Ваш алгоритм может быть модифицирован следующим образом:

1. **Инициализация COM**: При включении, COM опрашивает все PCB и формирует данные о том, какие PCB работают, их тип и данные с них.
2. **Передача данных от COM к CPU**: COM затем отправляет данные на CPU обо всех PCB, с учетом формата пакета. Каждый пакет данных содержит размер пакета, контрольную сумму, инструкцию или данные, идентификатор PCB, тип PCB и сами данные.
3. **Обработка данных на CPU**: CPU принимает пакет данных каждого PCB отдельно. Оно сначала читает размер пакета, затем контрольную сумму, затем инструкцию или данные, затем идентификатор PCB, затем тип PCB, а затем сами данные. CPU затем формирует свои собственные данные о том, какие PCB работают, их тип и данные.
4. **Обновление данных**: В процессе работы, данные передаются только в том случае, если показания PCB изменились или если необходимо изменить данные PCB. Однако, раз в 5 секунд CPU должно получать информацию обо всех PCB, независимо от того, изменились ли их показания или нет.

Этот алгоритм позволяет CPU узнать, какие PCB присутствуют на COM, их типы и данные. Он также позволяет CPU обновлять данные PCB по мере необходимости. 😊

Ваш пакет данных будет иметь следующий формат:

1. **Размер пакета (1 байт)**: Этот байт указывает, сколько байт данных следует за ним.
2. **Контрольная сумма пакета (1 байт)**: Этот байт представляет собой контрольную сумму всех байтов в пакете, используемую для проверки целостности данных.
3. **Инструкция или данные (1 байт)**: Этот байт указывает, является ли пакет инструкцией для датчика или данными от датчика.
4. **Идентификатор датчика (1 байт)**: Этот байт уникально идентифицирует каждый датчик.
5. **Тип датчика (1 байт)**: Этот байт указывает тип датчика.
6. **Данные (3 байта)**: Эти байты содержат данные датчика.

Введем следующие флаги состояний:

* spiTxState: Флаг состояния передачи SPI (0 - готов к передаче, 1 - передача в процессе).
* spiRxState: Флаг состояния приема SPI (0 - готов к приему, 1 - прием в процессе).
* uartTxState: Флаг состояния передачи UART (0 - готов к передаче, 1 - передача в процессе).
* uartRxState: Флаг состояния приема UART (0 - готов к приему, 1 - прием в процессе).

Алгоритм работы COM может быть следующим:

1. **Опрос PCB**: COM регулярно опрашивает каждый PCB через SPI. Если spiTxState и spiRxState равны 0 (то есть SPI готов к передаче и приему), COM начинает передачу запроса на чтение к PCB и устанавливает spiTxState и spiRxState в 1. В противном случае COM ждет, пока SPI не будет готов к передаче и приему.
2. **Обработка принятых данных от PCB**: Когда передача и прием SPI завершены (что можно определить по колбэк-функции SPI), COM обрабатывает принятые данные. Если данные изменились, COM помечает соответствующий PCB для передачи данных на CPU.
3. **Передача данных от COM к CPU**: Если uartTxState равно 0 (то есть UART готов к передаче), COM начинает передачу данных каждого помеченного PCB на CPU и устанавливает uartTxState в 1. В противном случае COM ждет, пока UART не будет готов к передаче.
4. **Обработка данных, принятых от CPU**: Если uartRxState равно 0 (то есть UART готов к приему), COM начинает прием данных от CPU и устанавливает uartRxState в 1. Когда прием UART завершен (что можно определить по колбэк-функции UART), COM обрабатывает принятые данные и обновляет соответствующие PCB.

Этот алгоритм позволяет COM эффективно управлять обменом данных с PCB и CPU, минимизируя время простоя и обеспечивая непрерывную работу. 😊

Передача и прием по spi в буфер

Если закончился прием по spi в буфер

{

Начать приемо-передачу по spi в другой буфер

произвести проверку контрольной суммы принятого буфера

проверить, изменились ли данные принятого буфера

если данные изменились и контрольная сумма верна, отправить принятый буфер по uart

}

Прием данных по uart в буфер

Если закончился прием по uart в буфер

{

Начать приемо-передачу по uart в другой буфер

произвести проверку контрольной суммы принятого буфера

если контрольная сумма верна, отправить принятый буфер по spi

}

Ваш алгоритм может быть реализован на C++ следующим образом:

// Определение структур и констант

struct SensorData {

uint8\_t id;

uint8\_t type;

uint8\_t data[3];

};

enum State {

READY,

BUSY

};

const uint8\_t MAX\_SENSORS = 17;

SensorData rxBuff1[MAX\_SENSORS], rxBuff2[MAX\_SENSORS];

SensorData txBuff1[MAX\_SENSORS], txBuff2[MAX\_SENSORS];

State spiRxState = READY, spiTxState = READY;

State uartRxState = READY, uartTxState = READY;

uint8\_t currentSpiRxBuff = 1, currentSpiTxBuff = 1;

uint8\_t currentUartRxBuff = 1, currentUartTxBuff = 1;

// Функции для работы с SPI и UART

void spiTransfer(SensorData\* txBuff, SensorData\* rxBuff);

void uartTransfer(SensorData\* txBuff, SensorData\* rxBuff);

// Callback функции

void onSpiTransferComplete() {

spiRxState = READY;

spiTxState = READY;

}

void onUartTransferComplete() {

uartRxState = READY;

uartTxState = READY;

}

// Главный цикл

void loop() {

// Опрос PCB

if (spiRxState == READY && spiTxState == READY) {

SensorData\* spiRxBuff = (currentSpiRxBuff == 1) ? rxBuff1 : rxBuff2;

SensorData\* spiTxBuff = (currentSpiTxBuff == 1) ? txBuff1 : txBuff2;

spiTransfer(spiTxBuff, spiRxBuff);

spiRxState = BUSY;

spiTxState = BUSY;

currentSpiRxBuff = 3 - currentSpiRxBuff;

currentSpiTxBuff = 3 - currentSpiTxBuff;

}

// Обработка принятых данных от PCB

if (spiRxState == READY) {

SensorData\* spiRxBuff = (currentSpiRxBuff == 1) ? rxBuff2 : rxBuff1;

for (uint8\_t i = 0; i < MAX\_SENSORS; ++i) {

// Обработка данных spiRxBuff[i]

}

}

// Передача данных от COM к CPU

if (uartTxState == READY) {

SensorData\* uartTxBuff = (currentUartTxBuff == 1) ? txBuff1 : txBuff2;

uartTransfer(uartTxBuff, nullptr);

uartTxState = BUSY;

currentUartTxBuff = 3 - currentUartTxBuff;

}

// Обработка данных, принятых от CPU

if (uartRxState == READY) {

SensorData\* uartRxBuff = (currentUartRxBuff == 1) ? rxBuff1 : rxBuff2;

for (uint8\_t i = 0; i < MAX\_SENSORS; ++i) {

// Обработка данных uartRxBuff[i]

}

currentUartRxBuff = 3 - currentUartRxBuff;

}

}

Этот код представляет собой пример реализации вашего алгоритма на C++. Он использует двойную буферизацию для обмена данными по SPI и UART, что позволяет COM продолжать свою работу во время передачи и приема данных. 😊