по шине, а также одно или несколько ведомых Slave устройств, которые подчиняются мастеру в процессе передачи данных.
- В отличие от UART имеется сигнал синхронизации, по которому мастер определяет момент передачи данных (в обе стороны).

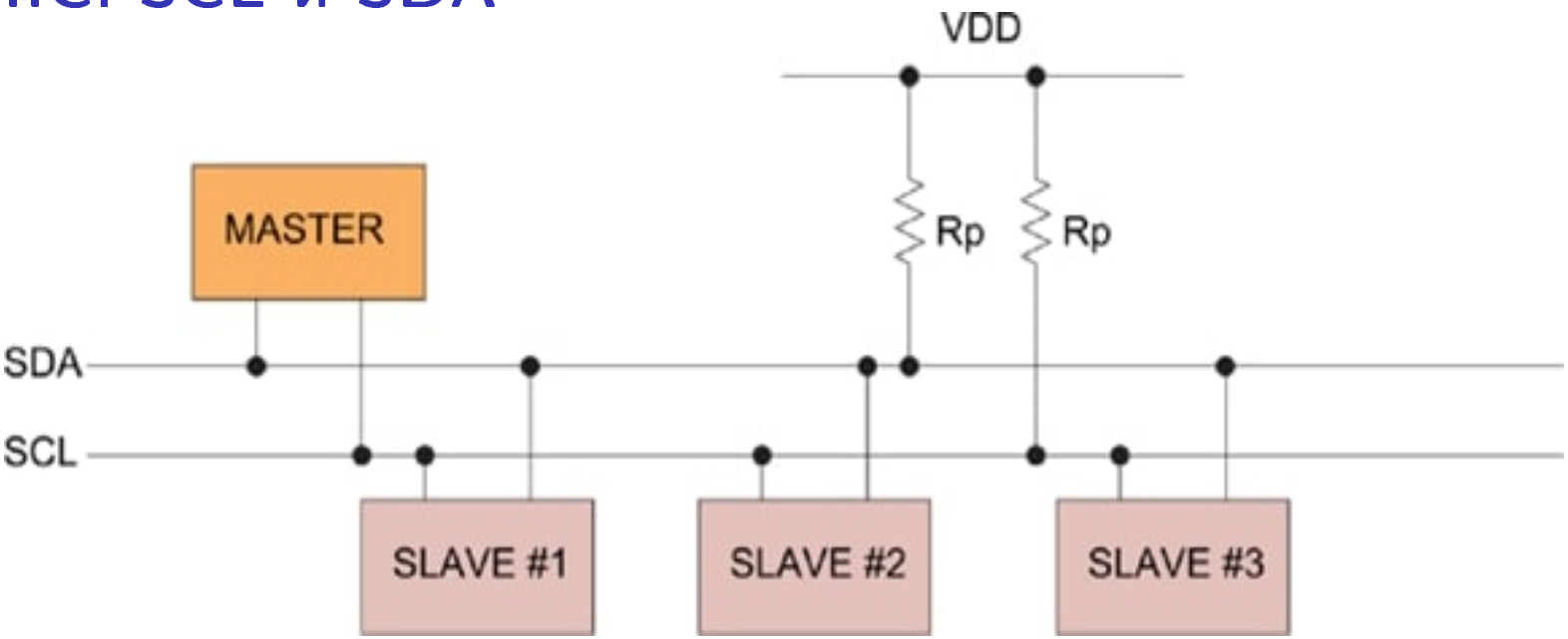
IIC. SCL и SDA

Рассмотрим схематику подробнее.



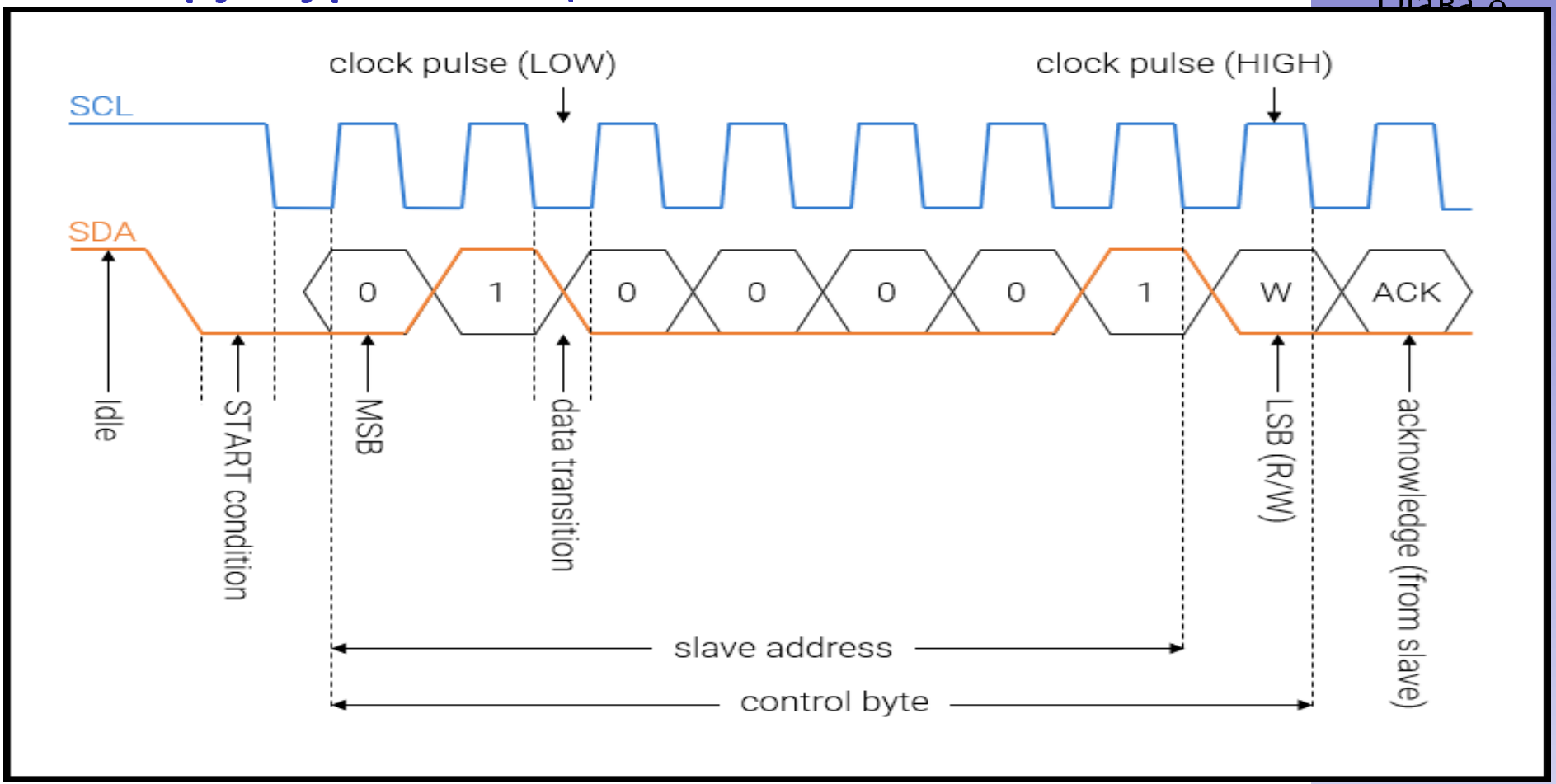
- Шина SCL - Serial Clock - сигнал синхронизации. Чтение данных при восходящем фронте, изменение шины данных, только когда SCL в низком состоянии.
- Шина SDA - Serial Data - сигнал данных. Одна шина используется в обе стороны.

IIC. SCL и SDA



Обе шины подтянуты наверх резисторами (обычно 4,7кОм). Это позволяет избавиться от шумов на линии. Так же это решает проблему неопределенного уровня на шине после отправки 0 (после отправки 0 устройство ничего не делает с шиной => значение непредсказуемо (шина может остаться в нуле и другое устройство подумает, что отправлен не 0, а 00)). С подтяжкой каждому устройству достаточно только опускать сигнал, подниматься он будет сам.

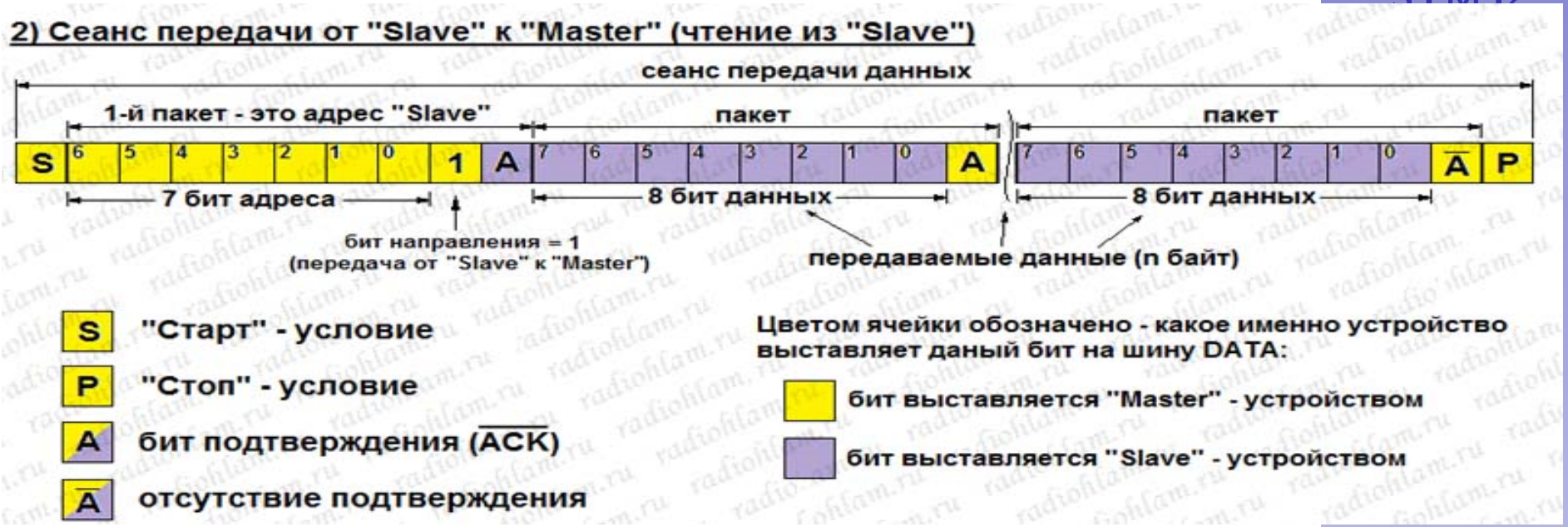
IIC. Структура сообщения



- Итак, сначала рассмотрим шину SCL. Синхронизация по возрастающему фронту, данные на SDA выставлены до фронта SCL и меняются только после спада SCL.

IIC. Структура сообщения

- Т.к. на одной шине несколько устройств, каждому программист заранее присваивает адрес. Он может быть неизменяемым или разработчики могут оставить возможность его смены.
- Поэтому в любом сообщении первый байт - адрес. Он состоит из 7 бит и в конце один бит, который определяет тип сообщения, которое будет передано - Read/Write.



- Следующий байт(ы) - данные: либо адрес регистра устройства, который хочет **прочитать** master либо наоборот **записать** данные (бит R/W!).
- Каждое сообщение начинается со стартовой последовательность (квадрат S). Как она устроена - ниже.
- В конце каждого байта - подтверждение корректной его обработки второй стороной - Ack/Nack.

IIC. Пакеты в обе стороны

1) Сеанс передачи от "Master" к "Slave"

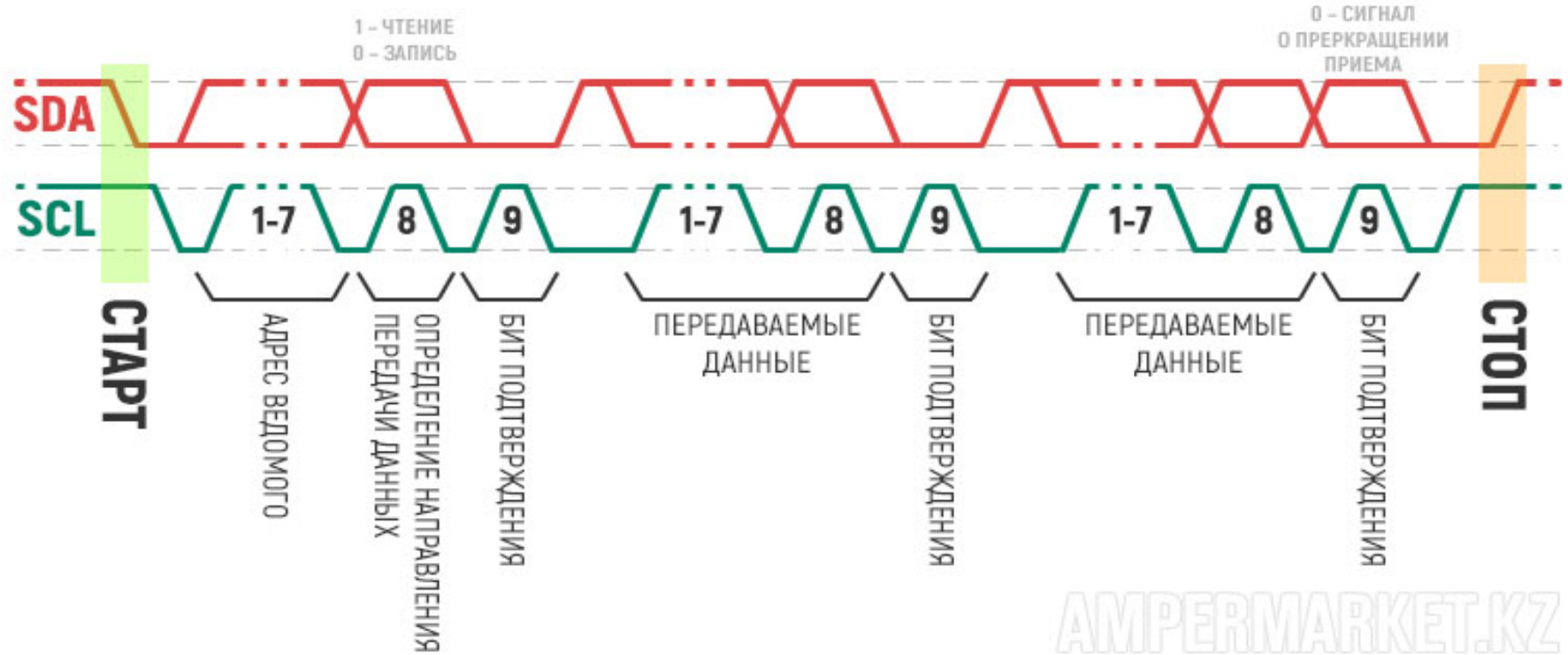
2) Сеанс передачи от "Slave" к "Master" (чтение из "Slave")

- S — "Старт" - условие
- P — "Стоп" - условие
- A — бит подтверждения ($\overline{ACK}$)
- Ā — отсутствие подтверждения

Цветом ячейки обозначено - какое именно устройство выставляет даный бит на шину DATA:

- бит выставляется "Master" - устройством
- бит выставляется "Slave" - устройством

IIC. Start,Stop. Ack, Nack



- Start - SDA вниз; SCL вверху.
- Stop - SDA и SCL вверх - возврат в состояние покоя
- ACK - Acknowledge - логический 0. Данные приняты.
- NACK - Логическая 1. Данные не приняты. Еще Nack отправляет мастер в конце чтения.

Код. Стандартные методы SPL

Можно написать IIC так же, как мы это делали с UART.

Можно так же, через циклы и ожидание или на буфере. Подробно останавливаться вероятно не будем. Но вот код от нейросетки:

```
I2C_InitTypeDef I2C_InitStructure;
I2C_InitStructure.I2C_ClockSpeed = 100000;
I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
I2C_InitStructure.I2C_OwnAddress1 = 0x33; // Любой адрес
I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
I2C_Init(I2C1, &I2C_InitStructure);
I2C_Cmd(I2C1, ENABLE);
```

GPIO

```
// Включение тактирования
RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

// Настройка GPIO
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOB, &GPIO_InitStructure);

// Подключение к AF
GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_I2C1);
GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_I2C1);
```

Обратите внимание на красный текст!!!

Код. Стандартные методы SPL


```
I2C_InitTypeDef I2C_InitStructure;
I2C_InitStructure.I2C_ClockSpeed = 100000;
I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
I2C_InitStructure.I2C_OwnAddress1 = 0x33; // Любой адрес
I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
I2C_Init(I2C1, &I2C_InitStructure);
I2C_Cmd(I2C1, ENABLE);
```

- Mode - I2C. Этот же контроллер управляет еще и SMBus - протокол основанный на IIC.
- DutyCycle - Отношение времени 1 и 0 в SCL. При высоких скоростях обязательно выставлять приведенный вариант.
- OwnAddress - Адрес, как подчиненного.
- Ack - Отправлять ли Ack мастеру при чтении или нет.
- Address - Размер адреса - 7 или 10 бит.

Код. SPL. Простая отправка

Отправка через циклы полностью аналогична UART

```
uint8_t data[] = {0x02, 0x01}; // Регистр и значение
while (I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

I2C_GenerateSTART(I2C1, ENABLE);
while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

I2C_Send7bitAddress(I2C1, 0x94, I2C_Direction_Transmitter); // Адрес устройства
while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

I2C_SendData(I2C1, data[0]);
while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

I2C_SendData(I2C1, data[1]);
while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

I2C_GenerateSTOP(I2C1, ENABLE);
```

Код. SPL. IIC на прерываниях

Для буферной отправки настроим
NVIC аналогично тому, как мы делали это для UART

```
// Включение прерываний (после I2C_Init)
I2C_ITConfig(I2C1, I2C_IT_EVT | I2C_IT_BUF | I2C_IT_ERR, ENABLE);


// NVIC
NVIC_InitTypeDef NVIC_InitStructure;
NVIC_InitStructure.NVIC_IRQChannel = I2C1_EV_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Код. SPL. Простая отправка

Отправка через буфер такая же. Только названия другие.


```
void I2C1_EV_IRQHandler(void) {
    uint32_t event = I2C_GetLastEvent(I2C1);

    if (event == I2C_EVENT_MASTER_MODE_SELECT) {
        I2C_Send7bitAddress(I2C1, slave_addr << 1, I2C_Direction_Transmitter);
    }
    else if (event == I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED) {
        if (tx_buffer_pos < tx_len) {
            I2C_SendData(I2C1, tx_buffer[tx_buffer_pos++]);
        } else {
            I2C_GenerateSTOP(I2C1, ENABLE);
        }
    }
    else if (event == I2C_EVENT_MASTER_BYTE_TRANSMITTED) {
        if (tx_buffer_pos < tx_len) {
            I2C_SendData(I2C1, tx_buffer[tx_buffer_pos++]);
        } else {
            I2C_GenerateSTOP(I2C1, ENABLE);
        }
    }

    // Обработка ошибок
    if (event == I2C_EVENT_MASTER_MODE_SELECT_FAILED) {
        I2C_GenerateSTOP(I2C1, ENABLE);
    }
}
```

Резюме

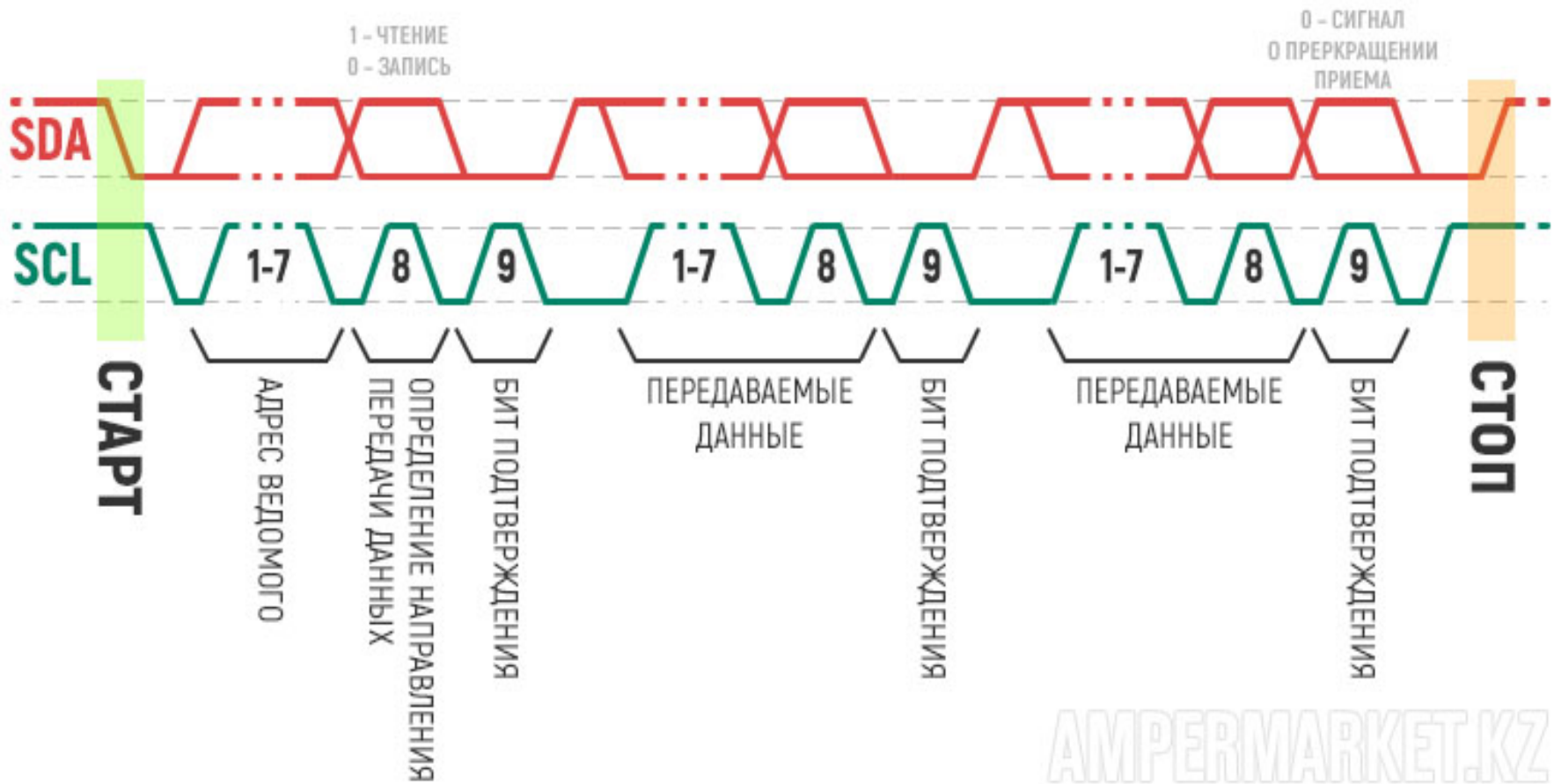
Почему на пред. слайде в команде Send7bitAddress адрес смещен на 1 бит влево? (вспомните структуру байта адреса устройства).
В целом, если вам понадобится мощный параллельный IIC можете это написать.
У IIC много проблем, поэтому так же сильно зарываться в него, как в UART мы не будем:

- Если зависло одно устройство - зависла вся шина
- Скорость сравнительно низкая - всё, что можно повесить не на IIC лучше на него не вешать, особенно дисплеи, там задержка будет колоссальная!!.
- Но, т.к. IIC довольно прост мы напишем свою реализацию этого протокола.
- Часто IIC пишут как I^2C т.к. $I * I = I^2$ Читают ай-ту-си.

Свой I2C

Итак, в реализации своего I2C нет какого-то великого смысла, но так вы точно станете сильнее. К тому же сложным это может показаться только в начале.

Давайте обновим в памяти структуру пакета:



Свой I2C

Сначала напишем (SCL/SDA)(HIGH/LOW)
и задержку между битами, т.к. 1 мс - слишком много.

Конструктор

```
#include "soft_i2c.h"

soft_i2c::soft_i2c(pin & sclPin,
                   pin & sdaPin):
                   _sclPin(sclPin),
                   _sdaPin(sdaPin)
{
    sdaHigh();
    sclHigh();
}

void soft_i2c::sdaHigh()
{
    _sdaPin.setBit();
}

void soft_i2c::sdaLow()
{
    _sdaPin.resetBit();
}

void soft_i2c::sclHigh()
{
    _sclPin.setBit();
}

void soft_i2c::sclLow()
{
    _sclPin.resetBit();
}

void soft_i2c::delay()
{
    volatile uint8_t iter = 25;
    while(iter)
    {
        iter--;
    }
}
```

Свой I2C. Чтение шин

```
bool soft_i2c::sdaRead()
{
    bool data = (_sdaPin.getGPIOx()->IDR & _sdaPin.getPinNumber());
    return data;
}

bool soft_i2c::sclRead()
{
    bool data = (_sclPin.getGPIOx()->IDR & _sclPin.getPinNumber());
    return data;
}

void soft_i2c::softI2cInit()
{
    _sdaPin.pinInit();
    _sclPin.pinInit();
}
```

Свой I2C. Start, ACK


```
bool soft_i2c::generateStart()
{
    sdaHigh();
    sclHigh();
    delay();
    if(!sdaRead())
    {
        return false;
    }
    sdaLow();
    delay();
    if(sdaRead())
    {
        return false;
    }
    sclLow();
    delay();
    if(sclRead())
    {
        return false;
    }
    return true;
}
```

```
bool soft_i2c::generateStop()
{
    sclLow();
    delay();
    sdaLow();
    delay();
    sclHigh();
    delay();
    sdaHigh();
    delay();
    return true;
}
void soft_i2c::sendAch()
{
    sclLow();
    delay();
    sdaLow();
    delay();
    sclHigh();
    delay();
    sclLow();
    delay();
}
```

```
void soft_i2c::sendNach()
{
    sclLow();
    delay();
    sdaHigh();
    delay();
    sclHigh();
    delay();
    sclLow();
    delay();
}
```

Свой I2C. Send


```
void soft_i2c::writeBit(bool bit)
{
    sclLow();
    delay();
    if(!bit)
    {
        sdaLow();
    }
    else
    {
        sdaHigh();
    }
    delay();
    sclHigh();
    delay();
}
```

```
bool soft_i2c::send(uint8_t package)
{
    for(uint8_t iter = 0x80; iter > 0x00; iter >>=1)
    {
        if(package & iter)
        {
            writeBit(1);
        }
        else
        {
            writeBit(0);
        }
    }
    sclLow();
    delay();
    return true;
}
```

Свой I2C. Read


```
uint8_t soft_i2c::read()
{
    sdaHigh();
    uint8_t number = 0;
    for(int8_t iter = 7; iter >= 0; iter--)
    {
        number <<= 1;
        sclLow();
        delay();
        sclHigh();
        delay();
        if(sdaRead())
        {
            number |= 0x01;
        }
    }
    sclLow();
    delay();
    return number;
}
```

```
bool soft_i2c::readAch()
{
    sclLow();
    delay();
    sclHigh();
    delay();
    if(sdaRead())
    {
        sclLow();
        delay();
        return false;
    }
    sclLow();
    delay();
    return true;
}
```

Практика

IRLocator360 - датчик для обнаружения инфракрасного мяча Robocup Junior Soccer.
Внимательно прочитайте как работают разные режимы и напишите код, который будет выбирать подходящий в конкретный момент времени.
Документация - в репозитории, папка docs.

Материалы курса можно найти по ссылке:
https://github.com/haaroner/ezh239