|  |  |
| --- | --- |
| Gerb-BMSTU_01 | Министерство науки и высшего образования Российской Федерации  Федеральное государственное бюджетное образовательное учреждение  высшего образования  «Московский государственный технический университет  имени Н.Э. Баумана  (национальный исследовательский университет)»  (МГТУ им. Н.Э. Баумана) |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | Специальное машиностроение |

|  |  |
| --- | --- |
| КАФЕДРА | Автономные информационные и управляющие системы |

|  |  |
| --- | --- |
| дисциплина | Цифровые устройства и микропроцессоры |

|  |
| --- |
| домашнее задание«Студент1» |

|  |
| --- |
| Устройство измерения температуры«Студент1» |
| *название домашнего задания* |

|  |  |
| --- | --- |
| Группа | СМ5-81 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент |  |  |  |  | Зельницкий Н.А. |
|  |  |  | *подпись* |  | *фамилия, и.о.* |

|  |  |  |  |
| --- | --- | --- | --- |
| Преподаватель |  |  | Кичигин А.А. |
|  | *подпись* |  | *фамилия, и.о.* |

1 Техническое задание

Устройство должно измерять температуру окружающей среды, выдавать ее на 4 семисегментных индикатора в формате "xx.xx" и записывать на sd карту с указанием метки времени в формате "<номер измерения> дд:мм:гггг чч:мм <температура>". В качестве датчика использовать RTD Pt100. В качестве микросхемы преобразователя данных с RTD использовать MAX31865. Измерение происходит каждый час. Должна быть предусмотрена возможность чтения данных записанных на sd карту на ПК (например, по интерфейсу RS-232). Так же на LCD дисплее отображается текущее время, количество измерений на SD карте. Должна быть предусмотрена возможность отображения сохраненных измерений на SD карте. Частота тактирования микроконтроллера ATmega-16 - 8МГц, питание микроконтроллера, индикаторов и дисплея LCD - 5В, периферийных устройств (MAX31865, SD карта, часы реального времени) - 3.3В.

2 Разработка схемы электрической структурной

Схема электрическая структурная приведена в приложении А.

Схема электрическая структурная состоит из следующих блоков:

1) кварцевый резонатор,

2) кнопки,

3) 4 семисегментных индикатора,

4) разъем SD,

5) микроконтроллер,

6) преобразователь UART RS-232,

7) разъем программирования SPI,

8) микросхема max31865,

9) разъем DB9,

10) датчик Pt100,

11) микросхема реального времени PCF 8583,

12) разъем питания 5В,

13) фильтр по питанию,

14) защита от переполюсовки,

15) линейный преобразователь напряжения.

Блок кварцевый резонатор предназначен для стабилизации тактовой частоты микроконтроллера.

Блок кнопки, состоит из 4-х кнопок подключенных к выводам PC3, PC5, PC6, PC7 микроконтроллера, назначение кнопок следующее: при удержании PC3 на семисегментные индикаторы выводится текущее время в формате "мм.сс", нажатие кнопки PC5 позволяет посмотреть следующую запись на SD карте, кнопка PC6 возвращает счетчик записей в начало (к первой записи), по нажатию PC7 все содержимое SD карты отправляется по UART на ПК.

Блок семисегментные индикаторы включает в себя четырехразрядный семисегментный индикатор для отображения температуры или времени.

Блок SD карты включает в себя SD карту, которая по интерфейсу SPI обменивается данными с микроконтроллером.

Блок микроконтроллер состоит из микроконтроллера ATmega-16PU, к нему подключены все остальные блоки.

Блок преобразователь UART RS-232, предназначен для подключения приемо-передатчика UART к физическому разъему DB9.

Блок разъем программирования SPI предназначен для программирования микроконтроллера.

Блок микросхема MAX31865 включает в себя микросхему MAX31865 представляющую собой 15-ти битный АЦП, обменивающийся с микроконтроллером по интерфейсу SPI.

Блок разъем DB9 предназначен для передачи данных между ПК и микроконтроллером.

Блок датчик Pt100 представляет собой датчик температуры Pt100, на основе сопротивления данного датчика и микросхемы MAX31865 микроконтроллер получает данные о температуре.

Блок микросхема реального времени PCF 8583 представляет собой часы реального времени, необходимые для фиксирования времени измерения температуры, а также часовых интервалов, обменивается с МК по интерфейсу I2C.

Блок разъем питания 5В предназначен для подключения питания к микроконтроллеру.

Блок фильтр по питанию необходим для фильтрации питания МК.

Блок защита от переполюсовки предназначен для защиты МК от переполюсовки питания.

Блок линейный преобразователь напряжения предназначен для питания периферийных устройств, таких как SD карта, и MAX31865.

3 Разработка схемы электрической принципиальной

Схема электрическая принципиальная и перечень элементов приведены в приложении А.

Блок кварцевый резонатор, представлен кварцевым резонатором ZQ1 8 МГц, кристалл подключён между выводами XTAL1/XTAL2, каждая нога заземлена через конденсаторы C16 и C17 по 22 пФ.

Блок кнопки, SB1 — аппаратный «RESET»: при нажатии замыкает вывод RESET МК на землю, а R7 10 кОм держит линию подтянутой к питанию в остальное время. SB2–SB5 — пользовательские. Каждый соединён с портами PC3, PC5, PC6, PC7. Перед микроконтроллером стоят токоограничивающие резисторы R15–R18 по 1 кОМ; сами выводы подтянуты к VDD внутренним pull-up МК, поэтому логика «активный 0».

Блок четырёхразрядный 7-сегментный индикатор, сам индикатор HG1 CC56-12SRWA подключён по классической схеме мультиплексирования: сегменты A…DP (всего 8 линий) подключены к порту A; общие катоды/аноды разрядов коммутируют транзисторы VT1–VT4 (маломощные NPN) через базовые резисторы R9–R12 10 кОм. Базы управляются портом PB0–PB3. МК быстро «перещёлкивает» VT1–VT4, выводя в тот же момент код сегментов.

Блок разъем SD, X1 «SD\_card» имеет выводы CMD (MOSI), CLK (SCK), DAT0 (MISO), CS, CD. На линиях данных стоят подтяжки R1, R3, R5, R6 по 10 кОм к шине 3V3, чтобы SD корректно себя вела до инициализации. Сразу у разъёма стоят конденсаторы C3, C4 0,1 µF для борьбы с импульсными бросками тока.

Передача идёт по SPI, линию SCK формирует PB7, MOSI — PB6, MISO — PB7, а PC4 служит выбором CS (совместно с MAX31865).

Блок микроконтроллер, чип разбит на два УГО D2.1/D2.2. Питается от +5 В (VCC): локальная развязка выполнена керамиками C2, C5, C7 по 0,1 µF прямо у ножек питания.

Блок преобразователь UART RS-232, Функциональный узел реализован одной микросхемой **MAX202CDR** (позиция D3) . MAX202 содержит два драйвера и два приёмника с интегрированным зарядовым насосом, формирующим на базе внешних конденсаторов **C1–C4 = 0,1 µF** уровни **V+ ≈ +9 В** и **V − ≈ −9 В** из однополярного питания **+5 В**. TTL-выводы **T1IN/R1OUT** связаны соответственно с линиями **TXD (PD1)** и **RXD (PD0)** микроконтроллера, а выходы **T1OUT/R1IN** подают сигналы уровня EIA-232 на разъём **DBR-9M (X2)**, контакты 2 (TX) и 3 (RX). Такой подход обеспечивает двусторонний обмен данными со скоростями до 120 кбит/с при гальванической развязке по потенциалам ±10 В, что соответствует спецификации TIA/EIA-232-F.

Блок разъем программирования SPI, Разъём **XP4 «PLD-4»** предоставляет стандартный шлейф AVR-ISP (MISO, MOSI, SCK, RESET, VCC, GND). Проводимость линий SPI обеспечивается непосредственно выводами МК (**PB5/MOSI**, **PB6/MISO**, **PB7/SCK**), а общий сброс формируется выводом **RESET** через подтяжку **R7 = 10 kОМ**. Такой интерфейс позволяет выполнять внутрисистемное программирование флэш-памяти ATmega16 без демонтажа микросхемы.

Блок микросхема max31865, Измерительный канал построен на специализированном Δ-Σ-АЦП MAX31865 (D1). Эталонный резистор R2 = 430 Ом формирует токовый источник ≈ 2,05 мА для платинового датчика. Четырёхпроводное подключение (пары FORCE ± и RTDIN ±) компенсирует омическое сопротивление линий. Декуплирующие конденсаторы C1, C6, C8 = 0,1 µF шунтируют аналого-цифровое и цифровое питания чипа. Управление осуществляется по интерфейсу SPI (SDI, SDO, SCLK), однако для защиты временных шин модуль имеет индивидуальный сигнал CS. MAX31865 выполняет 15-битное оцифровывание, что даёт разрешение ≈ 0,0076 °C на диапазоне 0–400 °C при Pt100.

Блок разъем DB9, разъём **DBR-9M (X2)** служит физическим интерфейсом к внешнему СOM-порту персонального компьютера, принимая уровни ±9 В с микросхемы MAX202. Назначения ножек выведены прямо на схеме: **2 — RS232TX**, **3 — RS232RX**, **5 — GND**.

Блок датчик Pt100, к контактам **FORCE ±** подаётся стабильный токовый источник чипа MAX31865, напряжение на выводах **RTDIN ±** считывается тем же чипом; возвратный токовый вывод **ISENSOR** позволяет измерителю контролировать реальный ток через датчик. Трех проводная схема исключает погрешность, вызванную омическим сопротивлением кабеля и разъёмов.

Блок микросхема реального времени PCF 8583, микросхема PCF8583P (D2) питается от стабилизированного напряжения 3,3 В и стабилизируется кварцевым резонатором ZQ2 = 32,768 кГц с нагрузочными конденсаторами C18, C19 = 22 пФ. Информационный обмен ведётся по шине I²C (SCL — PC0, SDA — PC1). Вывод NINT обеспечивает аппаратное прерывание будильника на вход МК PC2. Локальная фильтрация питания реализована конденсаторами C11–C13 = 0,1 µF, что позволяет RTC сохранять ход времени при импульсных помехах.

Блок разъем питания 5В, Коннектор XP3 (DJB-02B) имеет нумерацию: контакт 1 — +5 В, контакт 3 — GND . Далее напряжение поступает на цепи защиты и фильтрации, после чего распределяется по функциональным узлам.

Блок защита от переполюсовки, выполнен шоттки-диодом **VD1** в прямом включении между входом +5 В и потребителями. При обратном подключении источника питания диод запирается, ограничивая обратный ток до единиц микроампер и предотвращая выход из строя downstream-электроники. Применение шоттки-структуры минимизирует прямое падение (≈ 0,3 В) при штатной полярности.

Блок фильтр по питанию, представлен конденсатором C2.

Блок линейный преобразователь напряжения, LDO ADP165 (DA1) формирует шину 3V3 из входных +5 В . Типичный ток нагрузки — до 150 мА; остаточное падение напряжения ΔV ≈ 250 мВ при 100 мА. Конденсаторы C3, C4, C12 = 0,1 µF рекомендуются даташитом для устойчивости петли обратной связи и подавления шумов. Логический вход EN подтянут резистором R4 = 10 kОм к выходному напряжению, что обеспечивает автозапуск без внешнего сигнала. Сформированная шина 3,3 В питает SD-накопитель, MAX31865, PCF8583 и логические каскады индикации, что снижает тепловые потери и электромагнитные выбросы относительно прямого питания от 5 В.

4 Разработка алгоритма программы

Блок схема алгоритма программы представлена в приложении А.

Таблица 1 — Глобальные переменные объявленные в main

| Переменная | Тип | Назначение |
| --- | --- | --- |
| fwd\_hold\_cnt, bwd\_hold\_cnt | volatile uint16\_t | Сколько миллисекунд удерживается соответствующая кнопка «вперёд/назад»; используется для определения долгого нажатия. |
| fwd\_rep\_cnt, bwd\_rep\_cnt | volatile uint16\_t | Таймер повтора событий при удержании кнопок (автоповтор навигации). |
| ms\_ticks | volatile uint32\_t | Счётчик миллисекунд, увеличивается в прерывании TIMER0\_OVF; служит единой «системной» временной базой. |
| nav\_end\_event, nav\_start\_event | volatile uint8\_t | Одноразовые флаги «перейти в конец лога» / «в начало лога», выставляются на двойной клик соответствующих кнопок. |
| last\_sent\_meas | volatile uint32\_t | Кол-во записей, уже выгруженных по UART, чтобы продолжать нумерацию после дампа. |
| meas\_no | uint32\_t | Текущий номер измерения, считая с 1 (заполняется при старте размером файла SD). |
| nav\_pos | uint32\_t | «Курсор» при просмотре лога: номер строки, показанной на LCD. |
| rtc\_time | struct pcf\_time | Последнее прочитанное время из RTC PCF8583. |
| disp\_digits[4] | volatile uint8\_t[4] | Четыре BCD-разряда, которые мультиплексно отображаются на 7-сегментном индикаторе. |
| linebuf[32], linebuf\_hist[32] | char[32] | Рабочие буферы для текущей строки лога (основной/исторический режим). |
| scroll\_buf[32], scroll\_len, scroll\_ofs, scroll\_t0 | char[32], uint8\_t, uint8\_t, uint32\_t | Хранят текст бегущей строки на LCD и параметры её прокрутки. |
| last\_sent\_entry | uint32\_t | Запоминает номер последней строки, успешно отправленной по UART (для прогресс-бара во время дампа). |

Таблица 2 — Глобальные массивы с областью видимости во всем проекте

| Модуль | Массив | Назначение |
| --- | --- | --- |
| main.c | disp\_digits | Данные для 7-сегментника. |
| sdcard.c | sec\_buf[512] | RAM-буфер целого сектора SD-карты; заполняется перед чтением/записью. |
| sevseg.c | dec2sevseg[10] | Таблица кодов сегментов для цифр 0–9. ctrl\_pins[4] —  маски выводов катодов/анодов. |

Файл main.c — это центральный модуль, управляющий всей работой измерительного стенда на МК ATmega16

Подключаемые файлы и стандартные заголовки: в начале подключаются пакеты для работы с семисегментными индикаторами, MAX31865, шины TWI, часов RTC PCF8583, SD-карты, LCD 16×2, а также стандартные заголовки AVR (avr/io.h, avr/interrupt.h, util/delay.h, stdlib.h, string.h). Благодаря этим включениям в main.c почти нет «ручной» работы с аппаратными регистрами — исключение составляет настройка таймеров и портов ввода-вывода.

Макросы и константы: частота F\_CPU задаётся 8 000 000 Гц, чтобы функции \_delay\_ms() и \_delay\_us() рассчитывали правильные паузы. Определены маски кнопок (PC3 — переключение режима, PC5 и PC6 — навигация вперёд / назад, PC7 — дамп журнала и регистр маски всех семисегмент-катодов. Для автоповтора кнопок заданы задержка перед первым повтором 250 мс, базовый период повторов 500 мс и минимально допустимый период 1 мс. Для бегущей строки на LCD заданы пауза в начале и конце 1000 мс и шаг прокрутки 250 мс.

Глобальные переменные ms\_ticks — глобальный счётчик миллисекунд, который увеличивается в прерывании TIMER0. fwd\_hold\_cnt и bwd\_hold\_cnt — время удержания кнопок «вперёд» и «назад» в миллисекундах. fwd\_rep\_cnt и bwd\_rep\_cnt — обратные таймеры автоповтора для тех же кнопок. last\_sent\_meas — число строк, уже переданных на ПК; нужно, чтобы после дампа продолжить нумерацию журнала. scroll\_t0, scroll\_len и scroll\_ofs — параметры бегущей строки (момент запуска, общая длина текста и текущий сдвиг). Структура flags (8 бит) содержит флаги: one\_sec, ten\_sec, nav\_fwd, nav\_bwd, uart\_dump, btn\_lock, history, hist. Первый выставляется каждую секунду, второй — раз в десять секунд (для проверки) или раз в час согласно тз, следующие два инициируют навигацию на одну строку, uart\_dump стартует выгрузку лога в UART, btn\_lock временно блокирует действия кнопок во время записи или дампа, history сообщает, что пользователь нажал кнопку переключения режима, а hist хранит текущий режим «история активна / неактивна». disp\_digits[4] — четыре ВCD-цифры для семисегментника. linebuf — строка с актуальным измерением, выводимая во втором ряду LCD в режиме Live. linebuf\_hist — строка, показанная в режиме History. scroll\_buf[32] — копия той же строки для плавной прокрутки.

Вспомогательные функции bcd2dec() преобразует один BCD-байт из регистра часов в десятичное число. time2str() формирует строку «ЧЧ:ММ:СС» из структуры времени RTC.

temperature\_to\_digits\_tenth() переводит число с плавающей точкой (градусы) в четыре BCD-разряда с одной десятичной точкой и знаком, помещая результат в disp\_digits. go\_to\_log\_start() и go\_to\_log\_end() позиционируют взаимодействующий с SD-картой итератор на первую или последнюю запись журнала, считывают строку, обрезают перевод строки, перерисовывают вторую строку LCD и запускают скролл.

scroll\_update() каждые 250 мс сдвигает 16-символьное «окно» по scroll\_buf и печатает его на LCD; в начале и в конце строки выдерживается секундная пауза. uart\_putc() и uart\_puts() передают байт или строку по UART, опрашивая флаг готовности UDRE. uart\_dump\_log() перебирает все записи SD-карты, отправляет их в UART, выводит прогресс на LCD, корректирует счётчик last\_sent\_meas, а в конце очищает SD-файл.

Обработчики прерываний ISR(TIMER1\_COMPA\_vect) срабатывает раз в 1024 мс, поднимает флаг one\_sec; через 10/60 итерации устанавливает ten\_sec. ISR(TIMER0\_OVF\_vect) тикает каждую миллисекунду. Он обслуживает динамическую индикацию семисегментника (если не идёт дамп), увеличивает ms\_ticks, опрашивает кнопки, ловит фронты, измеряет длительность удержания и двойные клики, рассчитывает период автоповтора и выставляет соответствующие флаги навигации, переключения режима или запуска дампа.

Инициализационные функции: uart\_init() включает передатчик и приёмник, настраивает скорость 56 кбод в режиме удвоенной скорости. timer0\_init() задаёт предделитель 64, разрешает прерывание TOIE0. timer1\_init() включает режим CTC, ставит делитель 256 и значение OCR1A = 1023, что даёт ровно 1024 тиков F\_CPU и интервал ≈ 1,024 с. port\_init() переводит PC3, PC5, PC6 и PC7 во входы с подтяжками; остальные линии настраиваются драйверами периферии.

Ход работы функции main(): после подключения заголовков и объявления глобальных переменных вызываются: uart\_init, port\_init, sevseg\_init, lcd\_init. На LCD временно выводится цифра 1 как индикатор этапа загрузки. Затем выполняется max31865\_init (на LCD появится цифра 2), функция timer0\_init, функция timer1\_init, далее — sd\_iter\_reset (логический «курсор» ставится на конец файла на SD-карте), twi\_init, глобальное разрешение прерываний sei() и инициализация RTC pcf\_init(). Число уже существующих записей считывается функцией sd\_get\_entry\_count и сохраняется в переменную meas\_no.

Далее запускается бесконечный цикл. Если установлен флаг history, программа меняет текущий режим: если режим History выключен, она копирует текущую строку лога из linebuf\_hist на LCD и подготавливает скролл; если включён — возвращается к режиму Live. При флаге one\_sec считывается время через TWI из RTC, запускается преобразование RTD-значения в температуру, число формируется на семисегментнике, LCD получает строку «ЧЧ:ММ:СС #N». При флаге ten\_sec (и если кнопки не заблокированы) формируется строка «HH:MM:SS;T=xx.x», записывается на SD-карту, копируется в linebuf\_hist, а режим History при необходимости обновляет индикацию и бегущую строку; после записи кнопки вновь разрешены. При флагах nav\_fwd или nav\_bwd итератор двигается на одну запись вперёд или назад; при nav\_\*\_event (двойной клик) он прыгает в начало или конец журнала соответственно. При флаге uart\_dump вызывается uart\_dump\_log, после чего SD-карта очищается, а переменные счётчика и буферы обновляются. Каждую итерацию цикла вызывается scroll\_update, которая сдвигает бегущую строку.

Алгоритм обработки кнопок и автоповтора ISR TIMER0 обнаруживает фронт кнопки, сохраняет время нажатия, через 250 мс активирует первую автоповторную прокрутку и далее уменьшает интервал до 1 мс пропорционально объёму журнала. Второй фронт, пришедший раньше 70 мс после первого, трактуется как двойной клик; это поднимает специальные флаги nav\_start\_event или nav\_end\_event.

Сенсор через MAX31865 работает в непрерывном режиме; каждую секунду считывается регистр RTD, преобразуется в температуру и отображается. Каждые десять секунд строка журнала записывается на SD; при удержании кнопки PC7 весь журнал побайтно передаётся в UART и затем файл на карте очищается. Семисегментник обновляется из ISR TIMER0, а LCD обновляется только из главного цикла, чтобы не нарушать тайминги дисплея.

Назначение модуля: sevseg.c реализует полный пакет для работы четырёхразрядного семисегментного индикатора. Драйвер выполняет инициализацию портов, преобразует двоичное 16-битное число в BCD-4, обеспечивает циклический вывод разрядов с возможностью гашения ведущих нулей, установки десятичной точки и отображения знака «минус».

Аппаратная привязка: в начале файла заданы макросы DATA\_PORT/ DATA\_DDR, указывающие на порт A, к которому подключены сегменты, и CTRL\_PORT/CTRL\_DDR, указывающие на порт B, где расположены управляющие выводы общих анодов/катодов разрядов. Для индивидуальных линий выбраны биты PB0…PB3, а маска CTRL\_PIN\_MASK объединяет их для массовых операций. Эта схема даёт возможность за один такт гасить все разряды или включать любой из них.

Глобальные статические параметры: dot\_pos хранит номер разряда (0–3), возле которого следует зажечь точку; значение 0xFF означает, что точка выключена. Флаг blank\_leading определяет, нужно ли подавлять вывод ведущих нулей. Для знака «минус» зарезервирован сегментный шаблон SEVSEG\_MINUS\_CODE = 0b00000010, соответствующий зажиганию только среднего горизонтального сегмента.

Константные таблицы: ctrl\_pins[4] и dec2sevseg[10] помещены в flash-память (PROGMEM), чтобы не расходовать ОЗУ. Первая таблица содержит готовые маски включения нужного разряда, вторая — коды сегментов для цифр 0…9. Использование pgm\_read\_byte() исключает лишнее копирование в SRAM и ускоряет доступ.

Инициализация: sevseg\_init() переводит все восемь линий порта A и четыре линии порта B в режим выхода, гасит сегменты (DATA\_PORT = 0) и разряды (CTRL\_PORT &= ~CTRL\_PIN\_MASK). В конце дополнительно активируется первый индикатор; замечание TODO в комментарии предлагает проверить необходимость этой строки, но она полезна как тест на целостность схемы при старте.

Преобразование числа: sevseg\_bin2bcd() разбивает 16-битное беззнаковое значение на тысячи, сотни, десятки и единицы простой последовательностью делений и модулей. Заполнение массива идёт от старшего разряда к младшему, поэтому достаточно один раз сдвинуть указатель в конец и декрементировать его по мере работы. Максимально поддерживаемое число — 9999.

Внутренний счётчик разрядов: статическая функция inc\_number\_digit() инкрементирует локальную переменную num\_dig и маскирует её по & 0x03, получая циклическую последовательность 0 → 1 → 2 → 3 → 0. Такой счётчик позволяет вызывать диспетчер вывода из прерывания таймера с частотой ≈1 кГц, обеспечивая частоту обновления каждого разряда ≈250 Гц без избытка вычислений. sevseg

Отрисовка одного разряда: display\_digit() сначала гасит всё (обнуляет порт данных и снимает все управляющие биты), затем выбирает нужный символ: если код ≤ 9, берётся шаблон из dec2sevseg; если код 0x0A, отображается «минус»; любые иные коды игнорируются, что даёт универсальный «пустой» символ. Для подавления лидирующих нулей функция проверяет, нет ли ненулевых разрядов старше текущего; если их нет и цифра равна нулю, разряд просто остаётся выключенным. При совпадении num\_dig с dot\_pos в шаблон OR’ом добавляется бит десятичной точки (LSB). После формирования маски сегментов она выводится в DATA\_PORT, а в CTRL\_PORT устанавливается бит выбранного разряда. Cхема алгоритмаCхема алгоритма

Процесс отображения: sevseg\_display\_process() является внешним интерфейсом для приложения. На каждый вызов она запрашивает очередной номер разряда у inc\_number\_digit() и передаёт управление display\_digit(). Таким образом за четыре последовательных вызова (например, четыре тика таймера) отрисовывается полное число.

Выключение: sevseg\_off() сбрасывает DATA\_PORT, мгновенно гасит все сегменты и может использоваться для экономии энергии во время длительного сна МК или при передаче буфера UART, когда световой шум нежелателен.

Взаимодействие с остальной прошивкой: главный модуль (main.c) предварительно преобразует температуру или счётчики в BCD-4 через sevseg\_bin2bcd(), затем помещает результат в глобальный массив disp\_digits. Прерывание TIMER0 вызывает sevseg\_display\_process(disp\_digits) с частотой 1 кГц; если идёт дамп журнала, отображение временно останавливается, и порт A обнуляется.

Назначение модуля: max31865.c формирует полностью автономный драйвер SPI-конвертера, обеспечивая инициализацию, считывание «сырых» данных RTD, перевод сопротивления сенсора Pt100 в температуру по уравнению Каллендара–Ван Дюзена, а также обработку и сброс флагов неисправностей.

Аппаратная привязка: вывод CS приёмника назначен PB4; макросы MAX31865\_CS\_PORT, MAX31865\_CS\_DDR, MAX31865\_CS\_PIN задают этот вывод и используются в inline-процедурах cs\_low() и cs\_high() для формирования SPI-кадров .

Регистровая карта: заголовок описывает восьмибитные адреса 0×00…0×07; конфигурационный регистр (адрес 0) содержит биты CFG\_BIAS, CFG\_MODE\_AUTO, CFG\_3WIRE, CFG\_FILT50HZ и проч., а регистр RTD\_MSB/LSB (адреса 1/2) хранит 15-битное измеренное значение сопротивления.

Калибровочные константы: номинальное сопротивление RTD R0 = 100 Ω, опорный резистор платы RREF = 430 Ω, а также коэффициенты A, B и C классического полинома Pt100 (значение C используется только ниже 0 °C).

Базовые операции SPI: функции write\_reg() и read\_reg() формируют однобайтовые транзакции (бит 7 адреса отвечает за признак записи), read\_reg16() расширяет процедуру до приёма двух последовательных байт; каждая транзакция обрамляется опусканием и подъёмом CS, что гарантирует атомарность обмена.

Процедура инициализации: max31865\_init(three\_wire, filter50Hz) настраивает CS-пин как выход, активирует общий драйвер SPI, производит начальную инициализацию SD-карты, формирует байт cfg, в который при необходимости включаются биты трёхпроводной компенсации и 50-герцового фильтра, далее включает ток RTD (CFG\_BIAS), выдерживает 10 мс для термостабилизации и переводит микросхему в непрерывный режим измерения (CFG\_MODE\_AUTO).

Чтение «сырых» данных: max31865\_read\_raw() извлекает 16-битное поле RTD, сдвигает его вправо на 1 бит (убирая флаг ошибки) и маскирует до 15 бит – результат соответствует диапазону 0…32767 .

Нормирование сопротивления: функция max31865\_read\_temperature() умножает «сырое» число на RREF/32768, получая реальное сопротивление Rt; далее вычисляет относительное значение Z = Rt/R0 .

Диагностика: max31865\_read\_fault() читает регистр FAULT\_STATUS и возвращает маску флагов обрыва, короткого замыкания и других сбоев; max31865\_clear\_fault() временно останавливает преобразования, записывает в CFG\_CLEAR\_FAULT, затем перезапускает режим bias + auto, полностью очищая регистр ошибок max31865max31865.

Интеграция с прошивкой: главный цикл проекта извлекает температуру через max31865\_read\_temperature() раз в секунду и передаёт её в процедуру temperature\_to\_digits\_tenth() для отображения; кроме того, «сырое» значение может быть журналировано на SD для последующего анализа. Модуль не содержит блокирующих задержек, кроме 10-миллисекундной паузы в инициализации, и поэтому не препятствует выполнению остальных задач системы.

Назначение модуля: sdcard.c реализует фундаментальный слой доступа к съёмному энергонезависимому запоминающему устройству стандарта Secure-Digital в режиме SPI. Принята парадигма «одна строка журнала — один логический сектор»: каждое сообщение длиной до 510 байт помещается целиком в сектор объёмом 512 байт, завершается символом LF, а оставшееся пространство заполняется маркером 0xFF — этот приём избавляет от расчётов смещения внутри блока и ускоряет навигацию по журналу.

Аппаратная привязка и базовые макросы: линия Chip-Select карты подключена к выводу PC4. Шаблоны spi\_x() (полудуплексная передача) опираются на универсальный драйвер spi\_transfer() и используются во всех транзакциях протокола SD SPI.

Глобальные структуры данных: статический буфер sec\_buf[512] хранит содержимое текущего сектора; флаг card\_sdhc различает карты адресных пространств SDSC (байтовая адресация) и SDHC (блочная адресация); счётчики next\_sector и it\_sector задают точку очередной записи и позицию итератора чтения соответственно; переменная eeprom\_next\_sector, размещённая в области EEPROM, обеспечивает устойчивость журнала к выключениям питания .

Командный интерфейс: функция sd\_cmd() формирует пять-байтную команду SD-SPI (префикс 0x40|CMD, аргумент 32 бит, контрольная сумма) и далее опрашивает карту до появления ответа с битом ready=0 . Такой неспешный 10-байтовый цикл выбран как компромисс между латентностью и совместимостью с медленными носителями.

Процедура инициализации (sd\_init): 1) CS переводится в высокий уровень (passive); 2) передаются 80 тактных импульсов SCK «вхолостую» для выхода карты из приёмника ; 3) выполняется команда CMD0 — переход в состояние IDLE; 4) через CMD8 проверяется поддержка напряжения 3,3 В и версии V2.0; 5) циклическим ACMD41 карта выводится из IDLE с возможным указанием бита HCS; 6) CMD58 считывает регистр OCR, по 30-му биту которого устанавливается тип SDHC/SDSC . Завершается инициализация чтением сохранённого в EEPROM номера хвостового сектора; при отсутствии корректного значения задействуется резервная процедура поиска хвоста (find\_tail).

Низкоуровневое чтение: read\_sector(lba) запрещает обращение к логическому сектору 0 (защита MBR), затем, учитывая тип карты, конвертирует LBA в адрес (для SDSC — умножение на 512), посылает CMD17, ждёт токен 0xFE не более 100 мс, и считывает 512 байт полезных данных плюс фиктивный CRC .

Низкоуровневая запись: write\_sector(lba) выполняет пред-ожидание готовности шины (wait\_ready ≤ 250 мс), посылает CMD24, токен 0xFE, пересылает содержимое буфера и две фиктивные контрольные суммы, после чего анализирует поле data-response (ожидается 0b101) и вновь ждёт выхода карты из состояния busy.

Логический формат хранилища: строка измерения копируется в начало sec\_buf, после чего запись завершается переводом строки '\n', а оставшиеся байты сектора инициализируются 0xFF. Ограничение 510 байт гарантирует, что вместе с \n и заполняющим шаблоном объём строго равен 512 байтам. Длина 0 запрещена — это рассогласовывает нумерацию и даёт код ошибки 4 sdcard.

Функции журналирования: sd\_write\_line() сохраняет сектор по адресу next\_sector, затем атомарно увеличивает счётчик и асинхронно обновляет копию в EEPROM, обеспечивая сохранность индекса даже при внезапном обесточивании sdcardsdcard. sd\_clear\_log() заполняет заданное число секторов 0xFF и сбрасывает указатель хвоста в 1.

Итеративное чтение: sd\_iter\_reset() обнуляет позицию, а sd\_read\_line(dir, dst, sz) реализует двунаправленный алгоритм: при dir>0 сектора читаются по возрастанию, при dir<0 — по убыванию, пропуская пустые блоки 0xFF; данные копируются функцией copy\_line(), которая отсекает либо символ LF, либо маркер конца блока — 0xFF sdcardsdcard. Функция sd\_read\_line\_at() поддерживает прямую адресацию по номеру записи; диапазон контролируется относительно динамического next\_sector для предотвращения вычитки вне текущего журнала.

Обнаружение хвоста: резервная функция find\_tail() (в данный момент закомментирована ради ускорения старта) последовательно просматривает карту от сектора 1, пока не встретит полностью пустой блок 0xFF; найденное значение становится новым next\_sector.

Хранение указателя хвоста: использование ячейки EEPROM как однонаправленного счётчика позволяет не тратить ресурс перезаписи — счётчик инкрементируется строго вперёд и обнуляется только при ручном очистке лога, что соответствует циклическому ресурсу >10⁵ записей.

Синхронизация по времени: задержки wait\_token() и wait\_ready() реализованы на основе миллисекундных busy-loop с \_delay\_ms(1), что обеспечивает детерминированную верхнюю границу ожидания и минимизирует влияние скорости конкретного носителя.

Обработка ошибок: все публичные API возвращают ненулевой код при отказе, детализируя причину (тайм-аут, некорректный ответ, попытка записи в LBA 0, выход за границы журнала, превышение длины строки и т.д.); значения кодов документированы в комментариях возле каждого return sdcardsdcard.

Взаимодействие с системой: модуль вызывается из главного цикла для периодического sd\_write\_line(), из навигационных обработчиков для sd\_read\_line() и из процедуры дампа -‐ uart\_dump\_log(). Благодаря полносекторному формату запись никогда не требует предварительного чтения, а чтение — лишних копирований, что снижает латентность и упрощает энергобаланс микроконтроллера при работе с картой.

Модуль TWI: драйвер формирует полный стек обмена по дву-проводной шине I2C в режиме Master-transmitter/Master-receiver с частотой 100 кГц. После вызова twi\_init() регистр TWBR вычисляется как (F\_CPU / LINE\_FREQ – 16)/2, что при 8 МГц генерирует требуемый SCL≈100 кГц. Биты TWEN и TWIE одновременно включают модуль и разрешают прерывание по любому событию на шине; глобальный маркер twi\_status переводится из состояния “не инициализирован” в TWI\_STATUS\_READY. Для каждой транзакции высокоуровневые процедуры twi\_write() и twi\_read() вызывают скрытую twi\_txrx(), которая заполняет структуру twi { SLA+W, начальный регистр, указатель буфера, счётчик байт } и запускает состояние START, записав в TWCR комбинацию TWINT|TWSTA|TWEN|TWIE. Сразу после запуска twi\_status выставляется в TWI\_STATUS\_TX\_BUSY либо TWI\_STATUS\_RX\_BUSY, что позволяет остальной прошивке опрашивать завершение операции.

Вектор ISR(TWI\_vect) реализует конечный автомат: код состояния (TWSR & 0xF8)>>3 служит индексом в таблице указателей функций line\_actions\_table[LINE\_CNT] — таким образом логика развёрнута в компактный jump-table в PROGMEM. Базовые под-процедуры включают line\_send\_slaw(), line\_send\_reg\_addr(), line\_tx\_data() (передача данных при записи или генерация повторного START для чтения), line\_rx\_data() и line\_rx\_last\_data(); есть и line\_error\_stop(), замыкающая TWI на STOP при любой ошибке — потеря арбитража, NACK или bus-error .

При передаче драйвер сначала посылает SLA+W, получает ACK, отсылает адрес регистра, а затем n-байт полезных данных; когда счётчик reach 0, ISR выставляет TWSTO, а twi\_status переключается на TWI\_STATUS\_TX\_COMPLETE. При чтении после повторного START и SLA+R приём каждого байта подтверждается ACK, пока не останется последний — для него ISR очищает TWEA, тем самым формируя NACK и закрывая цикл; по приходу последнего байта выставляется STOP и статус TWI\_STATUS\_RX\_COMPLETE. Любая ошибка сразу переводит twi\_status в TWI\_STATUS\_ERROR, что гарантирует выход из неопределённого состояния шины.

Таким образом, обмен выполняется полностью неблокирующим образом: основной код лишь инициирует twi\_write/twi\_read и затем асинхронно ждёт смены twi\_status, как это сделано в main.c перед использованием свежих данных часов реального времени twi.

Модуль PCF8583: заголовок pcf8583.h определяет SLA-адрес 0xA0 (байт SLA+W), внутренние адреса регистров PCF8583 и компактную структуру pcf\_time { seconds, minutes, hours }. Функция pcf\_init() формирует массив из пяти байтов: секунды, минуты, часы, дата/месяц и день недели. Значения кодируются прямо из макросов TIME и DATE, переводя каждую цифру в BCD; дата конкатенируется по правилу производителя (два бита номера года, поля даты и месяца). Затем twi\_write() передаёт массив в регистр SEC устройства, тем самым одновременно за­давая полный набор временных счётчиков. Функция блокируется busy-loop’ом до тех пор, пока ISR не поставит twi\_status = TWI\_STATUS\_TX\_COMPLETE, после чего драйвер возвращается к состоянию READY, что гарантирует, что часы действительно приня­ли новые данные twimain.

Для выборки времени используется pcf\_read\_time(): она вызывает twi\_read() с тем же базовым адресом SEC и размером структуры pcf\_time. Из-за неблокирующего характера TWI чтению не требуется ждать — главный цикл опрашивает twi\_status, и как только ISR завершит транзакцию, данные уже находятся в буфере, готовые к обработке (преобразованию bcd→dec и отображению) pcf8583.

В совокупности два модуля образуют иерархию: низкоуровневый twi.c реализует реентерабельный конечный автомат I2C, а pcf8583.h использует его абстракцию для инициализации и периодического опроса аппаратных часов PCF8583 без участия задержек в основном потоке программы, что соответствует требованиям реального времени и минимизирует джиттер в цикле измерений.

Назначение модуля lcd16x2: реализация низкопроизводительного драйвера символьного индикатора HD44780-совместимого формата 2×16 знаков, работающего в 4-битном полудуплексном режиме. Модуль обеспечивает программное (bit-bang) формирование полубайтового интерфейса, начальную и вторичную инициализацию контроллера, передачу команд и данных, управление курсором и простые высокоуровневые сервисы вывода буфера/строки.

Структура аппаратной привязки: сигнал RS размещён на PD2, строб E на PD3, линии данных DB7…DB4 — на PD7…PD4 (старшие биты порта D). Маска LCD\_DATA\_MASK==0xF0 позволяет атомарно обновлять лишь нужные линии, не затрагивая нижние четыре бита порта.

Алгоритм инициализации: функция lcd\_init() после 15-мс паузы, описанной в datasheet, трижды посылает полубайт 0b0011 (команда «8-битный интерфейс»), постепенно сокращая паузу до 100 мкс, затем однократно — 0b0010 (переход в 4-битный протокол). Далее передаются полноформатные команды Function Set (0b0010 0100 — двухстрочный, 5×8 точек), Clear Display и Display ON; строгие задержки 1,5 мс/50 мкс соблюдаются документированными вызовами \_delay\_ms/\_delay\_us.

Примитивы передачи: lcd\_send\_hfbyte() выставляет старший полубайт на шину, генерируя фронт E→0—1, тем самым защёлкивая данные в HD44780. lcd\_send\_byte() упаковывает полный байт, вызывая hfbyte-процедуру дважды с длительностью 50 мкс. lcd\_send\_cmd()/lcd\_send\_char() различаются лишь состоянием RS: команда «0», данные «1» .

Высокоуровневые сервисы: lcd\_disp\_buf() и lcd\_disp\_str() выводят массив фиксированной длины или С-строку соответственно, пузырьковой логикой «post-increment», что экономит четыре такта AVR на циклическом условии lcd16x2. lcd\_mov\_cursor() конвертирует абсолютную позицию 0…31 в два окна DDRAM и формирует адресную команду (1<<LCD\_DDRAM)|addr; таким образом достигается унификация двух строк дисплея.

Чистая программо-ориентированная модель задержек (busy-flag HD44780 не читается) облегчает разводку платы — требуется только выходное соединение; побочным эффектом является гарантированная верхняя граница длительности операций, критичных для real-time цикла измерений.

Назначение модуля spi: предоставление минимального, но полнофункционального драйвера аппаратного контроллера SPI периферии ATmega16 для режима Master, CPOL=1, CPHA=1 (SPI-mode-1). Константа fCLK/16 (500 кГц при F\_CPU = 8 МГц) задаётся битом SPR0, обеспечивая безопасную частоту для SD-карт и MAX31865.

Инициализация: spi\_init() конфигурирует PB5 (MOSI) и PB7 (SCK) как выходы, PB6 (MISO) как вход; регистр SPCR получает маску (SPE|MSTR|CPOL|CPHA|SPR0). Такой набор одновременно включает модуль, фиксирует полярность SCK «неактивный 1» и выборку данных по заднему фронту, согласуя фазы со слэйвами SD и MAX31865, указанные в их datasheet .

Базовая транзакция: spi\_transfer() заносит байт в SPDR, циклически опрашивает SPSR & SPIF и немедленно возвращает принятый байт. Отсутствие побочных эффектов (функция нечувствительна к прерыванию) допускает вызовы из ISR и из основного потока без аренды шины — координация доступа возложена на вышестоящие модули SD и MAX31865 lcd16x2spi.

Верхнеуровневое использование: В SD-драйвере spi\_transfer() участвует в формировании CMD-кадров, в приёме токенов 0xFE и обработке busy-time-out, что иллюстрируется блок-схемой SPI стр. 1–2 • В max31865.c вызовы spi\_transfer() лежат в основе write\_reg/read\_reg16, где на уровне CS-процедур создаётся атомарный «транзакционный» контур Cхема алгоритмаCхема алгоритма.

5 Проверка программы в симуляторе – результаты компиляции, описание методики проверки в симуляторе и результаты самой проверки.

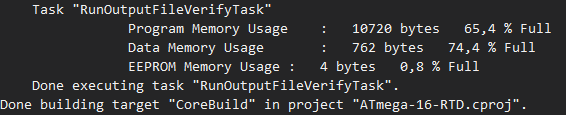


Рисунок 1 — Результат компиляции, количество используемой памяти

Из рисунка , видно, что используется 10720 байт памяти программ, 762 байта памяти данных, 4 байта EEPROM памяти.

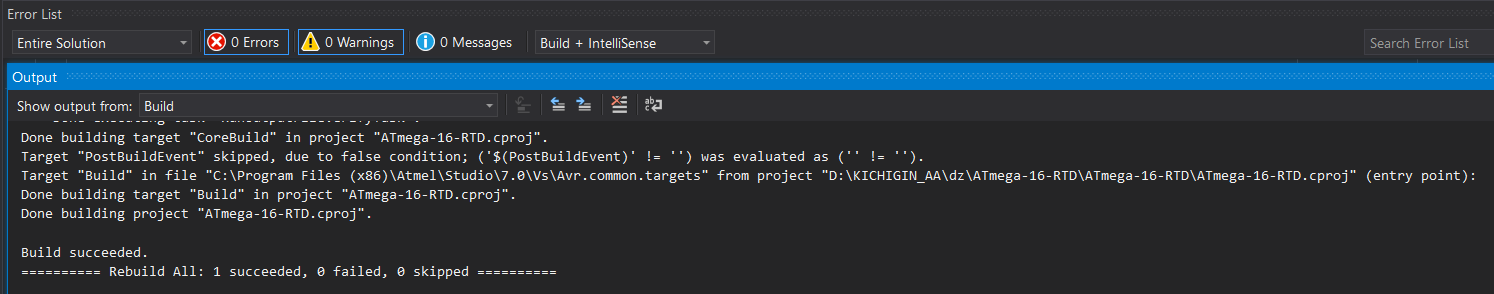


Рисунок — Результат компиляции, предупреждения и ошибки

Из рисунка видно, что компиляция завершена успешно, ошибок и предупреждений нет.

Методика проверки в симуляторе: собрать в симуляторе Proteus схему в соответствии со схемой электрической принципиальной, проверить ее на правильность работы.

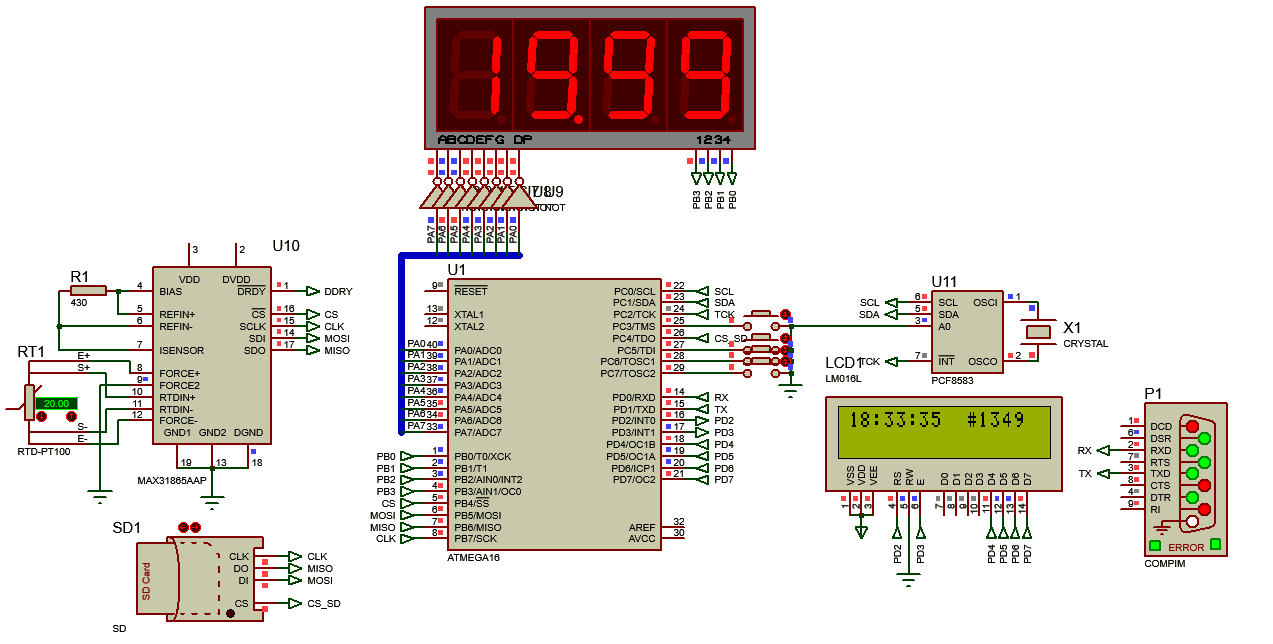


Рисунок — Схема устройства в симуляторе Proteus

На схеме на рисунке 3 показана схема устройства в симуляторе Proteus, данная схема полностью повторяет схему реального устройства, что позволяет проверить работу устройства сразу целиком.

При проверке устройства в симуляторе необходимо: запустить симуляцию, по умолчанию устройство включается сразу и на индикаторах отображается текущая температура с микросхемы MAX31865,

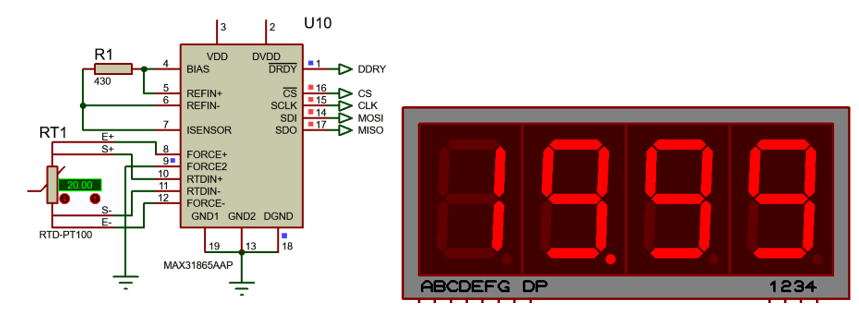


Рисунок — Проверка устройства в симуляторе, сразу после включения

затем необходимо убедиться в том, что изменение температуры на датчике приводит к изменению отображаемой температуры на семисегментных индикаторах,



Рисунок — Проверка устройства в симуляторе, реакция на изменение температуры

после этого необходимо убедиться в корректности записи данных на SD карту, для этого необходимо путем нажатия кнопок PC5, PC6 пролистать журнал записей и сравнить с показаниями датчика температуры,

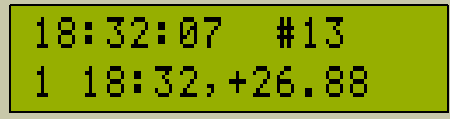


Рисунок — Проверка устройства в симуляторе, чтение с SD карты, 1 отчет

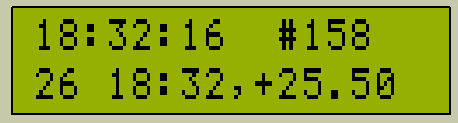


Рисунок — Проверка устройства в симуляторе, чтение с SD карты, 26 отчет

так же в корректности данных записываемых на SD можно убедиться, открыв файл sdcard.mmc в текстовом редакторе,

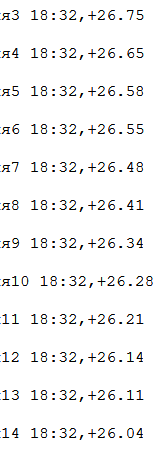


Рисунок — Проверка устройства в симуляторе, чтение с SD, файл sdcard.mmc

после этого необходимо проверить удержание кнопок PC5, PC6, при их удержании журнал должен начать быстро перелистываться, так же при нажатии PC7 МК переходит в режим отправки данных по UART и на экране должна отобразится надпись UART.



Рисунок — Проверка устройства в симуляторе, передача данных по UART

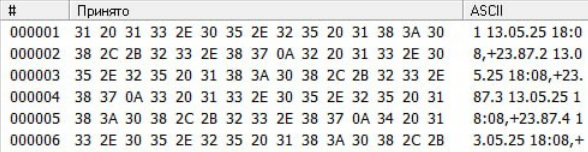


Рисунок — Проверка устройства в симуляторе, принятые по UART данные

устройство должно отправлять данные на ПК единожды, то есть после отправки данных, вся информация на SD карте стирается, сохраняется только номер последнего измерения, чтение журнала и отправка данных, будет доступна только начиная с этого номера, при нажатии PC3 устройство переходит в режим отображения текущего измерения на LCD дисплее.

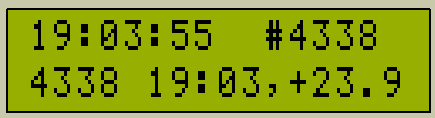


Рисунок — Проверка устройства в симуляторе, проверка работы режима отображения текущего измерения

Таким образом результаты проверки работы программы в симуляторе показали, что работа программы в симуляторе полностью соответствует техническому заданию.

6 Проверка устройства на стенде – описание методики проверки на стенде EasyAVR6, настройки программатора, результаты проверки.

Методика проверки на стенде: собрать устройство согласно схеме электрической принципиальной, включить стенд, на экране должно отобразиться время, на семисегментных индикаторах - температура.

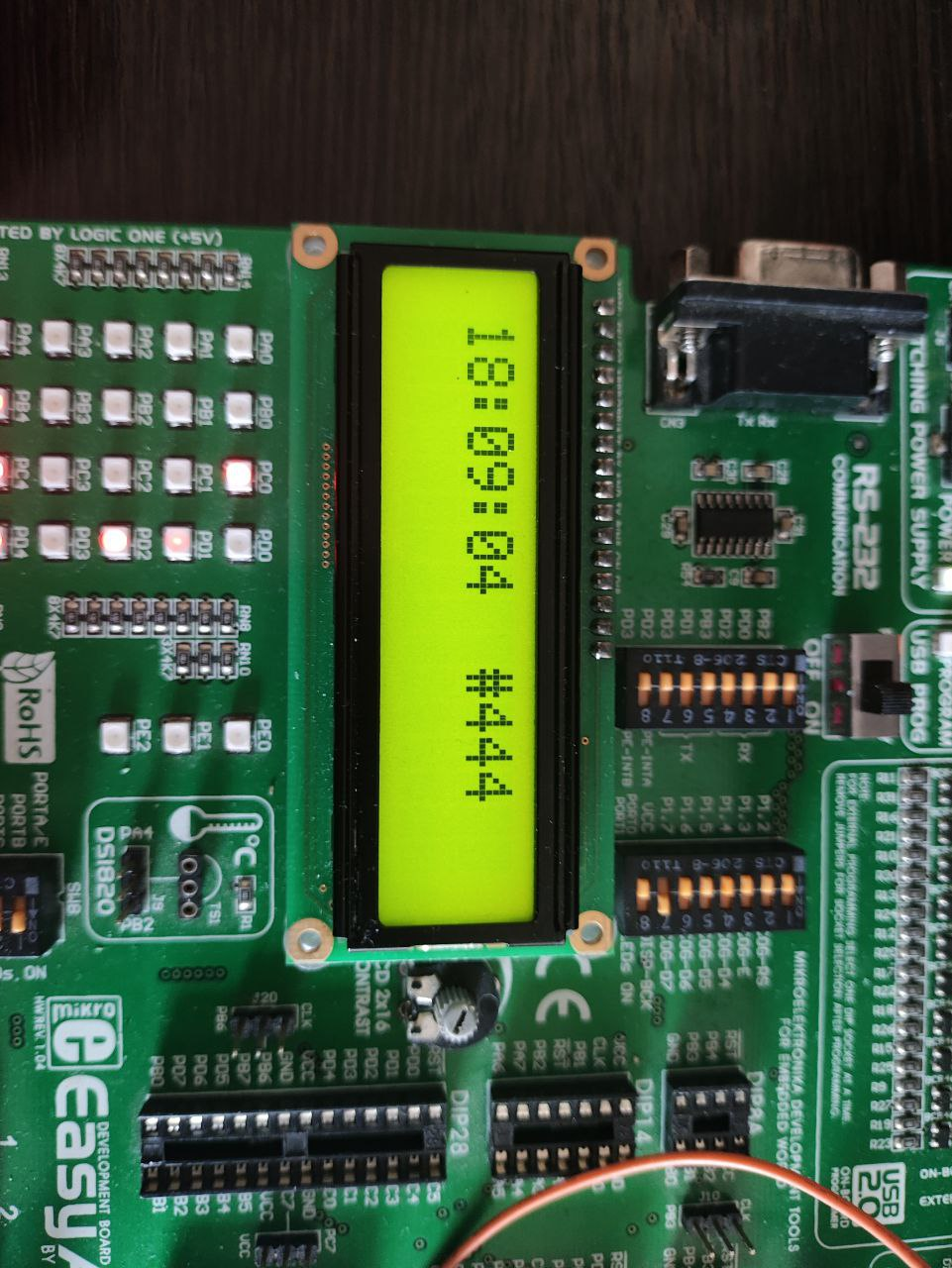


Рисунок 12 — Проверка устройства на стенде, сразу после включения



Рисунок 13 — Проверка устройства на стенде, сразу после включения, показания температуры

Проверить, что температура на индикаторе соответствует температуре датчика Pt100, затем нажатиями PC5 PC6 убедиться, что есть возможность чтения с SD карты, как вперед, так и назад, по двойному нажатию можно попасть, в конец/начало журнала.

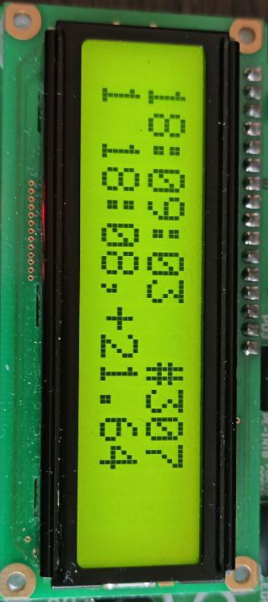


Рисунок 14 — Проверка устройства на стенде, отображение первой записи на SD карте



Рисунок 15 — Проверка устройства на стенде, отображение 17-й записи

При нажатии PC3 устройство должно переходить в режим истории и вторая строка дисплея должна отображать текущее измерение.



Рисунок 16 — Проверка устройства на стенде, режим истории

Проверить, что при отключении одного из периферийных устройств, измерения прекращаются.



Рисунок — Проверка устройства на стенде, нет соединения с микросхемой MAX 31865



Рисунок 18 — Проверка устройства на стенде, нет соединения с микросхемой SD



Рисунок 19 — Проверка устройства на стенде, нет соединения с микросхемой PCF8583

По нажатию PC7 все данные с SD карты отправляются по UART.

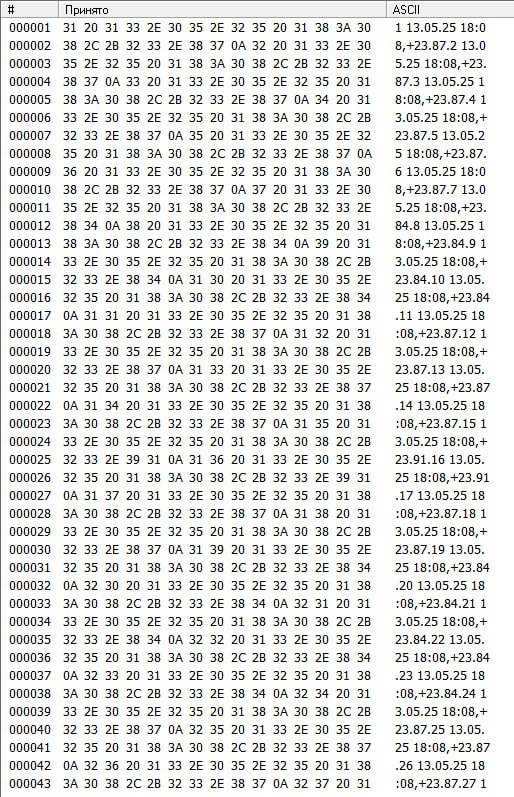


Рисунок 20 — Прием данных на ПК

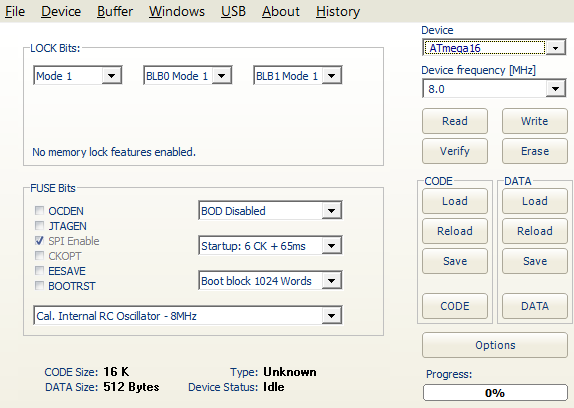


Рисунок 21 — Настройки программатора

**Приложение А**

1) схема электрическая структурная (рисунок А.1),

2) схема электрическая принципиальная (рисунок А.2),

3) перечень элементов (рисунки А.3, А.4),

4) схема алгоритма (рисунки А.5-А.33)

[5) листинг файла main.c (страницы 64 - 72),](#FileEntry2)

[6) листинг файла lcd.h (страница 73),](#FileEntry3)

[7) листинг файла lcd.c (страницы 74 - 76),](#FileEntry4)

[8) листинг файла max31865.h (страница 77),](#FileEntry5)

[9) листинг файла max31865.c (страницы 78 - 79),](#FileEntry6)

[10) листинг файла sdcard.h (страница 80),](#FileEntry7)

[11) листинг файла sdcard.c (страницы 81 - 84),](#FileEntry8)

[12) листинг файла twi.h (страница 85),](#FileEntry9)

[13) листинг файла twi.c (страницы 86 - 89),](#FileEntry10)

[14) листинг файла sevseg.h (страница 90),](#FileEntry11)

[15) листинг файла secseg.c (страницы 91 - 93),](#FileEntry12)

[16) листинг файла spi.h (страница 94),](#FileEntry13)

[17) листинг файла spi.c (страница 95)](#FileEntry14).

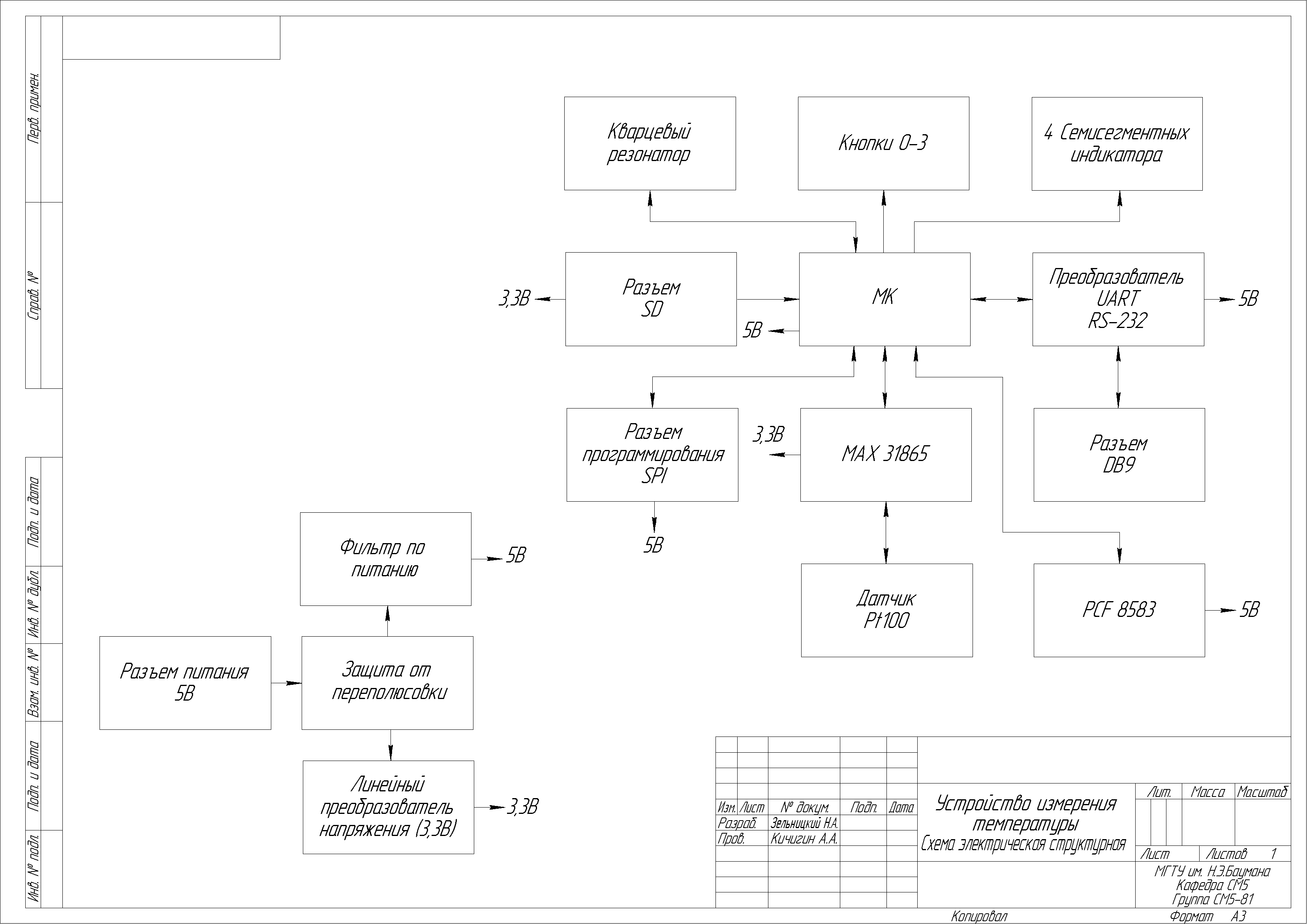


Рисунок А. — Схема электрическая структурная

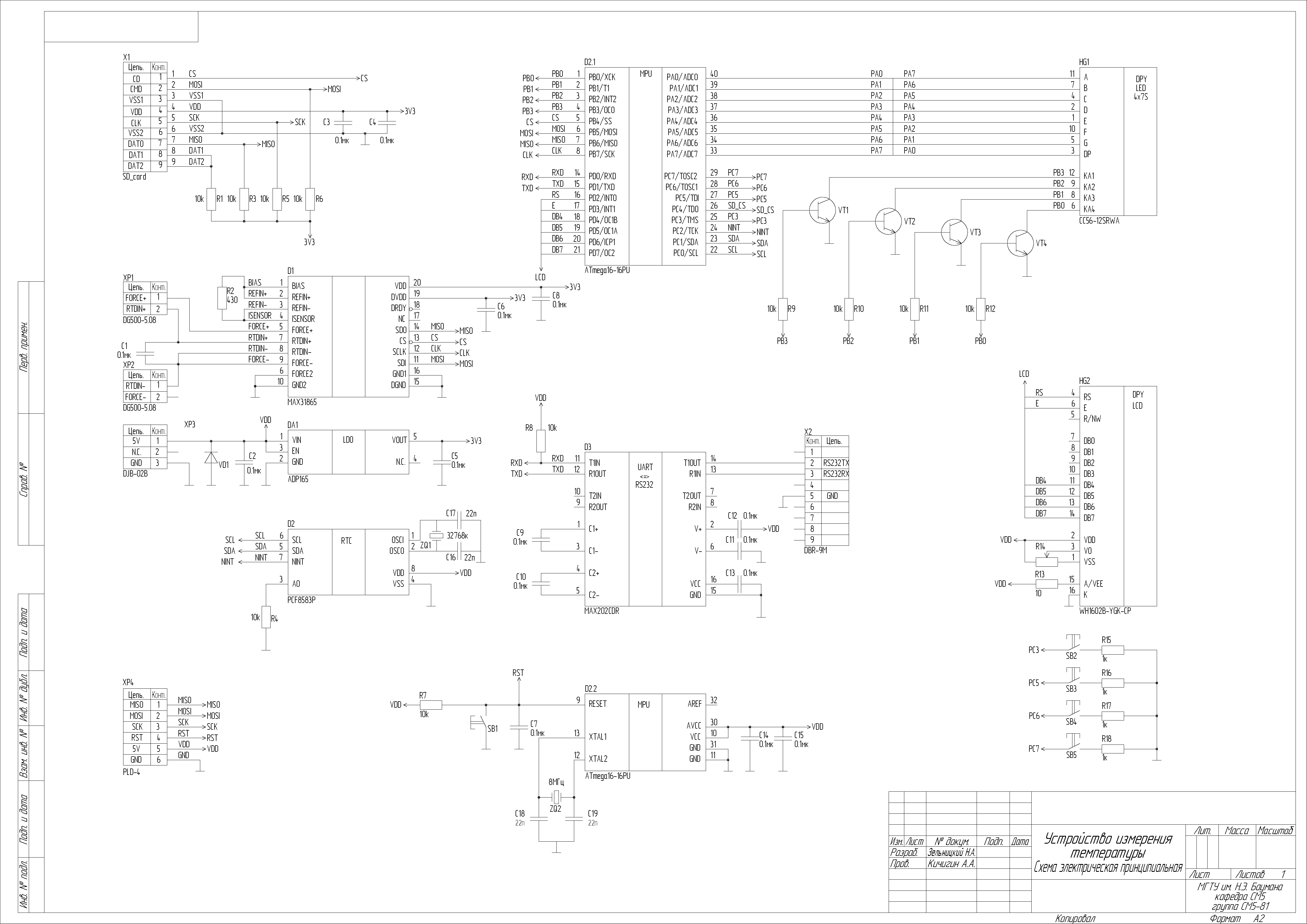


Рисунок А.2 — Схема электрическая принципиальная

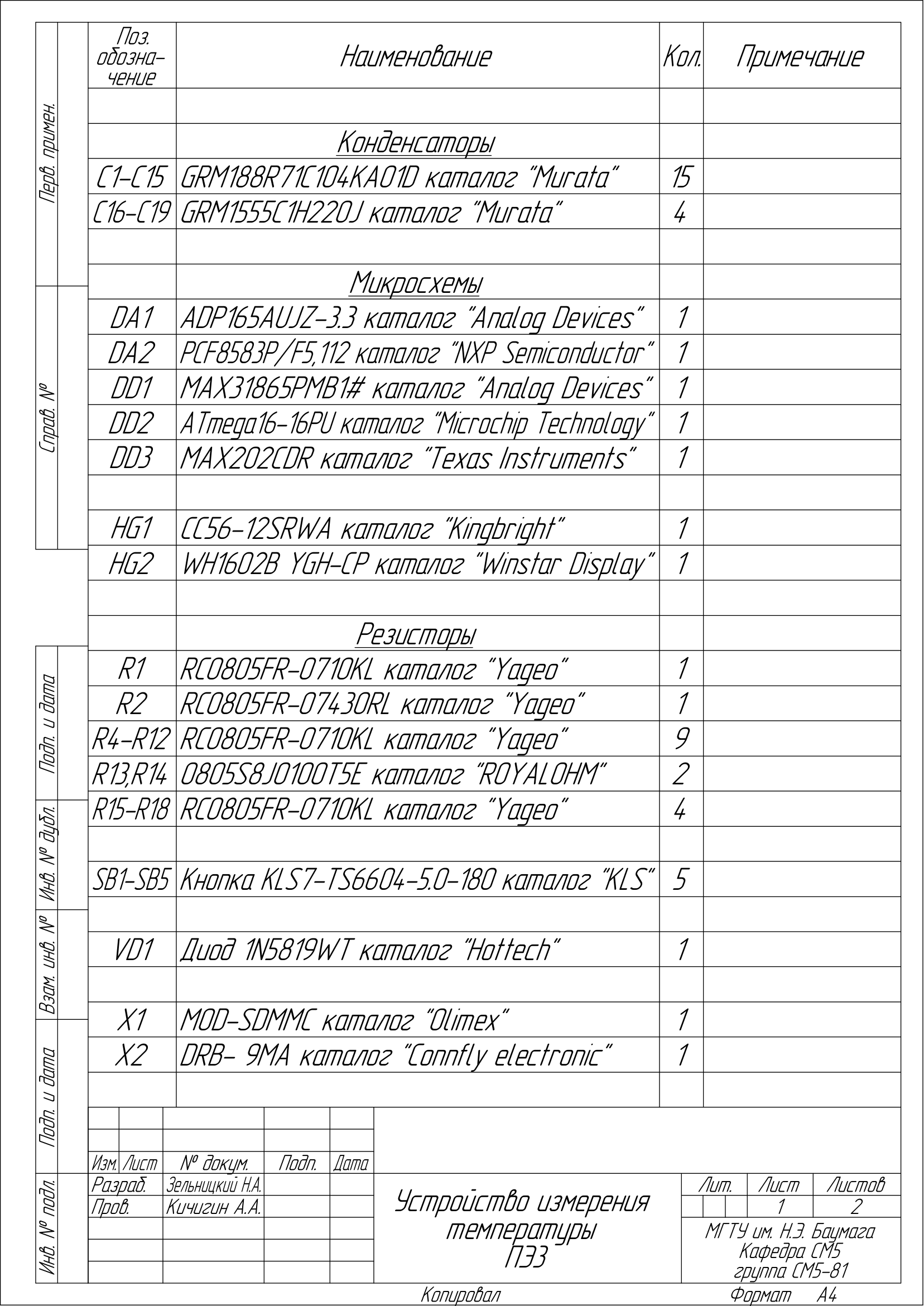


Рисунок А. — Перечень элементов, лист 1

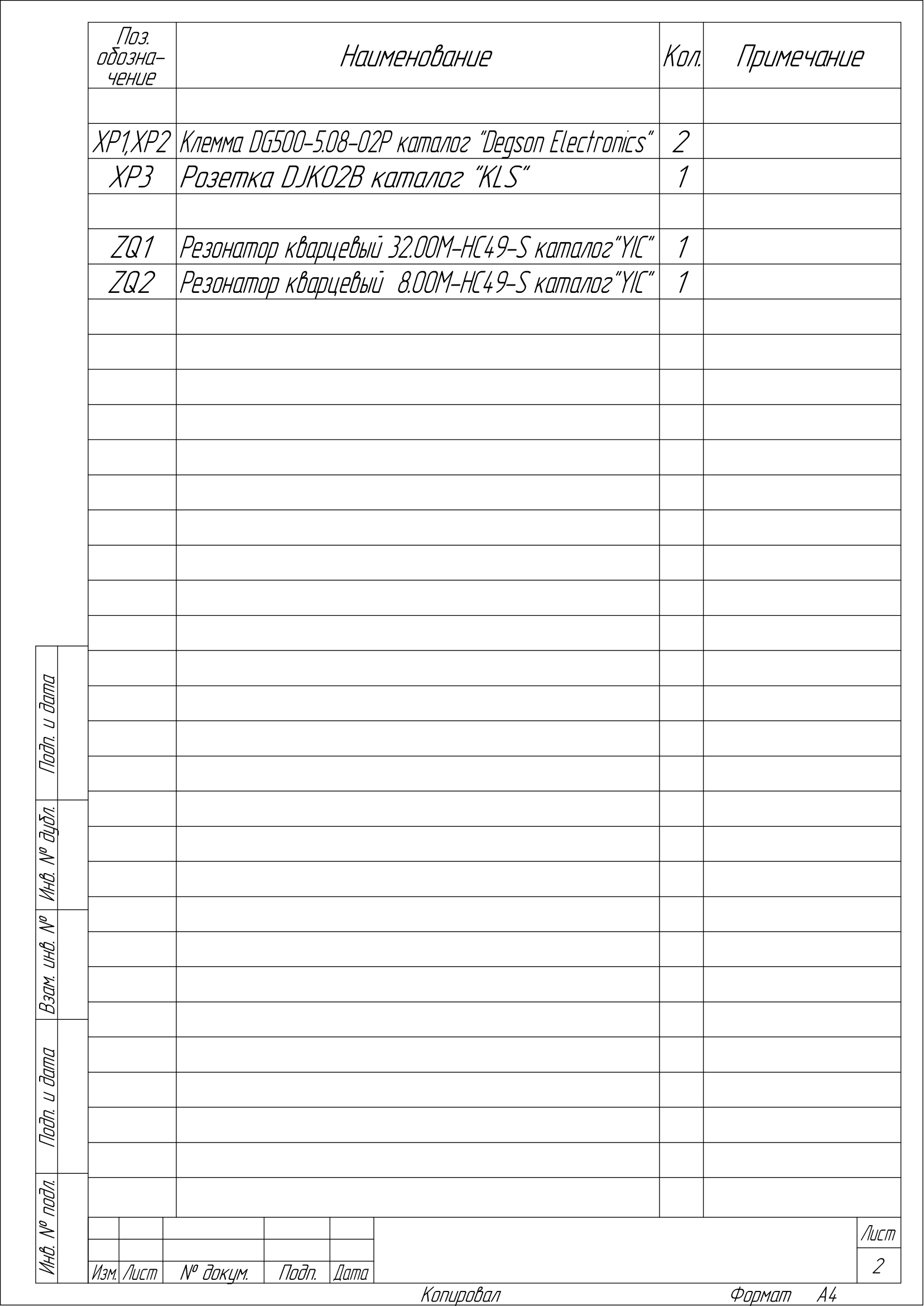


Рисунок А. 4 — Перечень элементов, лист 2

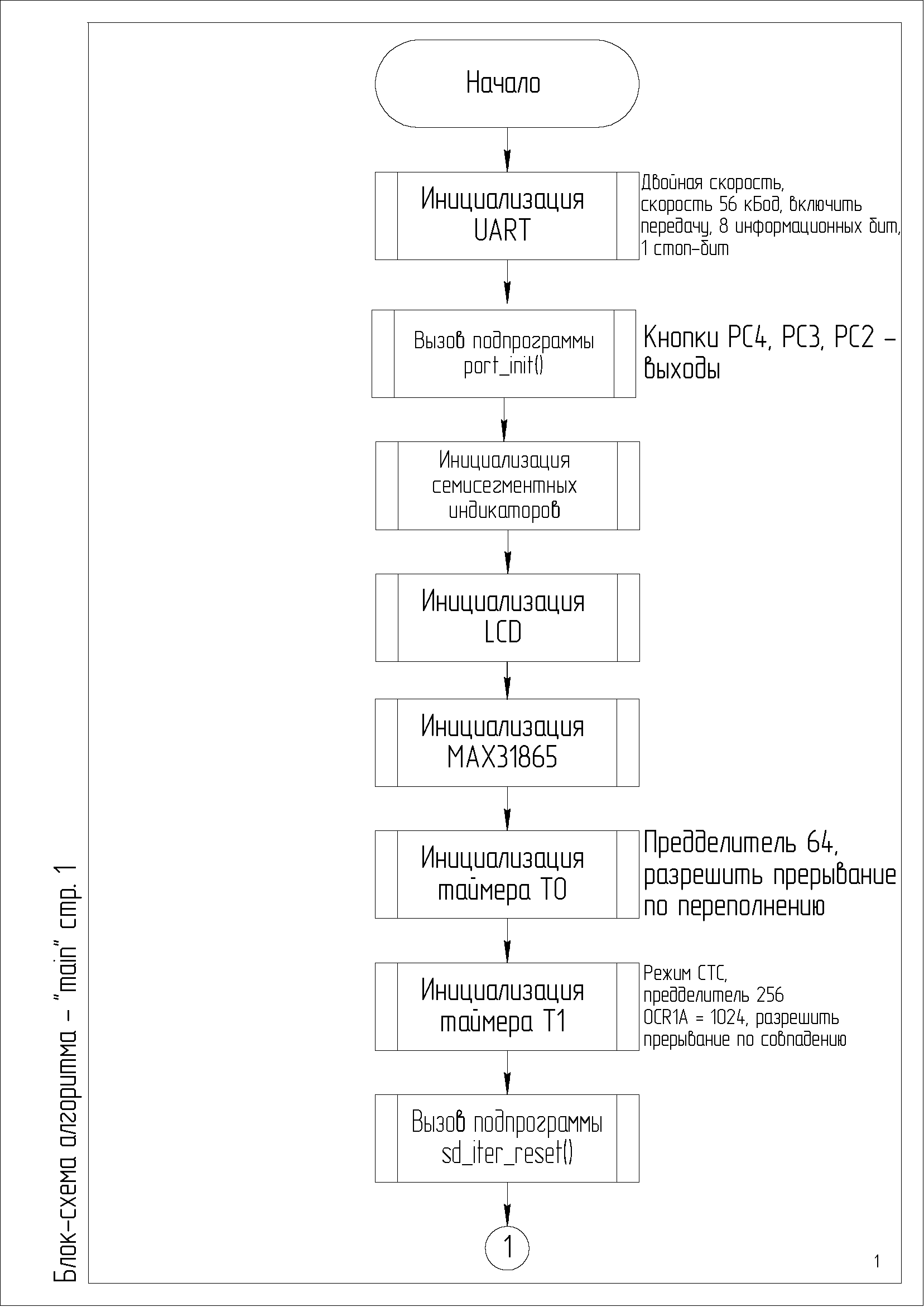


Рисунок А. 5 — Блок схема алгоритма, лист 1



Рисунок А. 6 — Блок схема алгоритма, лист 2

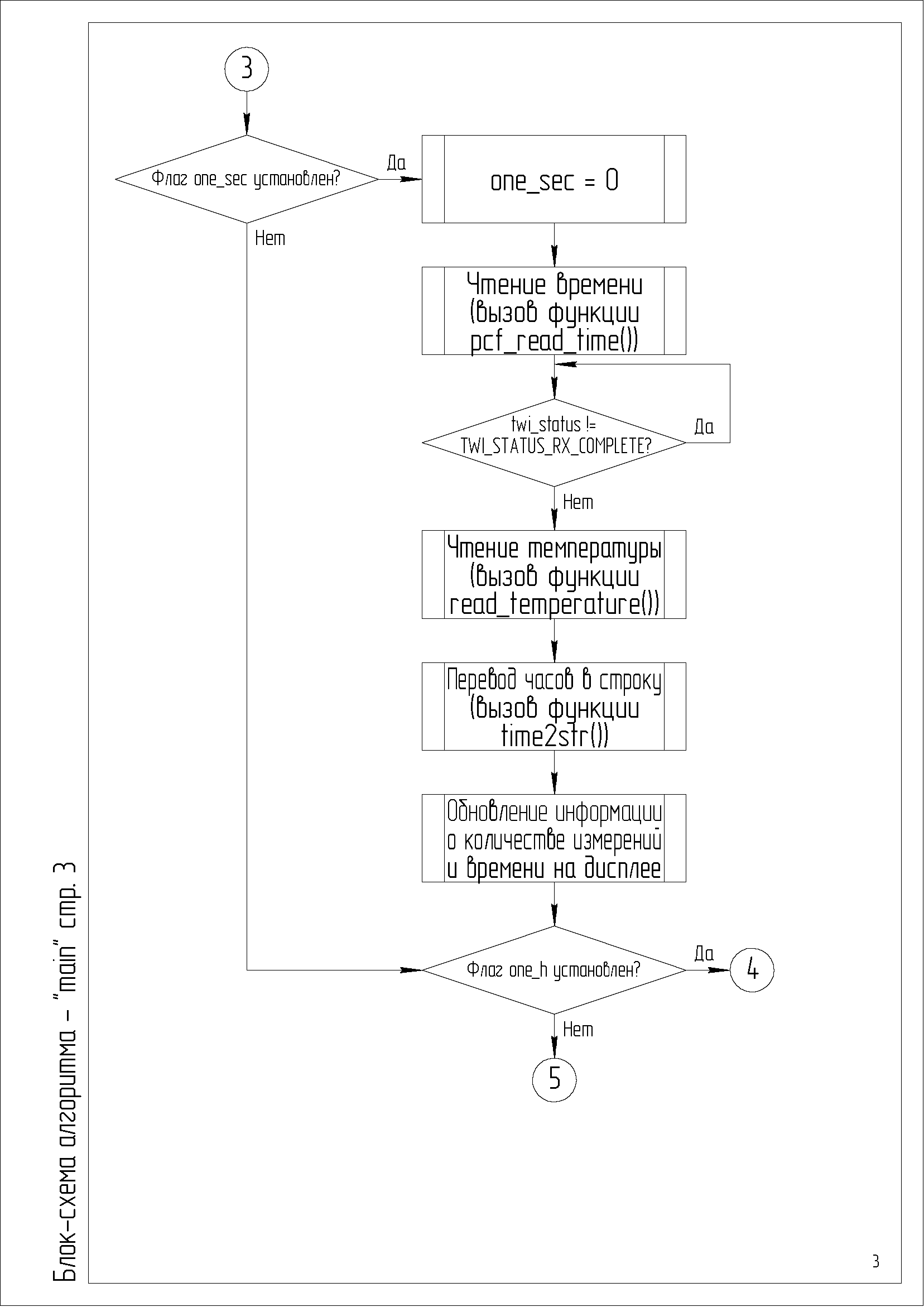
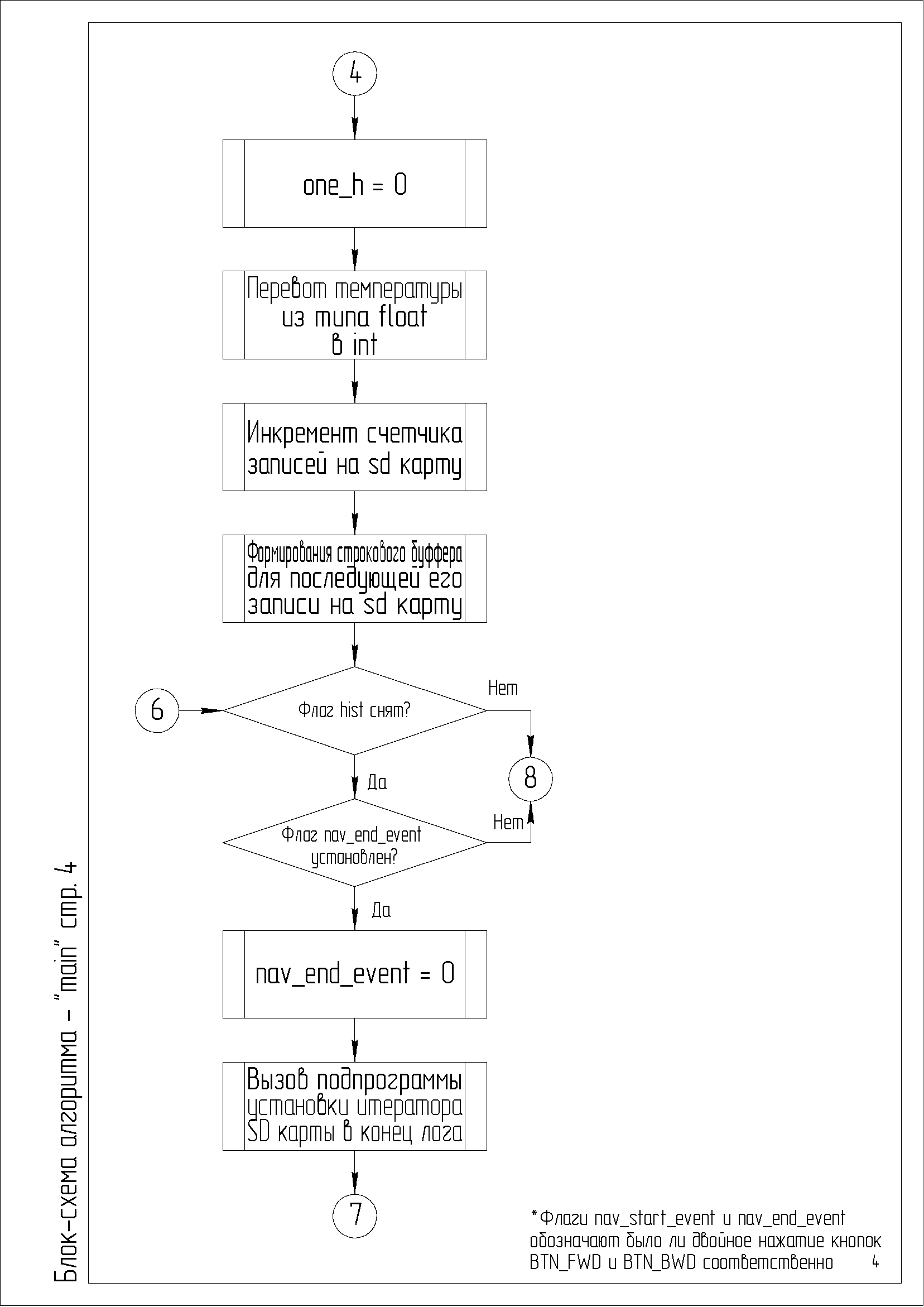
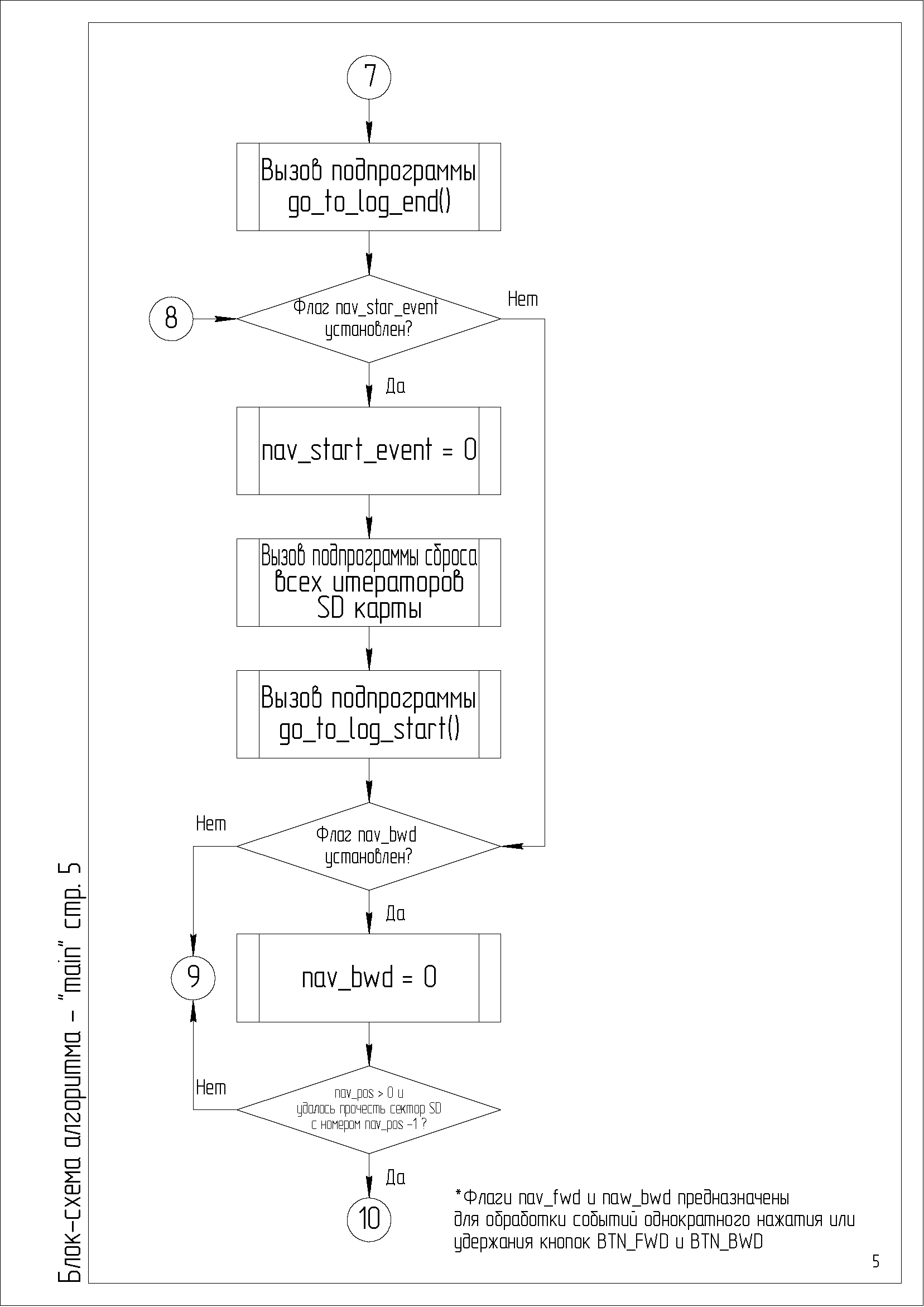


Рисунок А. 7 — Блок схема алгоритма, лист 3

 Рисунок А. 8 — Блок схема алгоритма, лист 4

 Рисунок А. 9 — Блок схема алгоритма, лист 5

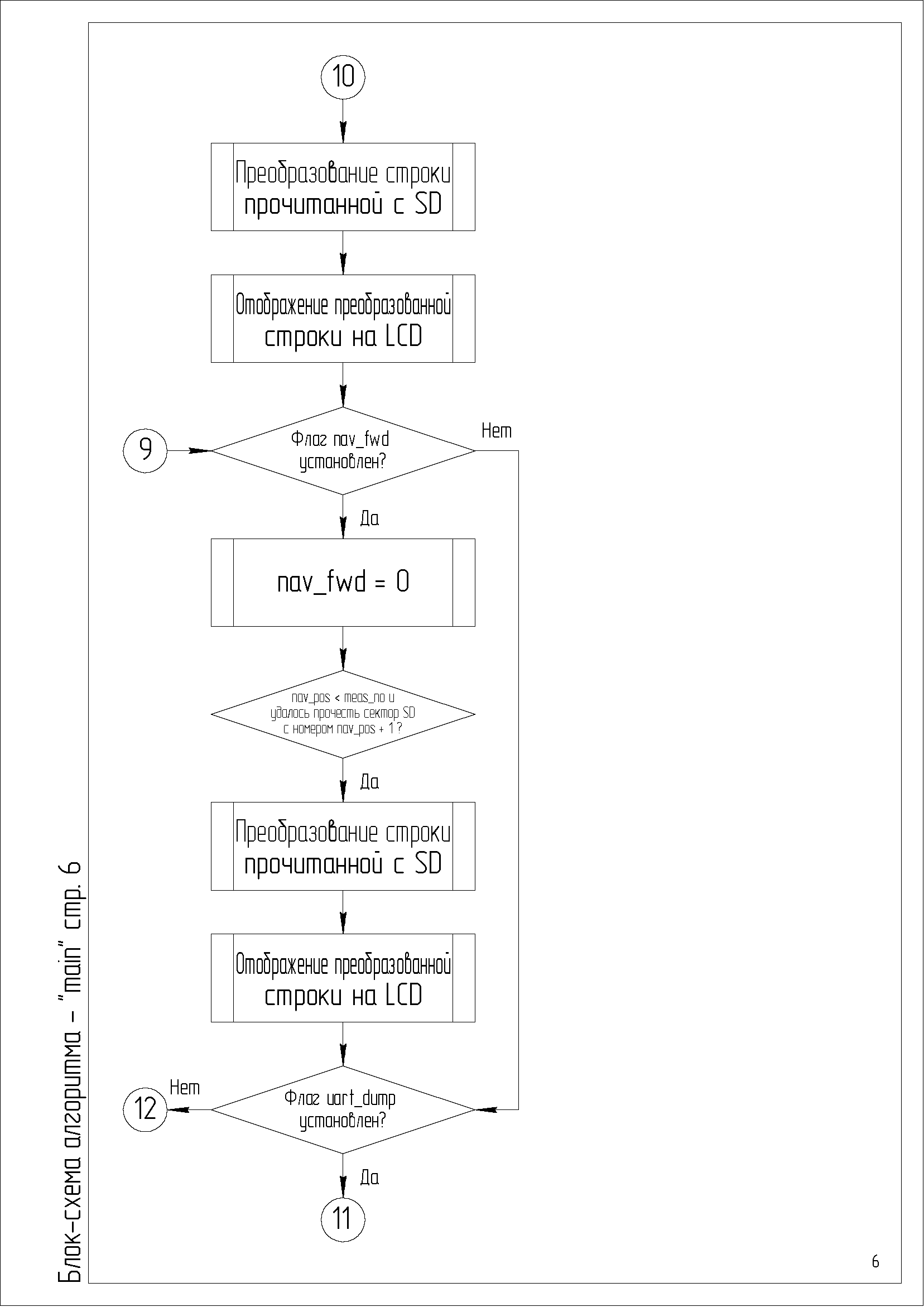


Рисунок А. 10 — Блок схема алгоритма, лист 6

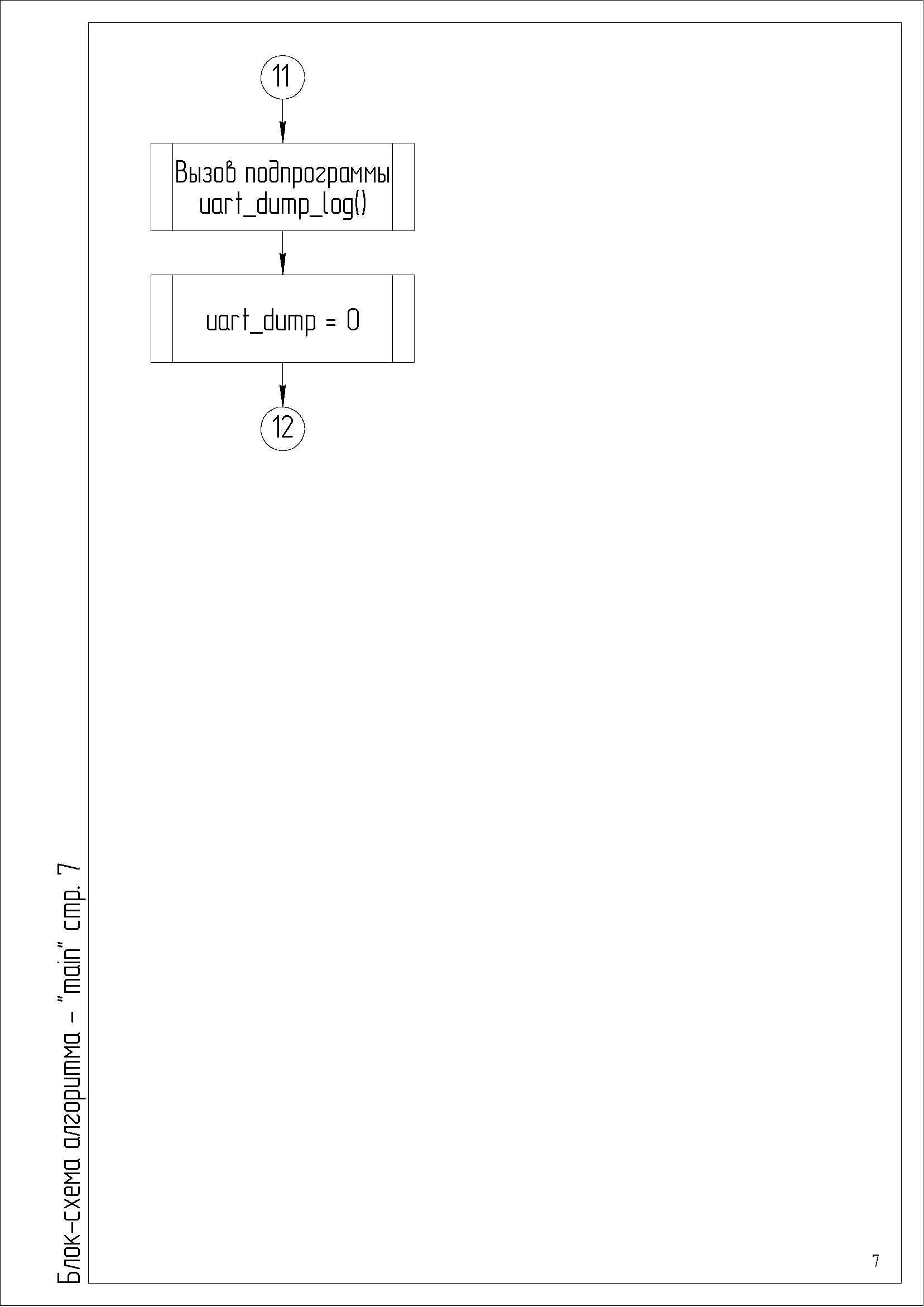


Рисунок А. 11 — Блок схема алгоритма, лист 7



Рисунок А. 12 — Блок схема алгоритма, лист 8

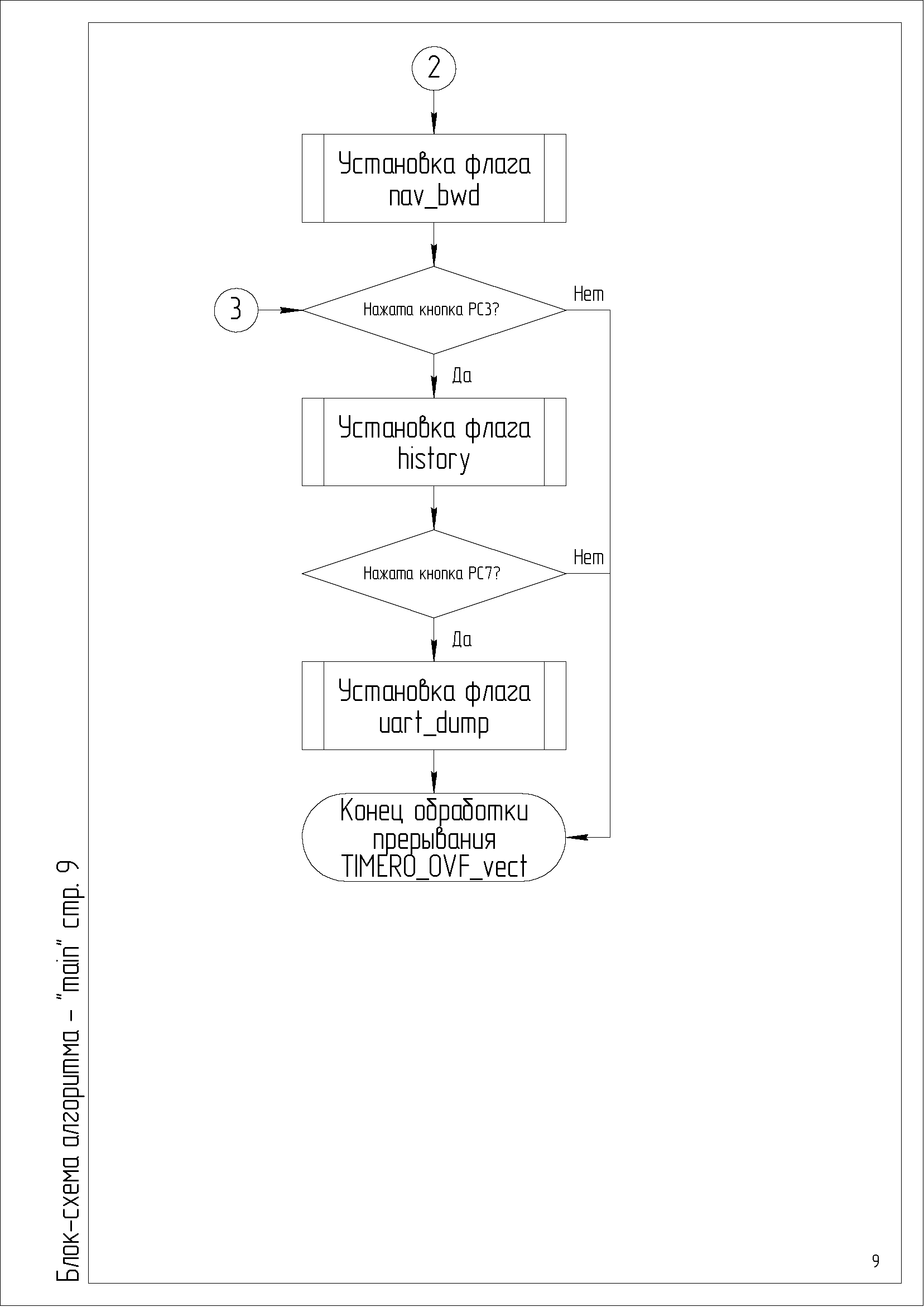


Рисунок А. 13 — Блок схема алгоритма, лист 9

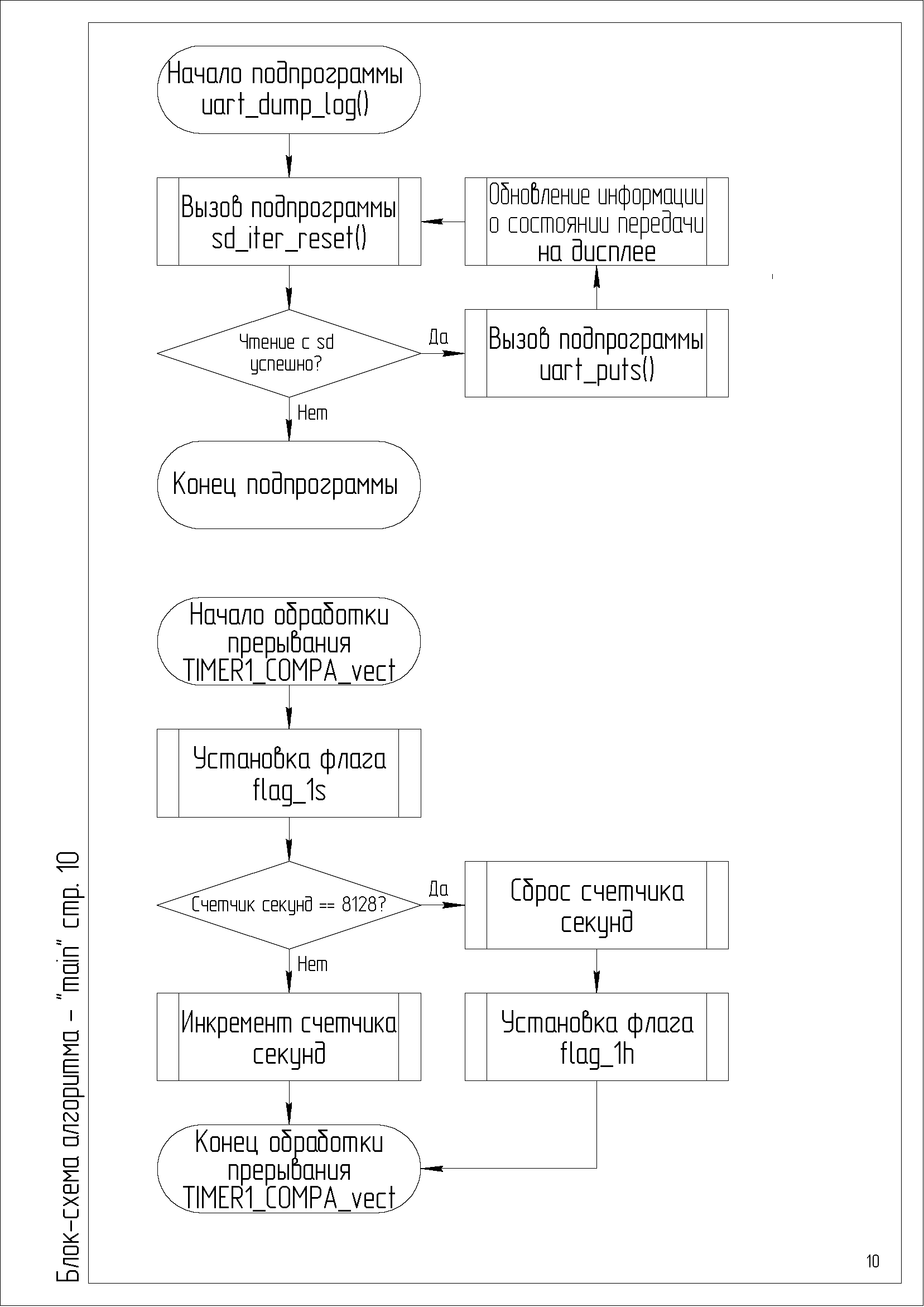


Рисунок А. 14 — Блок схема алгоритма, лист 10

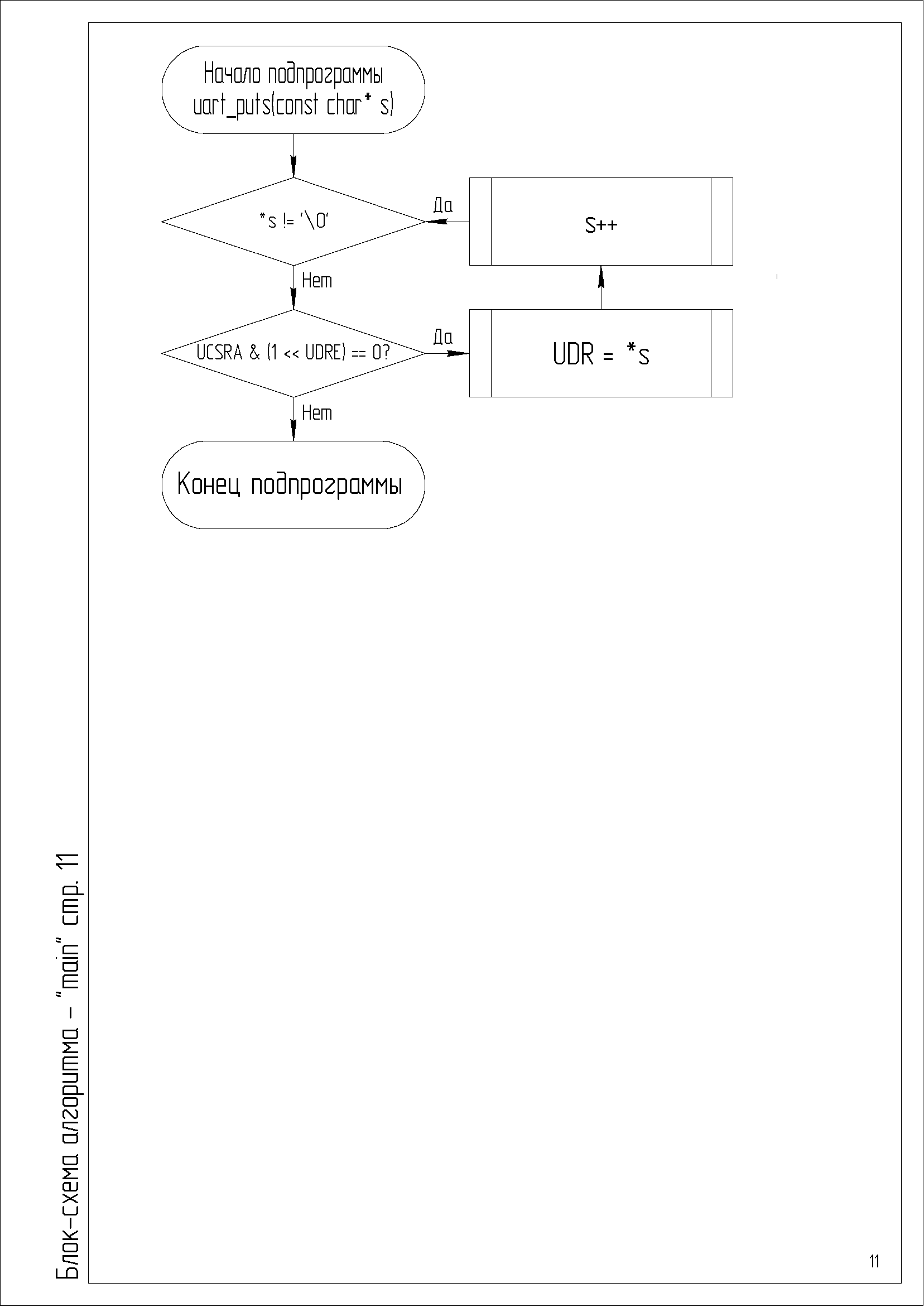


Рисунок А. 15 — Блок схема алгоритма, лист 11

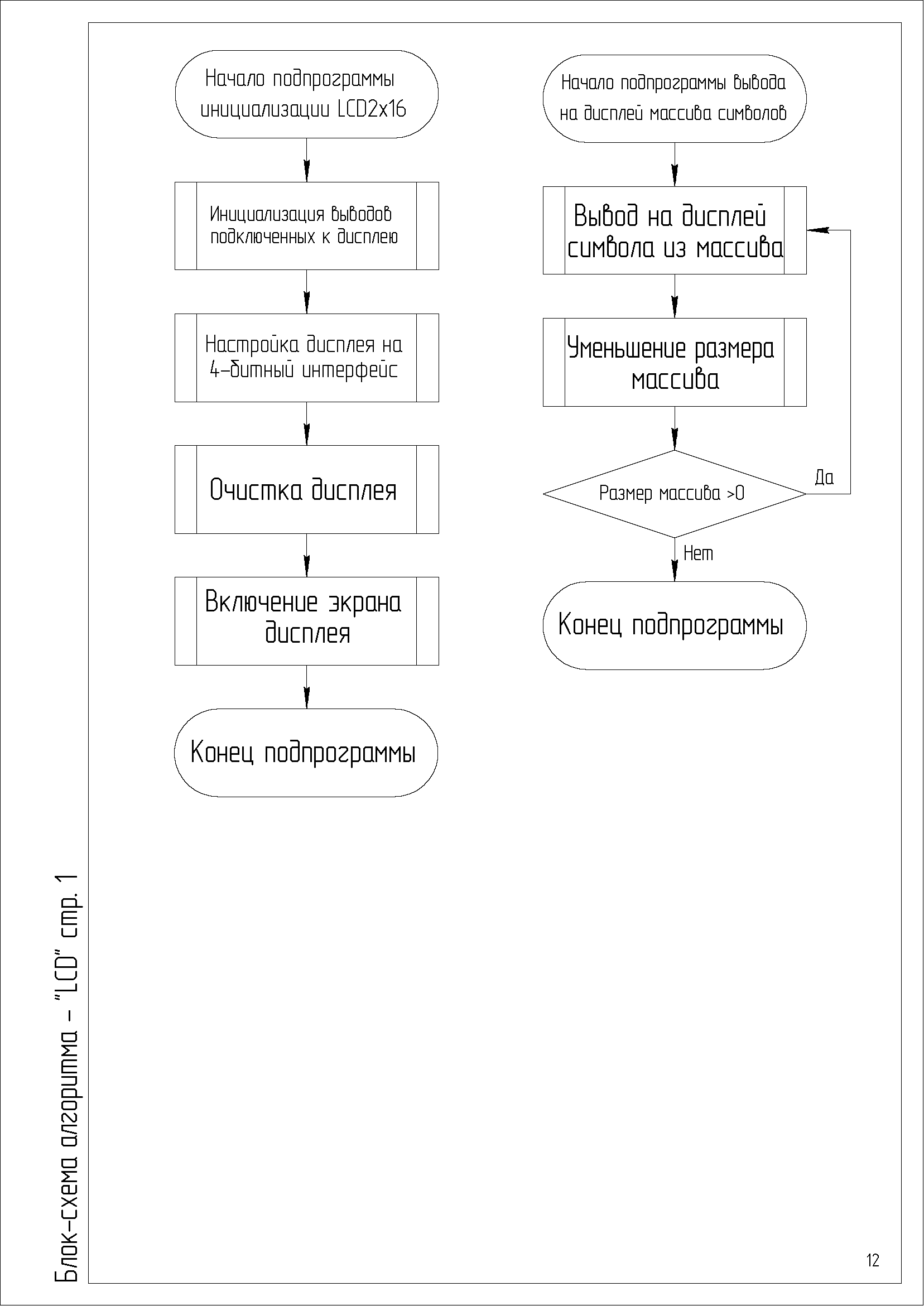


Рисунок А. 16 — Блок схема алгоритма, лист 12

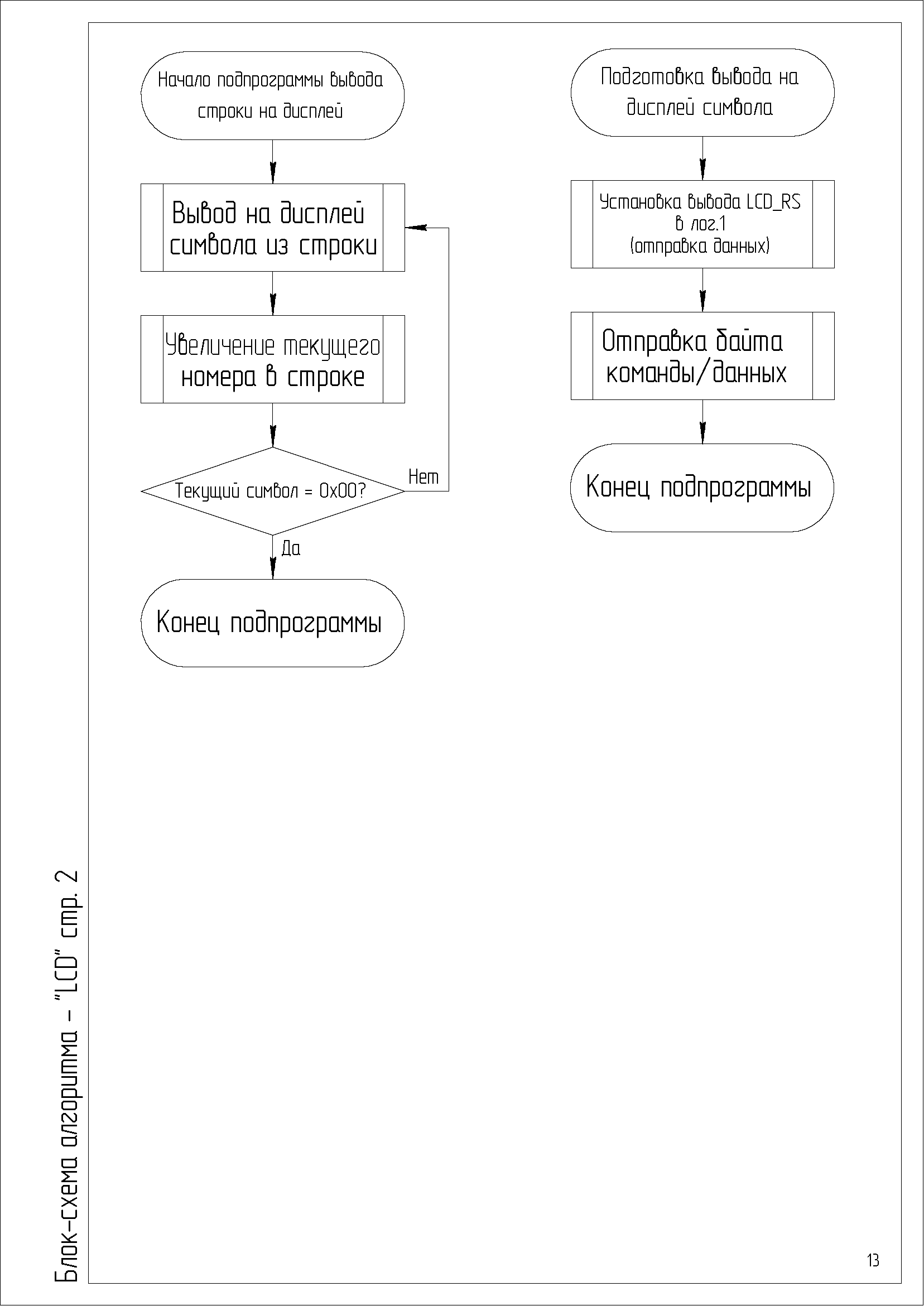


Рисунок А. 17 — Блок схема алгоритма, лист 13

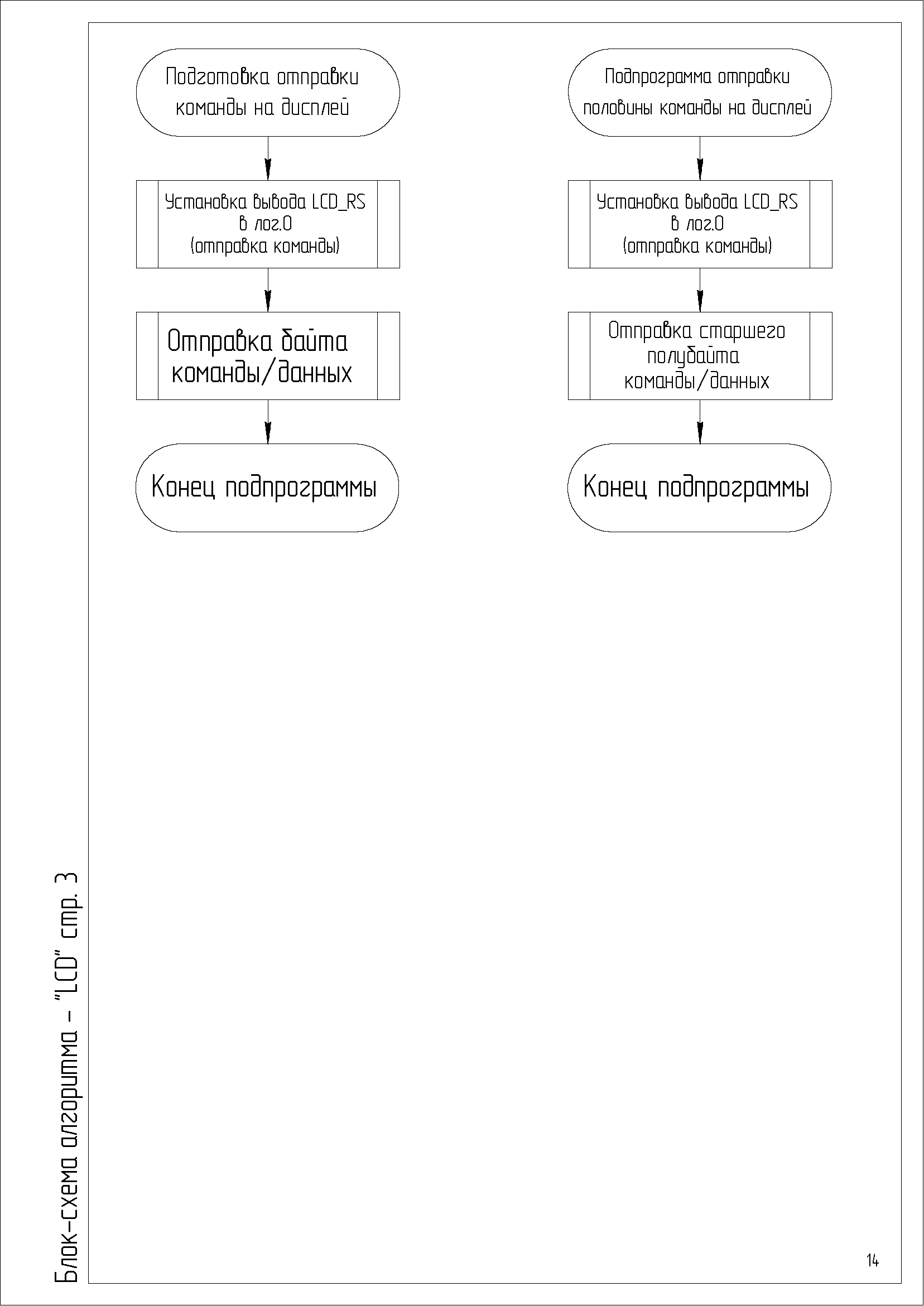


Рисунок А. 18 — Блок схема алгоритма, лист 14

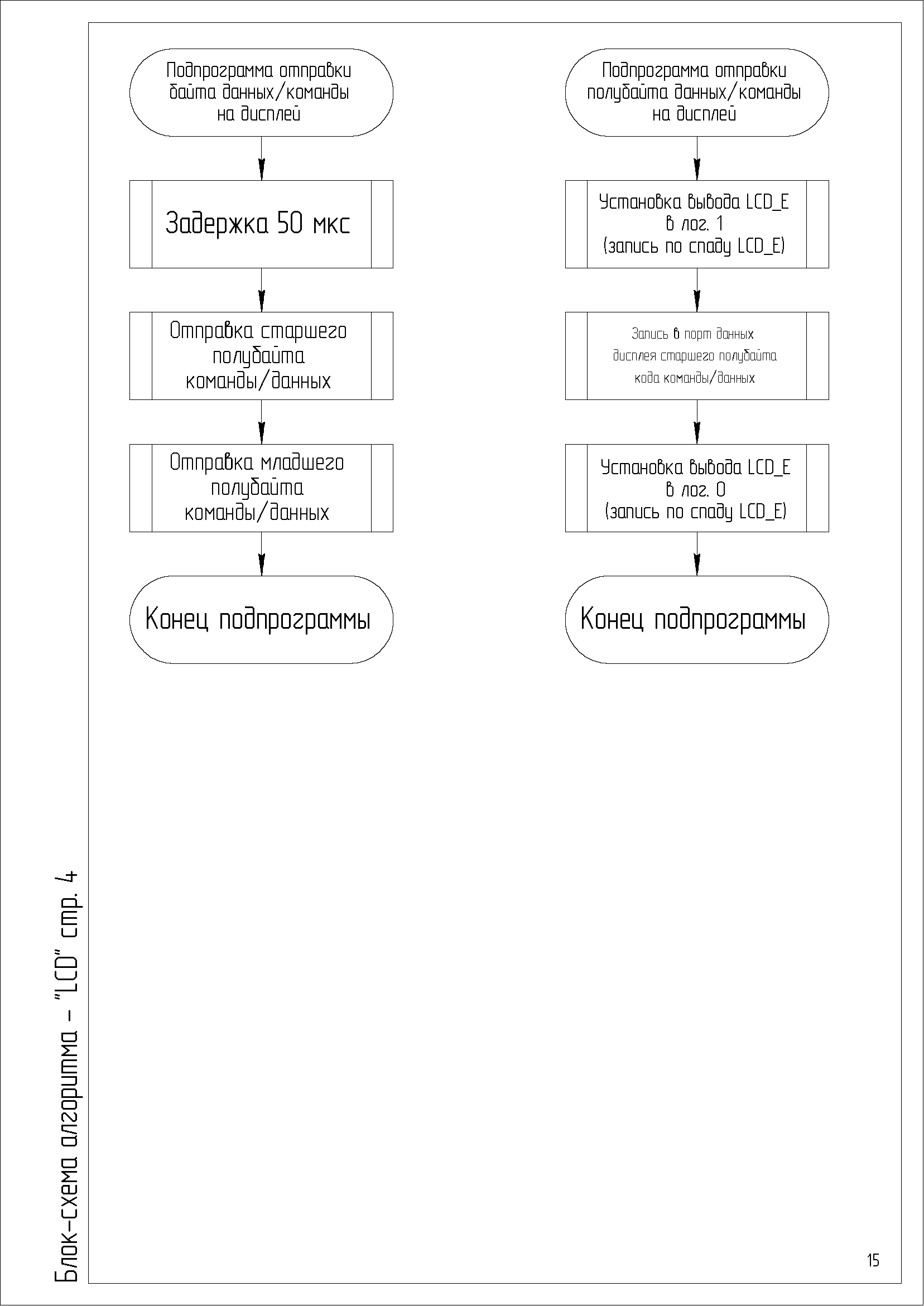


Рисунок А. 19 — Блок схема алгоритма, лист 15

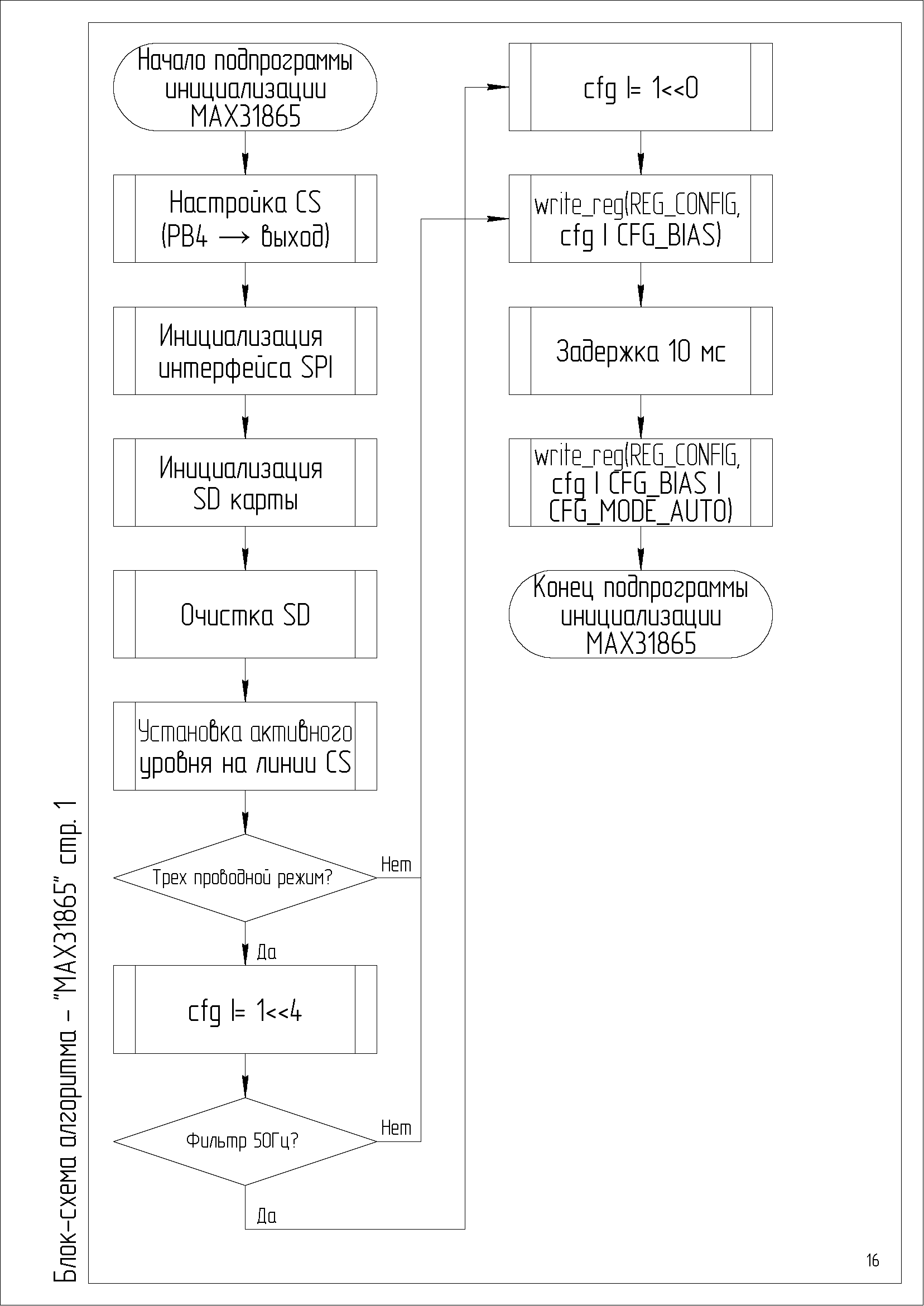


Рисунок А. 20 — Блок схема алгоритма, лист 16

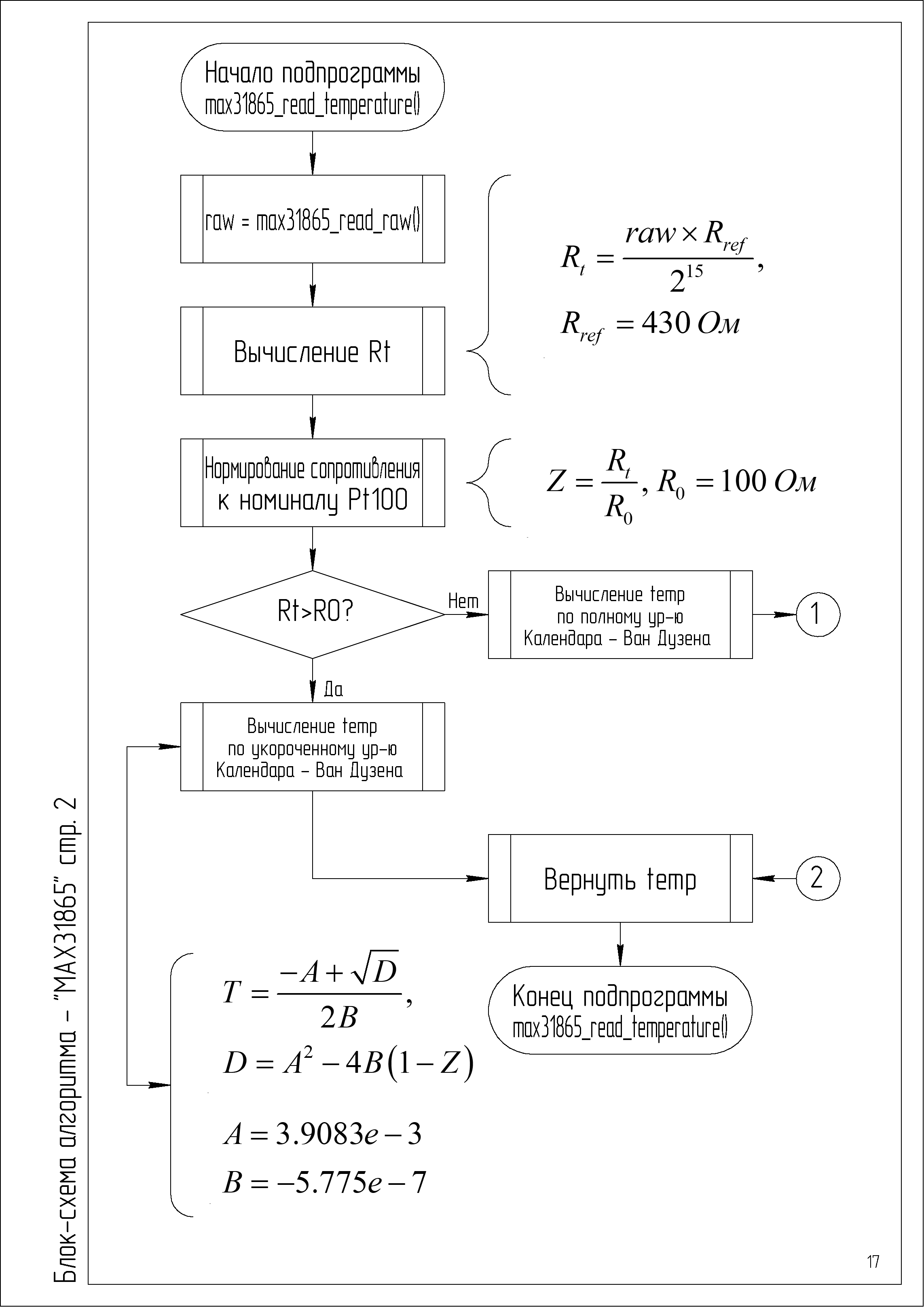


Рисунок А. 21 — Блок схема алгоритма, лист 17

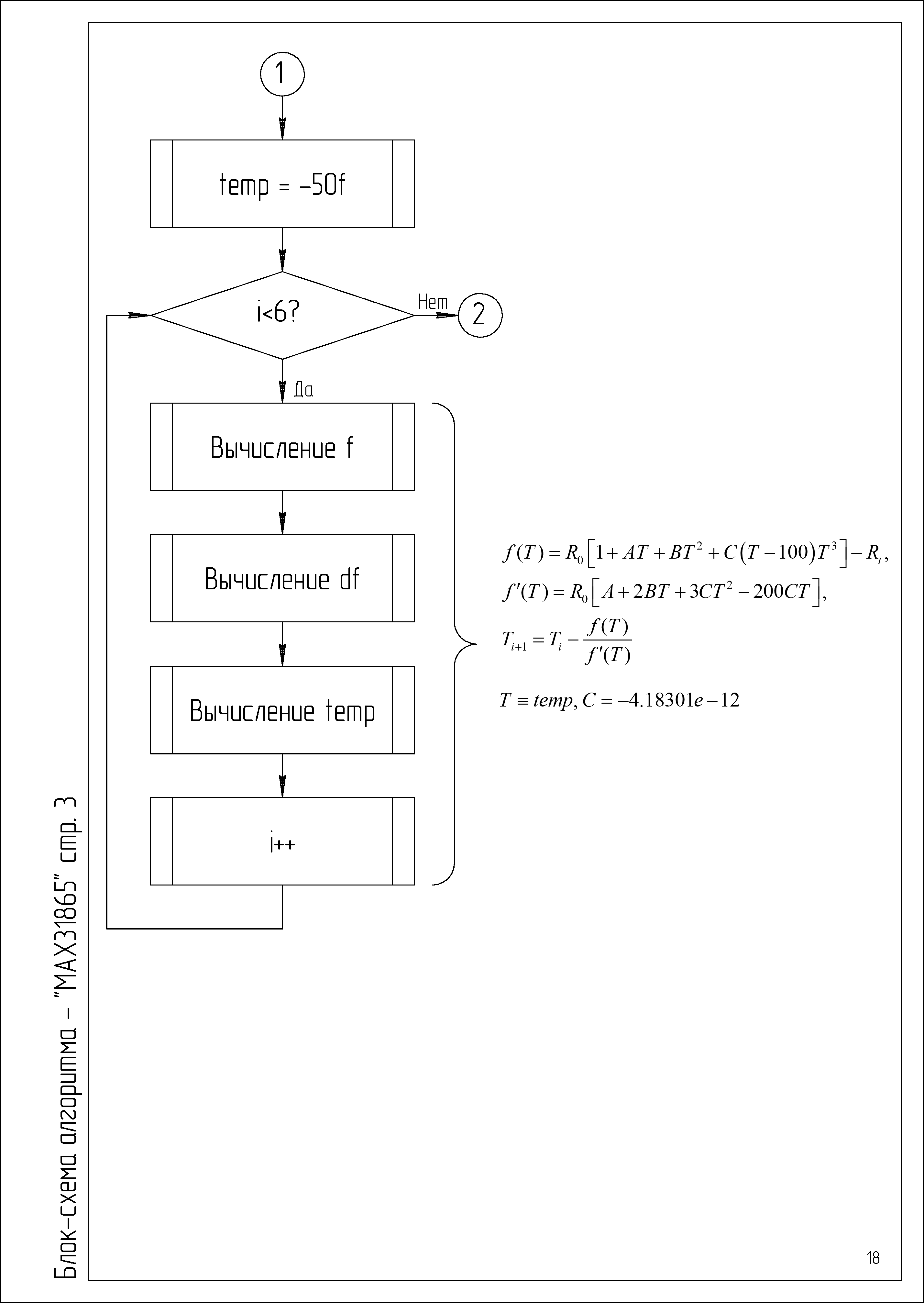


Рисунок А. 22 — Блок схема алгоритма, лист 18

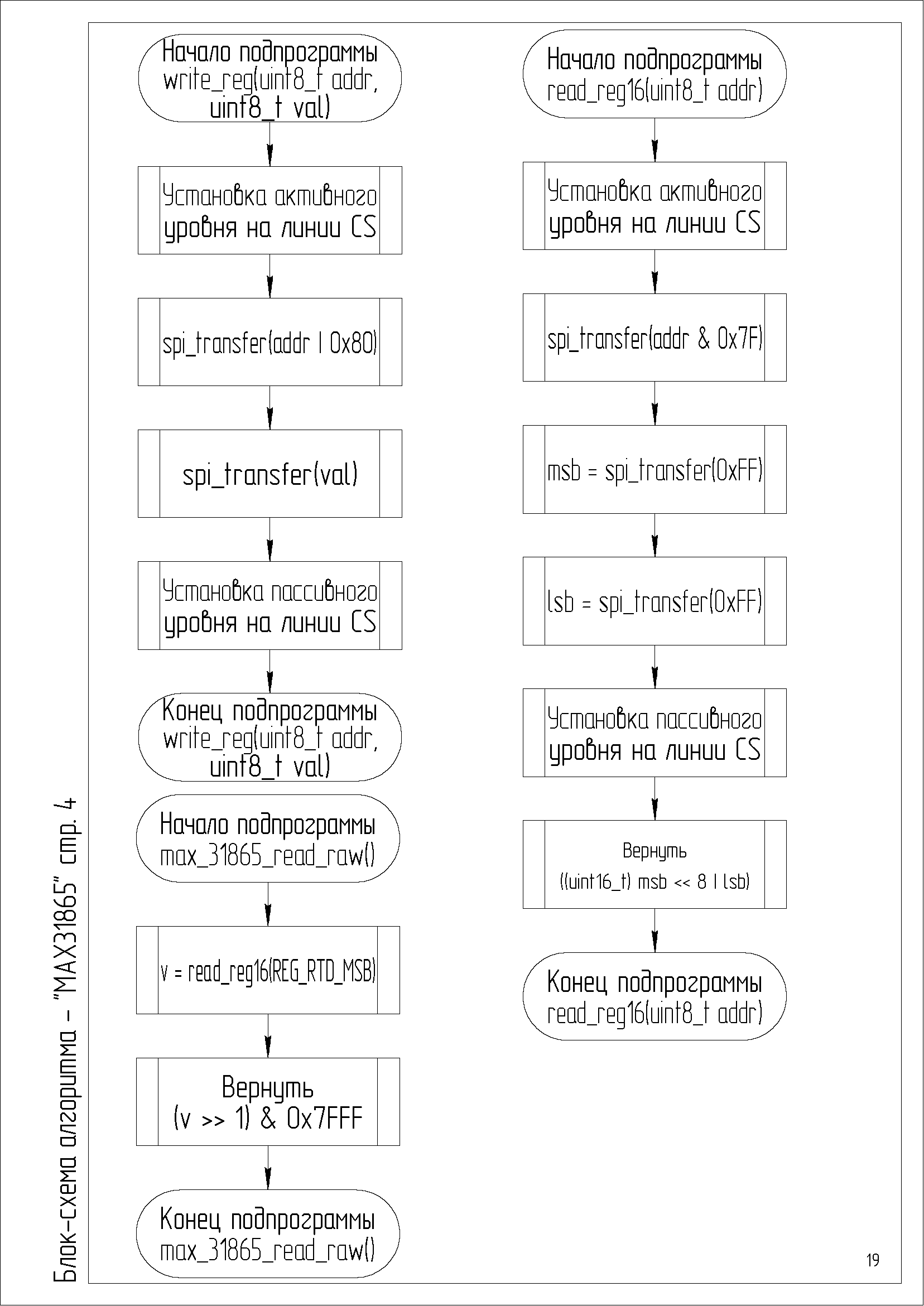


Рисунок А. 23 — Блок схема алгоритма, лист 19

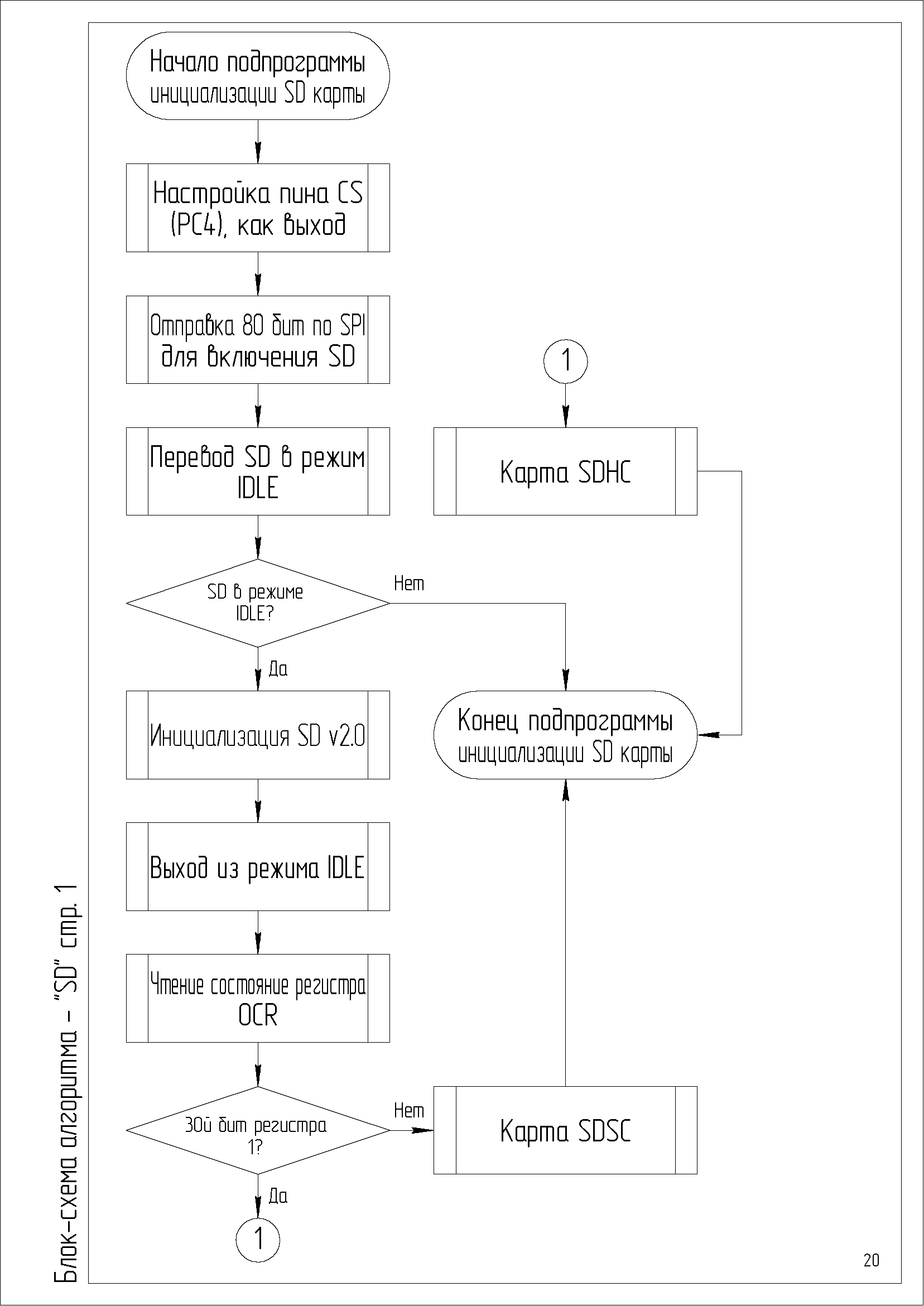


Рисунок А. 24 — Блок схема алгоритма, лист 20

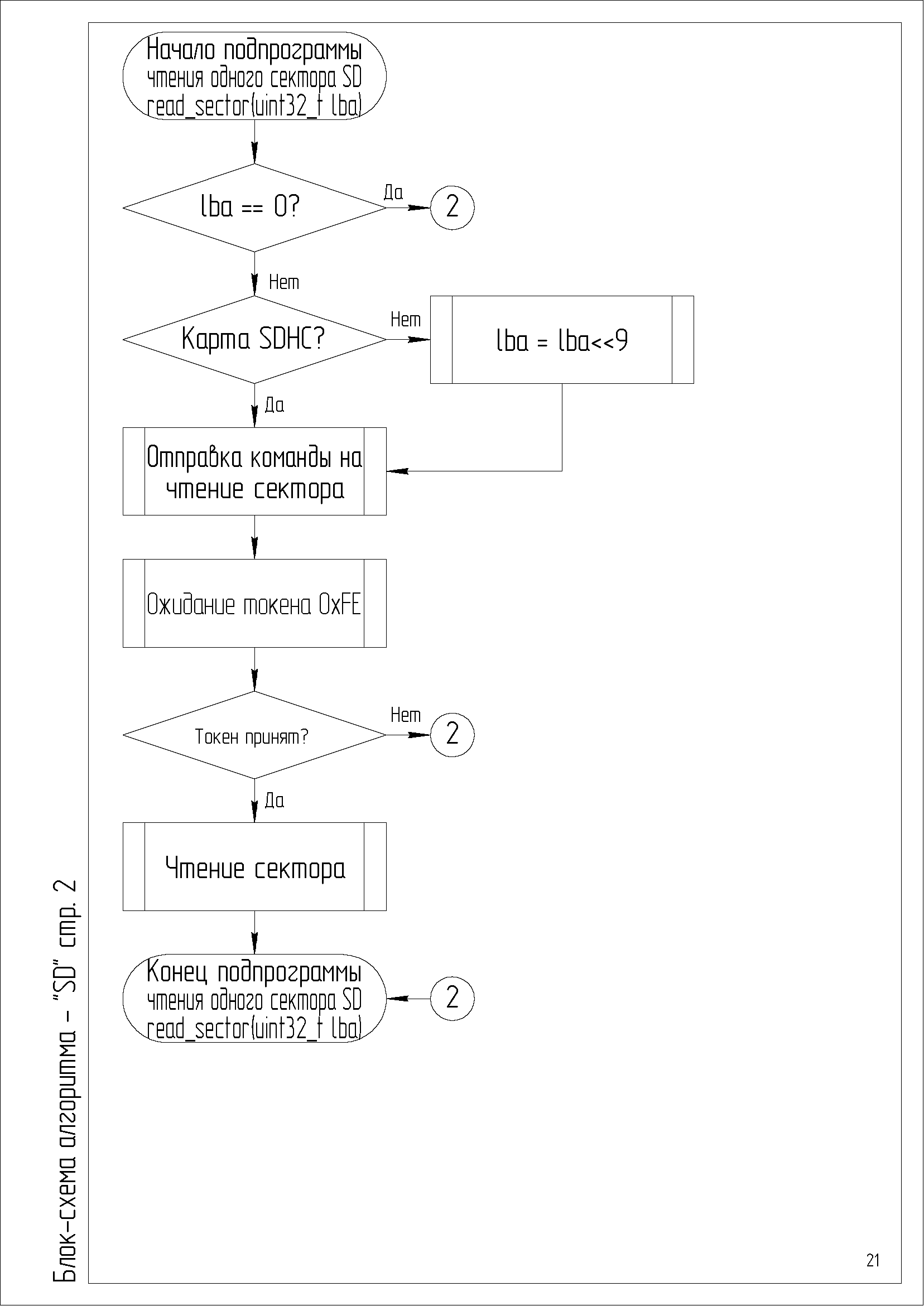


Рисунок А. 25 — Блок схема алгоритма, лист 21

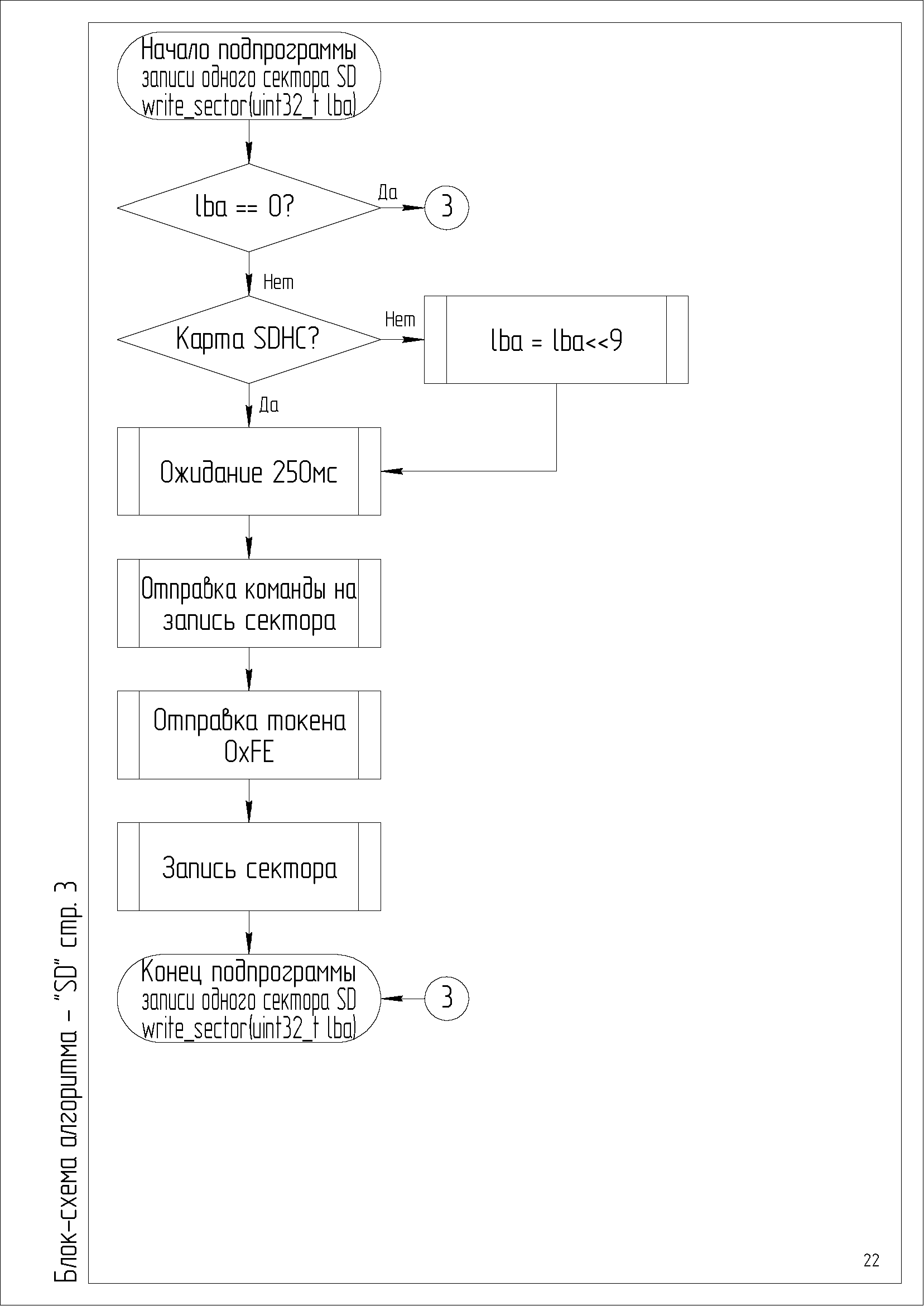


Рисунок А. 26 — Блок схема алгоритма, лист 22

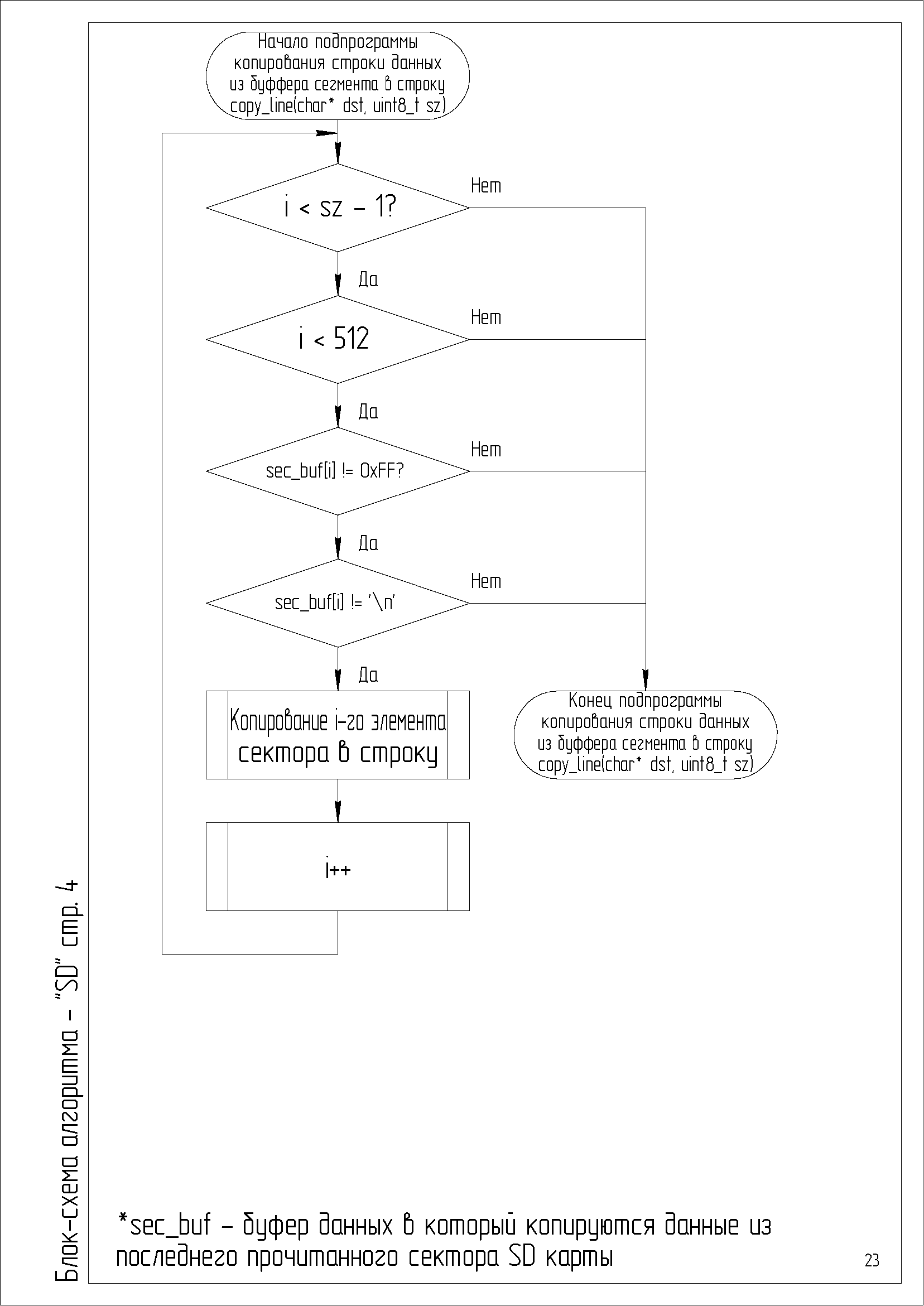


Рисунок А. 27 — Блок схема алгоритма, лист 23

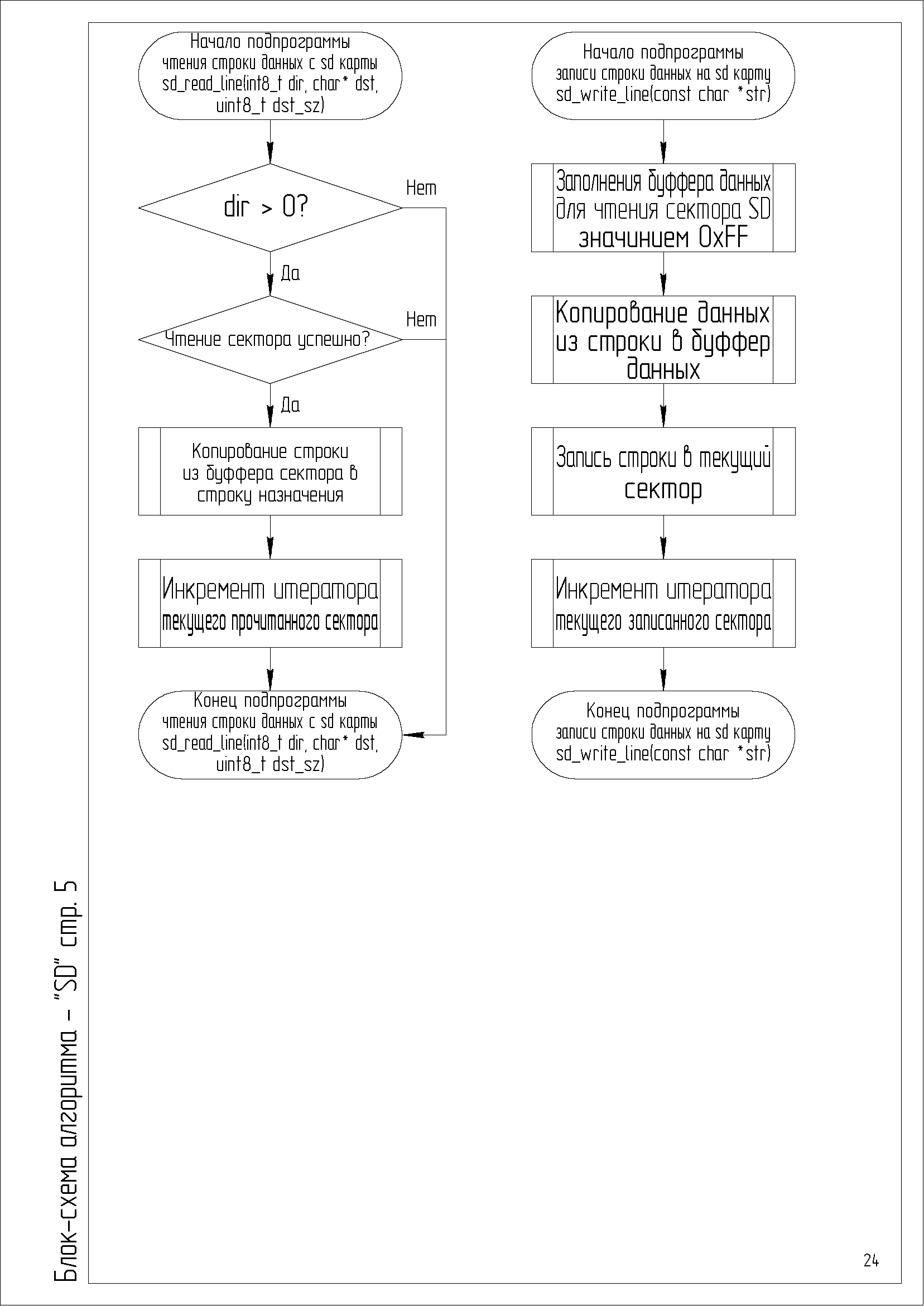


Рисунок А. 28 — Блок схема алгоритма, лист 24

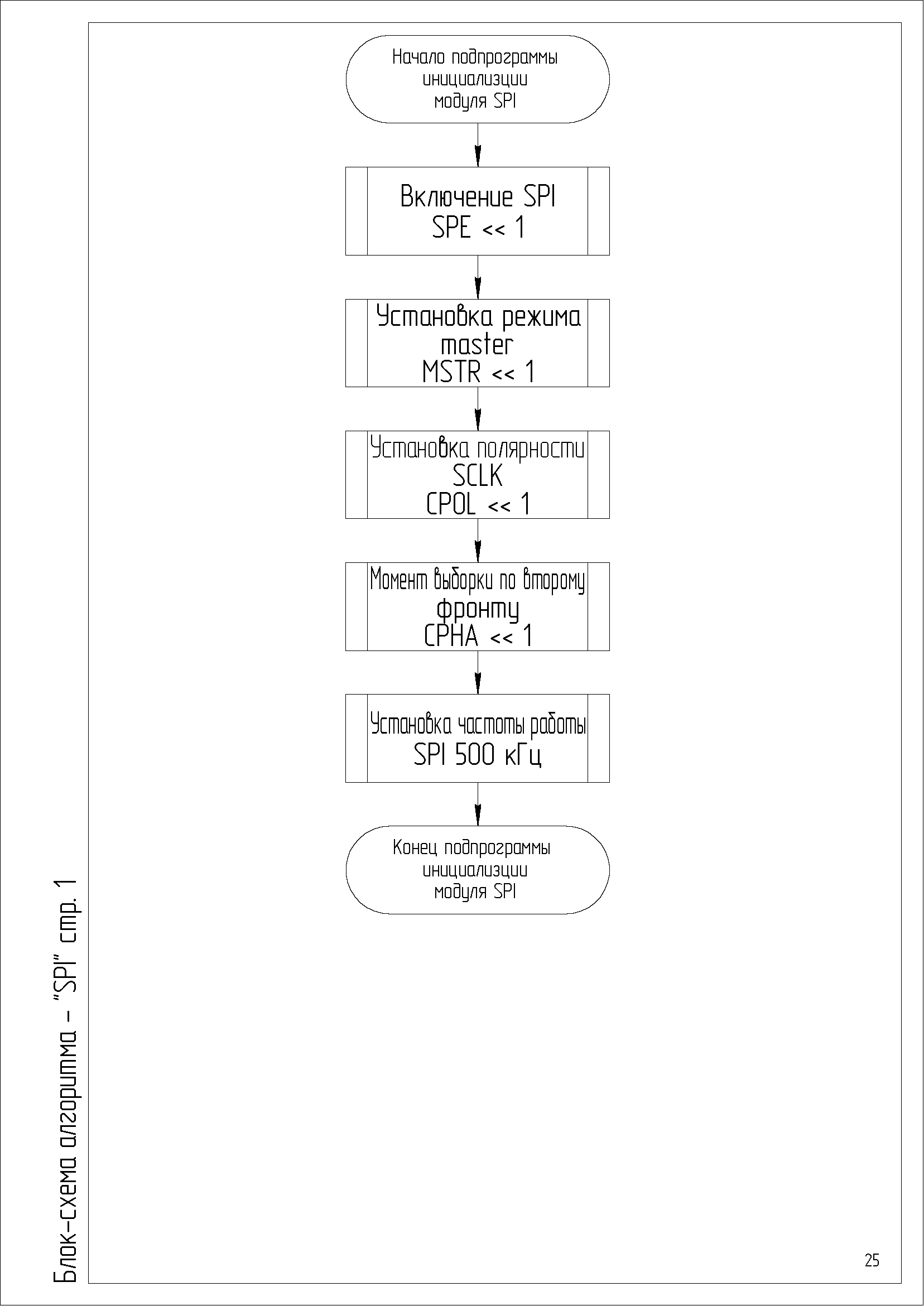


Рисунок А. 29 — Блок схема алгоритма, лист 25

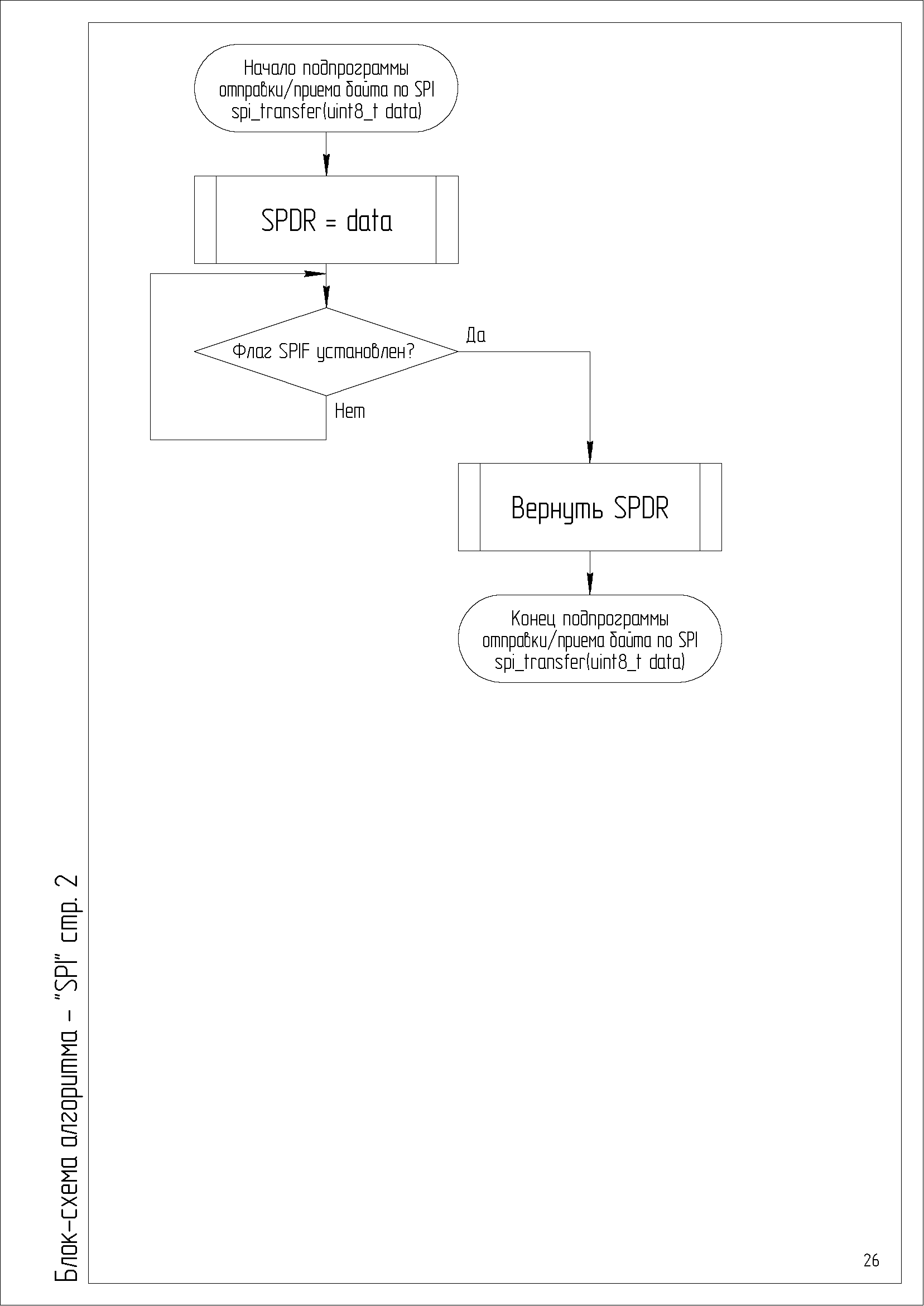


Рисунок А. 30 — Блок схема алгоритма, лист 26

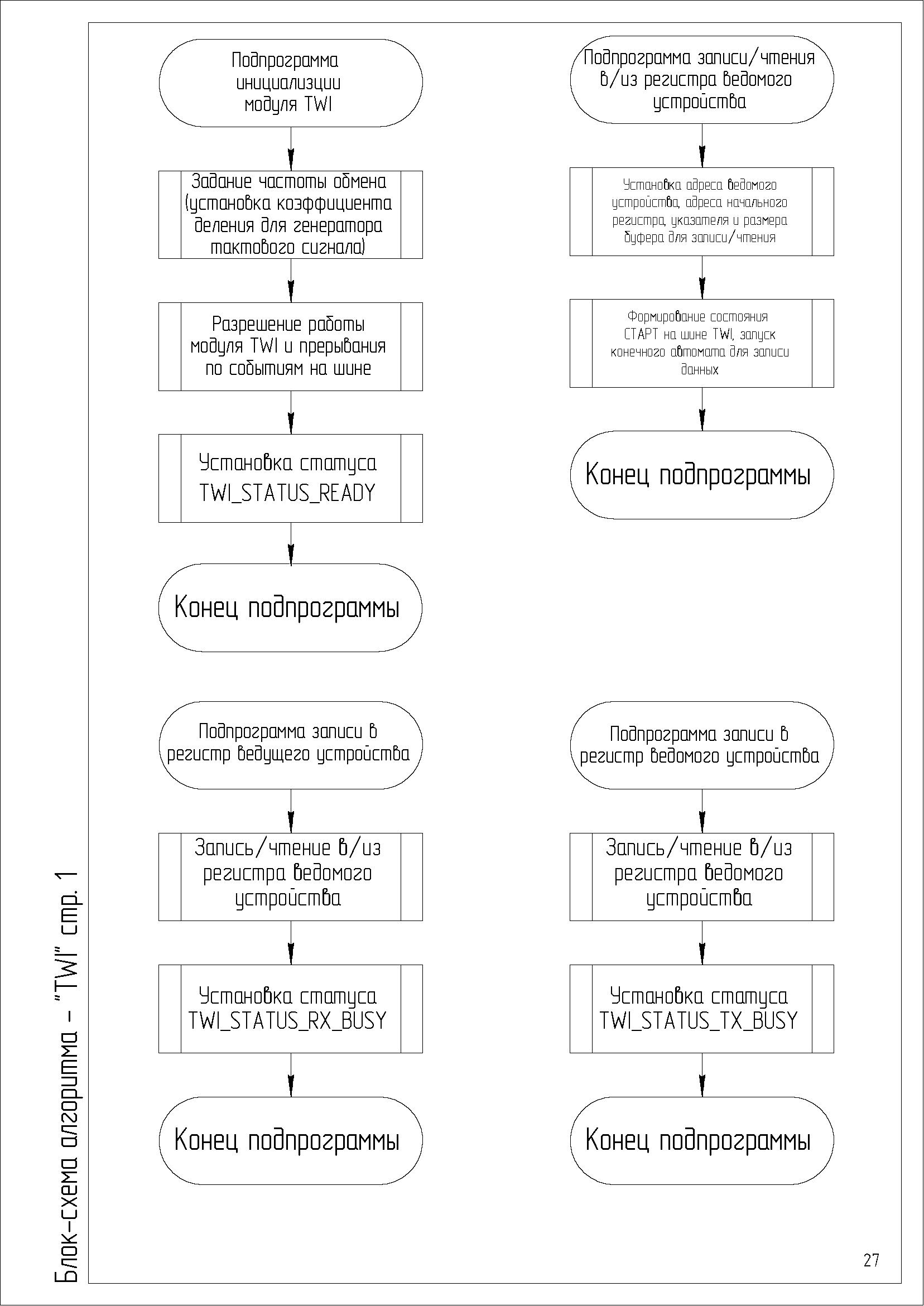


Рисунок А. 31 — Блок схема алгоритма, лист 27

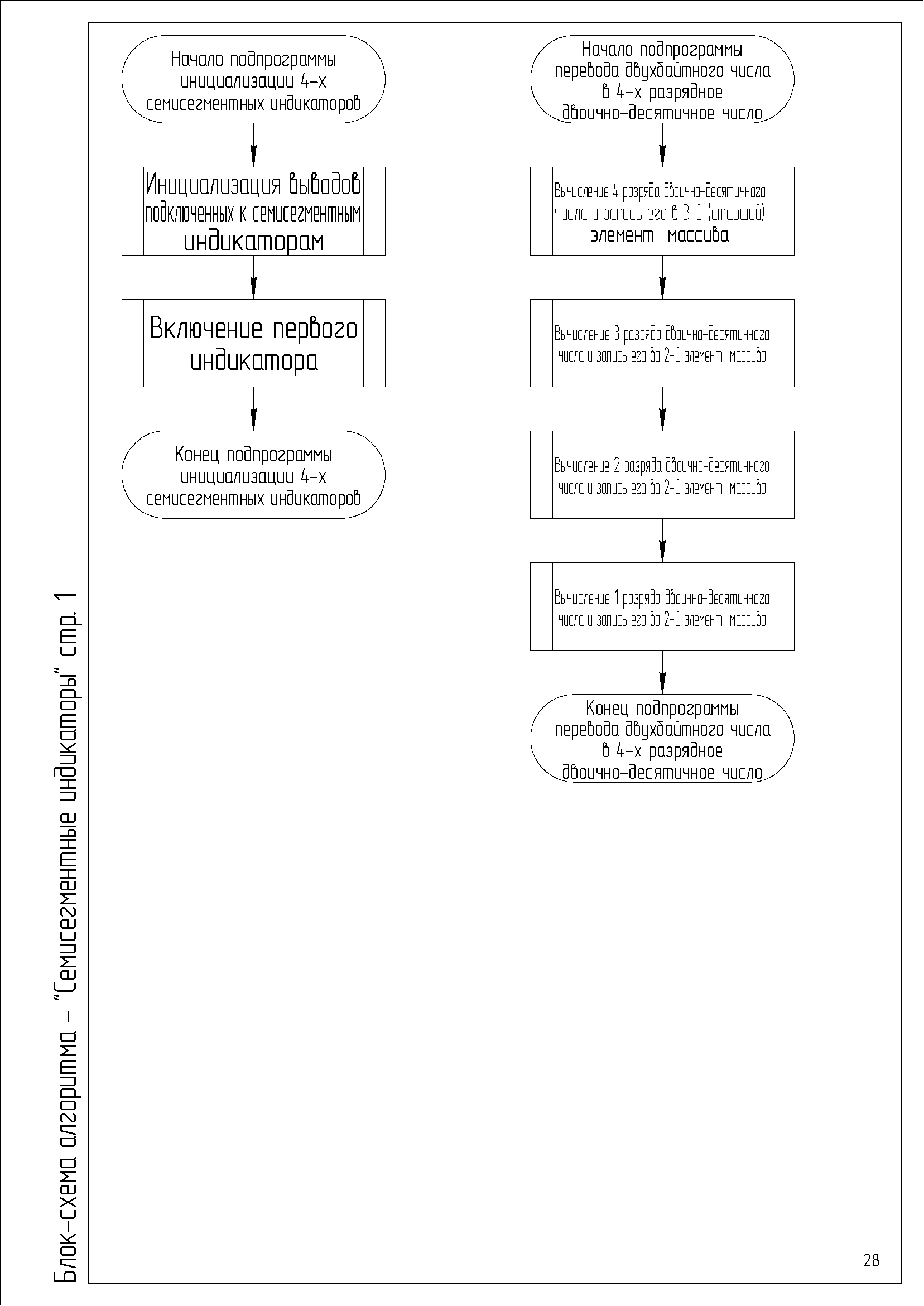


Рисунок А. 32 — Блок схема алгоритма, лист 28

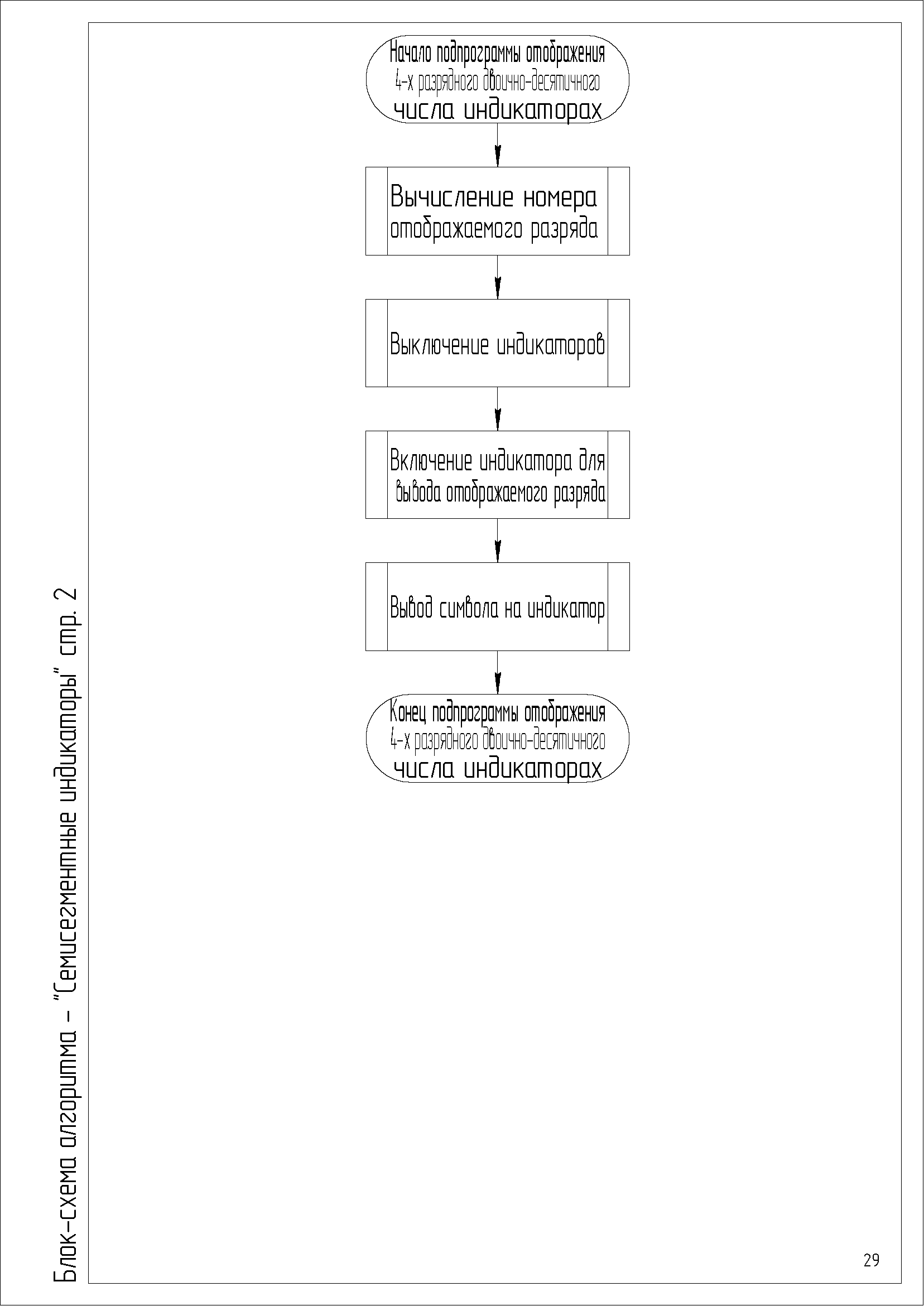


Рисунок А. 33 — Блок схема алгоритма, лист 2

Файл main.c

#include <avr/interrupt.h>

#include <avr/io.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <util/atomic.h>

#include <util/delay.h>

#include "lcd16x2.h"

#include "main.h"

#include "max31865.h"

#include "pcf8583.h"

#include "sdcard.h"

#include "sevseg.h"

#include "twi.h"

#define F\_CPU 8000000UL

#define BTN\_MASK (1 << PC3)

#define BTN\_FWD PC5

#define BTN\_BWD PC6

#define BTN\_FWD\_MASK (1 << BTN\_FWD)

#define BTN\_BWD\_MASK (1 << BTN\_BWD)

#define LED\_MASK (1 << PD7)

#define HOLD\_INITIAL\_DELAY\_MS 250

#define REP\_DELAY\_MIN\_MS 1

#define REP\_DELAY\_BASE\_MS 500

#define SCROLL\_PAUSE\_MS 1000

#define SCROLL\_STEP\_MS 250

#define ERR\_SD 1

#define ERR\_RTC 2

#define ERR\_TEMP 4

static volatile uint16\_t fwd\_hold\_cnt, fwd\_rep\_cnt;

static volatile uint16\_t bwd\_hold\_cnt, bwd\_rep\_cnt;

static volatile uint32\_t ms\_ticks = 0;

static volatile uint8\_t nav\_end\_event = 0;

static volatile uint8\_t nav\_start\_event = 0;

static volatile uint32\_t last\_sent\_meas = 0;

static uint8\_t error\_state = 0;

struct {

uint8\_t one\_sec : 1;

uint8\_t ten\_sec : 1;

uint8\_t nav\_fwd : 1;

uint8\_t nav\_bwd : 1;

uint8\_t uart\_dump : 1;

uint8\_t btn\_lock : 1;

uint8\_t history : 1;

uint8\_t hist : 1;

uint8\_t err : 1;

uint8\_t lcd\_clear : 1;

} volatile flags;

static uint32\_t meas\_no = 0;

static uint32\_t nav\_pos = 0;

static struct pcf\_time rtc\_time;

volatile uint8\_t disp\_digits[4];

static char linebuf[32];

static char linebuf\_hist[32];

static char scroll\_buf[32];

static uint8\_t scroll\_len = 0, scroll\_ofs = 0;

static uint32\_t scroll\_t0 = 0;

static uint32\_t last\_sent\_entry = 0;

static char\* strip\_date(char\* s) {

char\* p = strchr(s, ' ');

if (!p) return s;

char\* q = strchr(p + 1, ' ');

if (!q) return s;

memmove(p + 1, q + 1, strlen(q + 1) + 1);

return s;

}

static inline uint8\_t bcd2dec(uint8\_t v) {

return ((v >> 4) \* 10) + (v & 0x0F);

}

static inline void time2str(const struct pcf\_time\* t, char\* buf) {

sprintf(buf, "%02u:%02u:%02u", bcd2dec(t->hours & 0x3F),

bcd2dec(t->minutes & 0x7F), bcd2dec(t->seconds & 0x7F));

}

static void temperature\_to\_digits\_tenth(float temp) {

uint8\_t digits[4];

uint8\_t negative = (temp < 0);

if (negative) temp = -temp;

uint16\_t hundredths = (uint16\_t)(temp \* 100.0f + 0.5f);

if (hundredths > 9999) hundredths = 9999;

sevseg\_bin2bcd(hundredths, digits);

if (negative)

digits[3] = 0x0A;

else if (digits[3] == 0)

digits[3] = 0;

ATOMIC\_BLOCK(ATOMIC\_RESTORESTATE) {

for (uint8\_t i = 0; i < 4; i++) disp\_digits[i] = digits[i];

}

sevseg\_set\_dot(2);

sevseg\_blank\_leading(1);

}

static void go\_to\_log\_end(void) {

nav\_pos = meas\_no;

if (sd\_read\_line\_at(nav\_pos, linebuf, sizeof linebuf) == 0) {

char\* e = strchr(linebuf, '\n');

if (e) \*e = 0;

lcd\_mov\_cursor(16);

strip\_date(linebuf);

lcd\_disp\_str((uint8\_t\*)linebuf);

lcd\_disp\_str((uint8\_t\*)" ");

strncpy(scroll\_buf, linebuf, sizeof scroll\_buf - 1);

scroll\_buf[sizeof scroll\_buf - 1] = 0;

scroll\_len = strlen(scroll\_buf);

scroll\_ofs = 0;

scroll\_t0 = ms\_ticks;

}

}

static void go\_to\_log\_start(void) {

nav\_pos = 1;

if (sd\_read\_line\_at(nav\_pos, linebuf, sizeof linebuf) == 0) {

char\* e = strchr(linebuf, '\n');

if (e) \*e = 0;

lcd\_mov\_cursor(16);

strip\_date(linebuf);

lcd\_disp\_str((uint8\_t\*)linebuf);

lcd\_disp\_str((uint8\_t\*)" ");

strncpy(scroll\_buf, linebuf, sizeof scroll\_buf - 1);

scroll\_buf[sizeof scroll\_buf - 1] = 0;

scroll\_len = strlen(scroll\_buf);

scroll\_ofs = 0;

scroll\_t0 = ms\_ticks;

}

}

static inline void uart\_putc(char c) {

while (!(UCSRA & (1 << UDRE)));

UDR = c;

}

static void uart\_puts(const char\* s) {

while (\*s) uart\_putc(\*s++);

}

static void scroll\_update(void) {

if (scroll\_len <= 16) return;

uint32\_t now = ms\_ticks;

if (scroll\_ofs == 0 && now - scroll\_t0 < SCROLL\_PAUSE\_MS) return;

if (now - scroll\_t0 >= SCROLL\_STEP\_MS) {

scroll\_t0 = now;

if (++scroll\_ofs > scroll\_len - 16) {

scroll\_ofs = 0;

scroll\_t0 = now;

}

lcd\_mov\_cursor(16);

for (uint8\_t i = 0; i < 16; i++) {

char c = (i + scroll\_ofs < scroll\_len) ? scroll\_buf[i + scroll\_ofs]

: ' ';

lcd\_send\_char(c);

}

}

}

static inline uint8\_t pcf\_day(const struct pcf\_time\* t) {

return bcd2dec(t->day & 0x3F);

}

static inline uint8\_t pcf\_month(const struct pcf\_time\* t) {

return bcd2dec(t->month & 0x1F);

}

static inline uint8\_t pcf\_year(const struct pcf\_time\* t) {

uint8\_t yy\_compile =

((\_\_DATE\_\_[9] - '0') \* 10 + (\_\_DATE\_\_[10] - '0')); // «25» для 2025 г.

uint8\_t base = yy\_compile & 0xFC; // 24, 28, 32 …

return base + (t->day >> 6); // + YY(1-0)

}

static void show\_error(const char\* msg) {

lcd\_mov\_cursor(0);

lcd\_disp\_str((uint8\_t\*)msg);

\_delay\_ms(1000);

}

ISR(TIMER1\_COMPA\_vect) {

static uint8\_t sec\_cnt = 0;

flags.one\_sec = 1;

if (++sec\_cnt >= 1) {

sec\_cnt = 0;

flags.ten\_sec = 1;

}

}

ISR(TIMER0\_OVF\_vect) {

if (flags.uart\_dump || flags.err)

PORTA = 0x00;

else

sevseg\_display\_process(disp\_digits);

ms\_ticks++;

static uint8\_t prev = BTN\_FWD\_MASK | BTN\_BWD\_MASK | (1 << PC7) | (1 << PC3);

uint8\_t cur =

PINC & (BTN\_FWD\_MASK | BTN\_BWD\_MASK | (1 << PC7) | (1 << PC3));

uint8\_t changed = prev ^ cur;

uint8\_t pressed\_edge = changed & (~cur);

static uint32\_t last\_fwd\_ms = 0, last\_bwd\_ms = 0;

static uint8\_t fwd\_clicks = 0, bwd\_clicks = 0;

const uint32\_t DOUBLE\_CLICK\_MS = 70;

if (pressed\_edge & BTN\_FWD\_MASK) {

uint32\_t now = ms\_ticks;

fwd\_clicks =

(now - last\_fwd\_ms <= DOUBLE\_CLICK\_MS) ? fwd\_clicks + 1 : 1;

last\_fwd\_ms = now;

if (fwd\_clicks == 2) {

nav\_end\_event = 1;

fwd\_clicks = 0;

}

flags.nav\_fwd = 1;

fwd\_hold\_cnt = 0;

fwd\_rep\_cnt = HOLD\_INITIAL\_DELAY\_MS;

}

if (!(cur & BTN\_FWD\_MASK)) {

if (fwd\_hold\_cnt < UINT16\_MAX) fwd\_hold\_cnt++;

if (fwd\_hold\_cnt >= HOLD\_INITIAL\_DELAY\_MS) {

if (fwd\_rep\_cnt == 0) {

flags.nav\_fwd = 1;

uint32\_t delay = REP\_DELAY\_BASE\_MS / (meas\_no ? meas\_no : 1);

if (delay < REP\_DELAY\_MIN\_MS) delay = REP\_DELAY\_MIN\_MS;

fwd\_rep\_cnt = delay;

} else

fwd\_rep\_cnt--;

}

} else

fwd\_hold\_cnt = 0;

if (pressed\_edge & BTN\_BWD\_MASK) {

uint32\_t now = ms\_ticks;

bwd\_clicks =

(now - last\_bwd\_ms <= DOUBLE\_CLICK\_MS) ? bwd\_clicks + 1 : 1;

last\_bwd\_ms = now;

if (bwd\_clicks == 2) {

nav\_start\_event = 1;

bwd\_clicks = 0;

}

flags.nav\_bwd = 1;

bwd\_hold\_cnt = 0;

bwd\_rep\_cnt = HOLD\_INITIAL\_DELAY\_MS;

}

if (!(cur & BTN\_BWD\_MASK)) {

if (bwd\_hold\_cnt < UINT16\_MAX) bwd\_hold\_cnt++;

if (bwd\_hold\_cnt >= HOLD\_INITIAL\_DELAY\_MS) {

if (bwd\_rep\_cnt == 0) {

flags.nav\_bwd = 1;

uint32\_t delay = REP\_DELAY\_BASE\_MS / (meas\_no ? meas\_no : 1);

if (delay < REP\_DELAY\_MIN\_MS) delay = REP\_DELAY\_MIN\_MS;

bwd\_rep\_cnt = delay;

} else

bwd\_rep\_cnt--;

}

} else

bwd\_hold\_cnt = 0;

if (pressed\_edge & (1 << PC3)) flags.history = 1;

if (pressed\_edge & (1 << PC7)) flags.uart\_dump = 1;

prev = cur;

}

static void uart\_dump\_log(void) {

uint32\_t current\_entry = 0;

sd\_iter\_reset();

lcd\_mov\_cursor(0);

for (uint8\_t i = 0; i < 16; i++) lcd\_send\_char(' ');

uint8\_t shown\_blocks = 0;

static char current\_msg[16];

lcd\_mov\_cursor(0);

lcd\_disp\_str((uint8\_t\*)"UART");

while (current\_entry < meas\_no) {

++current\_entry;

if (current\_entry > last\_sent\_entry) {

sprintf(current\_msg, "%lu/%lu", current\_entry, meas\_no);

sd\_read\_line\_at(current\_entry, linebuf, sizeof linebuf);

uart\_puts(linebuf);

uart\_putc('\n');

}

}

if (current\_entry > last\_sent\_entry) {

last\_sent\_meas += current\_entry - last\_sent\_entry;

last\_sent\_entry = current\_entry;

}

sd\_iter\_to\_end();

\_delay\_ms(500);

}

static void uart\_init(void) {

UCSRA |= (1 << U2X);

UBRRL = 3;

UCSRC |= (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);

UCSRB |= (1 << RXCIE) | (1 << RXEN) | (1 << TXEN);

}

static void timer0\_init(void) {

TCCR0 = (1 << CS01) | (1 << CS00);

TIMSK |= (1 << TOIE0);

}

static void timer1\_init(void) {

TCCR1B = (1 << WGM12) | (1 << CS12);

OCR1A = 255;

TIMSK |= (1 << OCIE1A);

}

static void port\_init(void) {

DDRC &= ~(BTN\_MASK | BTN\_FWD\_MASK | BTN\_BWD\_MASK);

PORTC |= (BTN\_MASK | BTN\_FWD\_MASK | BTN\_BWD\_MASK);

DDRC &= ~(1 << PC7);

PORTC |= (1 << PC7);

}

static uint8\_t cnt\_err = 0;

static void update\_error\_display(void) {

if (cnt\_err == 0) {

lcd\_mov\_cursor(0);

lcd\_disp\_str(" ");

cnt\_err = 1;

}

if (error\_state & ERR\_SD)

show\_error("NO SD");

else if (error\_state & ERR\_RTC)

show\_error("NO RTC");

else if (error\_state & ERR\_TEMP)

show\_error("NO TEMP");

}

int main(void) {

float temperature = 0.0f;

uart\_init();

port\_init();

sevseg\_init();

lcd\_init();

lcd\_disp\_str((uint8\_t\*)"1");

max31865\_init(0, 0);

lcd\_disp\_str((uint8\_t\*)"2");

lcd\_mov\_cursor(16);

timer0\_init();

timer1\_init();

sd\_iter\_reset();

twi\_init();

sei();

pcf\_init();

if (sd\_erase\_sector(1) != 0) {

error\_state |= ERR\_SD;

}

if (error\_state & ERR\_SD) show\_error("NO SD");

twi\_status = TWI\_STATUS\_READY;

pcf\_read\_time(&rtc\_time);

uint16\_t tout = 1000;

while (twi\_status == TWI\_STATUS\_RX\_BUSY && tout--) {

\_delay\_ms(1);

}

if (twi\_status != TWI\_STATUS\_RX\_COMPLETE) {

error\_state |= ERR\_RTC;

}

if (error\_state & ERR\_RTC)

show\_error("NO RTC");

else

twi\_status = TWI\_STATUS\_READY;

meas\_no = sd\_get\_entry\_count();

while (1) {

if (flags.one\_sec) {

flags.one\_sec = 0;

pcf\_read\_time(&rtc\_time);

uint16\_t tmo = 1000;

while (twi\_status == TWI\_STATUS\_RX\_BUSY && tmo--) {

\_delay\_ms(1);

}

if (twi\_status != TWI\_STATUS\_RX\_COMPLETE) {

error\_state |= ERR\_RTC;

} else {

error\_state &= ~ERR\_RTC;

twi\_status = TWI\_STATUS\_READY;

}

temperature = max31865\_read\_temperature();

if (temperature > 85.0f) {

error\_state |= ERR\_TEMP;

} else {

error\_state &= ~ERR\_TEMP;

}

if (error\_state == 0) {

temperature\_to\_digits\_tenth(temperature);

char tstr[10], cnt[6];

time2str(&rtc\_time, tstr);

sprintf(cnt, "%lu", meas\_no + last\_sent\_meas);

lcd\_mov\_cursor(0);

lcd\_disp\_str((uint8\_t\*)tstr);

lcd\_mov\_cursor(10);

lcd\_disp\_str((uint8\_t\*)"#");

lcd\_disp\_str((uint8\_t\*)cnt);

}

}

if (error\_state & ERR\_SD) {

sd\_init();

if (sd\_erase\_sector(meas\_no + 1) == 0) {

error\_state &= ~ERR\_SD;

// lcd\_send\_cmd(1 << LCD\_CLR);

//\_delay\_ms(2);

// meas\_no = sd\_get\_entry\_count();

// nav\_pos = meas\_no;

}

}

if (error\_state) {

flags.err = 1;

update\_error\_display();

scroll\_update();

continue;

}

if (flags.err == 1) {

sd\_init();

pcf\_init();

}

cnt\_err = 0;

flags.err = 0;

if (flags.ten\_sec) {

flags.ten\_sec = 0;

if (!flags.btn\_lock) {

flags.btn\_lock = 1;

int16\_t t100 = (int16\_t)(temperature \* 100.0f + 0.5f);

++meas\_no;

sprintf(linebuf, "%lu %02u.%02u.%02u %02u:%02u,%+d.%02u",

meas\_no + last\_sent\_meas, // № измерения

pcf\_day(&rtc\_time), // ДД

pcf\_month(&rtc\_time), // ММ

pcf\_year(&rtc\_time), // ГГ

bcd2dec(rtc\_time.hours & 0x3F), // чч

bcd2dec(rtc\_time.minutes & 0x7F), // мм

t100 / 100, abs(t100) % 100); // t,xx

memset(linebuf\_hist, 0, sizeof linebuf\_hist);

strncpy(linebuf\_hist, linebuf, sizeof linebuf\_hist - 1);

strcat(linebuf, "\n");

if (sd\_write\_line(linebuf) != 0) {

error\_state |= ERR\_SD;

} else {

error\_state &= ~ERR\_SD;

sd\_erase\_sector(meas\_no + 1);

}

if (flags.hist) {

lcd\_mov\_cursor(16);

strip\_date(linebuf\_hist);

lcd\_disp\_str((uint8\_t\*)linebuf\_hist);

lcd\_disp\_str((uint8\_t\*)" ");

strncpy(scroll\_buf, linebuf\_hist, sizeof scroll\_buf - 1);

scroll\_buf[sizeof scroll\_buf - 1] = 0;

scroll\_len = strlen(scroll\_buf);

scroll\_ofs = 0;

scroll\_t0 = ms\_ticks;

}

flags.btn\_lock = 0;

}

}

if (flags.history && !flags.btn\_lock) {

flags.history = 0;

flags.hist = !flags.hist;

nav\_end\_event = nav\_start\_event = 0;

if (flags.hist) {

lcd\_mov\_cursor(16);

strip\_date(linebuf\_hist);

lcd\_disp\_str((uint8\_t\*)linebuf\_hist);

lcd\_disp\_str((uint8\_t\*)" ");

strncpy(scroll\_buf, linebuf\_hist, sizeof scroll\_buf - 1);

scroll\_buf[sizeof scroll\_buf - 1] = 0;

} else {

sd\_iter\_reset();

nav\_pos = meas\_no;

sd\_read\_line\_at(meas\_no, linebuf, sizeof linebuf);

char\* e = strchr(linebuf, '\n');

if (e) \*e = 0;

lcd\_mov\_cursor(16);

strip\_date(linebuf);

lcd\_disp\_str((uint8\_t\*)linebuf);

lcd\_disp\_str((uint8\_t\*)" ");

strncpy(scroll\_buf, linebuf, sizeof scroll\_buf - 1);

scroll\_buf[sizeof scroll\_buf - 1] = 0;

}

scroll\_len = strlen(scroll\_buf);

scroll\_ofs = 0;

scroll\_t0 = ms\_ticks;

}

if (!flags.hist) {

if (nav\_end\_event && !flags.btn\_lock) {

nav\_end\_event = 0;

flags.btn\_lock = 1;

sd\_iter\_to\_end();

go\_to\_log\_end();

flags.btn\_lock = 0;

}

if (nav\_start\_event && !flags.btn\_lock) {

nav\_start\_event = 0;

flags.btn\_lock = 1;

sd\_iter\_reset();

go\_to\_log\_start();

flags.nav\_bwd = 0;

flags.btn\_lock = 0;

}

if (flags.nav\_fwd && !flags.btn\_lock) {

flags.nav\_fwd = 0;

if (nav\_pos < meas\_no &&

sd\_read\_line\_at(++nav\_pos, linebuf, sizeof linebuf) == 0) {

char\* e = strchr(linebuf, '\n');

if (e) \*e = 0;

lcd\_mov\_cursor(16);

strip\_date(linebuf);

lcd\_disp\_str((uint8\_t\*)linebuf);

lcd\_disp\_str((uint8\_t\*)" ");

strncpy(scroll\_buf, linebuf, sizeof scroll\_buf - 1);

scroll\_buf[sizeof scroll\_buf - 1] = 0;

scroll\_len = strlen(scroll\_buf);

scroll\_ofs = 0;

scroll\_t0 = ms\_ticks;

}

}

if (flags.nav\_bwd && !flags.btn\_lock) {

flags.nav\_bwd = 0;

if (nav\_pos > 1 &&

sd\_read\_line\_at(--nav\_pos, linebuf, sizeof linebuf) == 0) {

char\* e = strchr(linebuf, '\n');

if (e) \*e = 0;

lcd\_mov\_cursor(16);

strip\_date(linebuf);

lcd\_disp\_str((uint8\_t\*)linebuf);

lcd\_disp\_str((uint8\_t\*)" ");

strncpy(scroll\_buf, linebuf, sizeof scroll\_buf - 1);

scroll\_buf[sizeof scroll\_buf - 1] = 0;

scroll\_len = strlen(scroll\_buf);

scroll\_ofs = 0;

scroll\_t0 = ms\_ticks;

}

}

}

if (flags.uart\_dump && !flags.btn\_lock) {

flags.btn\_lock = 1;

lcd\_send\_cmd(1 << LCD\_CLR);

\_delay\_ms(2);

uart\_dump\_log();

sd\_erase\_sector(0);

sd\_clear\_log(1);

meas\_no = 0;

nav\_pos = 0;

last\_sent\_entry = 0;

lcd\_send\_cmd(1 << LCD\_CLR);

\_delay\_ms(2);

lcd\_mov\_cursor(10);

lcd\_disp\_str((uint8\_t\*)"#0 ");

flags.btn\_lock = 0;

flags.uart\_dump = 0;

}

scroll\_update();

}

}

Файл lcd.h

#ifndef LCD16X2\_H

#define LCD16X2\_H

#include <avr/io.h>

/\* Битовые поля для команд LCD дисплея \*/

/\* Очистка дисплея с установкой курсора в начало первой строки \*/

#define LCD\_CLR 0 // Старший бит инструкции (DB0)

/\* Установка курсора в начало первой строки \*/

#define LCD\_HOME 1 // Старший бит инструкции (DB1)

/\* Установка направления вывода символов, разрешение сдвига экрана \*/

#define LCD\_MODE 2 // Старший бит инструкции (DB2)

#define LCD\_MODE\_I\_D \

1 // Направление вывода символов (DB1)

// 0: справа налево; 1: слева направо

#define LCD\_MODE\_SH 0 // Разрешение сдвига экрана при выводе символов (DB0)

/\* Включение/выключение дисплея и отображения курсора \*/

#define LCD\_ON 3 // Старший бит инструкции (DB3)

#define LCD\_ON\_D 2 // Включение дисплея (DB2)

#define LCD\_ON\_C 1 // Включение курсора (DB1)

#define LCD\_ON\_B 0 // Включение мигания курсора (DB0)

/\* Сдвиг курсора и экрана \*/

#define LCD\_SHIFT 4 // Старший бит инструкции (DB4)

#define LCD\_SHIFT\_S\_C \

3 // Разрешение сдвига экрана (DB3)

// 0: сдвиг только курсора; 1: сдвиг курсора и экрана

#define LCD\_SHIFT\_R\_L \

2 // Направление сдвига (DB2)

// 0: сдвиг влево; 1: сдвиг вправо

/\* Настройка интерфейса, количества строк для вывода, размера шрифта ЭТО В

\* FUNCTION SET в документации можно посмотреть\*/

#define LCD\_F 5 // Старший бит инструкции (DB5)

#define LCD\_F\_DL \

4 // Выбор интерфейса (DB4)

// 0: 4-битный интерфейс; 1: 8-битный интерфейс

#define LCD\_F\_N \

3 // Количество строк для вывода (DB3)

// 0: 1 строка; 1: 2 строки

#define LCD\_F\_F 2 // Размер шрифта (DB2) 0: 5х8 пикселей; 1: 5х11 пикселей

/\* Запись адреса DDRAM памяти в адресный указатель \*/

#define LCD\_DDRAM 7 // Старший бит инструкции (DB7)

void lcd\_init(void); // инициализация дисплея

void lcd\_disp\_buf(uint8\_t \*buf, uint8\_t size);

void lcd\_disp\_str(uint8\_t \*str);

void lcd\_mov\_cursor(uint8\_t new\_pos);

void lcd\_send\_cmd(uint8\_t cmd);

void lcd\_send\_char(uint8\_t data);

#endif

/\* End File \*/

Файл lcd.c

#define F\_CPU 8000000UL

#include <util/delay.h>

#include "lcd16x2.h"

#define LCD\_PORT PORTD // Порт к которому подключен LCD дисплей

#define LCD\_DDR DDRD // Выводы данных LCD должны быть подключены к старшим

#define LCD\_PIN PIND // выводам порта микроконтроллера

#define LCD\_DATA\_MASK 0xF0 // Маска выводов данных дисплея

#define LCD\_E 3 // Номер вывода E (запись данных в LCD осуществляется по спаду)

#define LCD\_RS 2 // Номер вывода RS (1 - данные; 0 - инструкция)

#define LCD\_PIN\_MASK (LCD\_DATA\_MASK | (1 << LCD\_RS) | (1 << LCD\_E))

/\*\*

\* Функция отправки полубайта данных/команды на дисплей

\* Код полубайта должен находится в старшем полубайте аргумента hfbyte

\*

\* \param hfbyte код полубайта данных/команды (старший полубайт)

\*/

static inline void lcd\_send\_hfbyte(uint8\_t hfbyte) {

LCD\_PORT |= (1 << LCD\_E);

/\* Выводы данных дисплея подключены к старшим разрядам порта МК \*/

uint8\_t temp = LCD\_PORT;

temp &= ~LCD\_DATA\_MASK;

temp |= hfbyte;

LCD\_PORT = temp;

LCD\_PORT &= ~(1 << LCD\_E); // защелкивание данных на дисплее

}

/\*\*

\* Функция отправки старшего полубайта команды на дисплей (используется при

\* инициализации). Код половины команды должен находится в старшем полубайте

\* аргумента hfcmd

\*

\* \param hfcmd старший полубайт команды

\*/

static void lcd\_send\_hfcmd(uint8\_t hfcmd) {

LCD\_PORT &= ~(1 << LCD\_RS);

lcd\_send\_hfbyte(hfcmd);

}

/\*\*

\* Функция отправки байта данных/команды на дисплей

\*

\* \param byte байт данных/команды

\*/

static void lcd\_send\_byte(uint8\_t byte) {

\_delay\_us(50);

lcd\_send\_hfbyte(byte & 0xF0);

lcd\_send\_hfbyte(byte << 4);

}

/\*\*

\* Функция вывода на дисплей символа

\* Время выполнения составляет 50 мкс

\*

\* \param symbol код символа

\*/

void lcd\_send\_char(uint8\_t symbol) {

LCD\_PORT |= (1 << LCD\_RS); // говорим, что будем передавать данные

lcd\_send\_byte(symbol); // отправляем байт

}

/\*\*

\* Функция отправки команды на дисплей

\* Время выполнения команд "Очистка дисплея с установкой курсора в начало первой

\* строки" и "Установка курсора в начало первой строки" составляет 1.5 мс;

\* время выполнения остальных команд составляет 50 мкс.

\*

\* \param cmd код команды

\*/

void lcd\_send\_cmd(uint8\_t cmd) {

LCD\_PORT &= ~(1 << LCD\_RS); // вывод в 0

lcd\_send\_byte(cmd);

}

/\* Функция инициализации дисплея LCD16x2 \*/

void lcd\_init(void) {

DDRD |= (1 << PD2) | (1 << PD3) | (1 << PD4) | (1 << PD5) | (1 << PD6) |

(1 << PD7);

/\* Задержка не менее 15 мс, после подачи питания на дисплей \*/

\_delay\_ms(15);

/\*\*

\* Инициализация дисплея и перевод его на 4-х битный интерфейс

\* согласно порядку, приведенному в datasheet

\* Пока не перевели на 4-битный интерфейс отправляем команды по полбайта

\* Отправка старшего полубайта команды настройки 8-битного интерфейса

\* Задержка не менее 4.1 мс

\* Отправка старшего полубайта команды настройки 8-битного интерфейса

\* Задержка не менее 100 мкс

\* Отправка старшего полубайта команды настройки 8-битного интерфейса

\* Задержка не менее 50 мкс

\* Отправка старшего полубайта команды настройки 4-битного интерфейса

\*/

lcd\_send\_hfcmd((1 << LCD\_F) | (1 << LCD\_F\_DL));

\_delay\_ms(5);

lcd\_send\_hfcmd((1 << LCD\_F) | (1 << LCD\_F\_DL));

\_delay\_us(100);

lcd\_send\_hfcmd((1 << LCD\_F) | (1 << LCD\_F\_DL));

\_delay\_us(50);

lcd\_send\_hfcmd(1 << LCD\_F);

/\*\*

\* Перевод дисплея на 4-битный интерфейс завершен. Теперь можно посылать

\* команды/данные по полному байту

\*/

/\* Подтверждение 4-битного интерфейса \*/

lcd\_send\_cmd((1 << LCD\_F) | (1 << LCD\_F\_N));

/\* Очистка дисплея, дисплей очищается не менее 1.5 мс \*/

lcd\_send\_cmd(1 << LCD\_CLR);

\_delay\_ms(2);

/\* Включение экрана дисплея \*/

lcd\_send\_cmd((1 << LCD\_ON) | (1 << LCD\_ON\_D));

}

/\*\*

\* Функция вывода на дисплей массива символов

\*

\* \param buf указатель на массив

\* \param size размер массива

\*/

void lcd\_disp\_buf(uint8\_t \*buf, uint8\_t size) {

do {

lcd\_send\_char(\*buf++); // каждый раз отправляем символ через указатель

// массива операция по передаче в функцию \*buf++

// эквивалетно: cd\_send\_char(\*buf); buf++;

} while (--size); // сравниваем с нулем здесь преддекремент сначала

// вычитаем единицу, потом сравниваем с 0 сначала надо

// уменьшить символ, потом уже проверить

}

/\*\*

\* Функция вывода на дисплей строки

\*

\* \param str указатель на строку

\*/

void lcd\_disp\_str(uint8\_t \*str) {

do {

lcd\_send\_char(\*str);

} while (\*++str);

}

/\*\*

\* Функция изменения позиции курсора дисплея

\*

\* \param new\_pos новая позиция курсора дисплея

\*/

void lcd\_mov\_cursor(uint8\_t new\_pos) {

/\* code \*/

uint8\_t addr;

// new\_pos от 0 до 31 (0-15 первая строка, 16-31 вторая)

if (new\_pos < 16) {

addr = new\_pos;

} else {

addr = (new\_pos - 16) + 0x40;

}

lcd\_send\_cmd((1 << LCD\_DDRAM) | addr);

}

/\* End File \*/

Файл max31865.h

#ifndef MAX31865\_H

#define MAX31865\_H

#include <avr/io.h>

#define F\_CPU 8000000UL

#include <stdint.h>

#include <util/delay.h>

/\* ===== Register Map ===== \*/

#define MAX31865\_REG\_CONFIG 0x00

#define MAX31865\_REG\_RTD\_MSB 0x01

#define MAX31865\_REG\_RTD\_LSB 0x02

#define MAX31865\_REG\_HFT\_MSB 0x03

#define MAX31865\_REG\_HFT\_LSB 0x04

#define MAX31865\_REG\_LFT\_MSB 0x05

#define MAX31865\_REG\_LFT\_LSB 0x06

#define MAX31865\_REG\_FAULT\_STATUS 0x07

/\* ===== Configuration Bits ===== \*/

#define MAX31865\_CFG\_BIAS (1 << 7)

#define MAX31865\_CFG\_MODE\_AUTO (1 << 6)

#define MAX31865\_CFG\_1SHOT (1 << 5)

#define MAX31865\_CFG\_3WIRE (1 << 4)

#define MAX31865\_CFG\_FAULT\_CYCLE\_NONE (0 << 2)

#define MAX31865\_CFG\_FAULT\_CYCLE\_AUTO (1 << 2)

#define MAX31865\_CFG\_FAULT\_CYCLE\_MAN1 (2 << 2)

#define MAX31865\_CFG\_FAULT\_CYCLE\_MAN2 (3 << 2)

#define MAX31865\_CFG\_CLEAR\_FAULT (1 << 1)

#define MAX31865\_CFG\_FILT50HZ (1 << 0)

/\* ===== Conversion Constants ===== \*/

#define MAX31865\_RREF 425.0f

#define MAX31865\_R0 100.0f

#define MAX31865\_A 3.9083e-3f

#define MAX31865\_B -5.775e-7f

#define MAX31865\_C -4.18301e-12f

#define MAX31865\_CS\_PORT PORTB

#define MAX31865\_CS\_DDR DDRB

#define MAX31865\_CS\_PIN PB4 /\* SS \*/

/\* ===== Public API ===== \*/

void max31865\_init(uint8\_t three\_wire, uint8\_t filter50Hz);

float max31865\_read\_temperature(void);

uint16\_t max31865\_read\_raw(void);

uint8\_t max31865\_read\_fault(void);

void max31865\_clear\_fault(void);

#endif // MAX31865\_H

//

Файл max31865.c

#include "max31865.h"

#include "sdcard.h"

#include "spi.h"

static inline void cs\_low(void) { MAX31865\_CS\_PORT &= ~(1 << MAX31865\_CS\_PIN); }

static inline void cs\_high(void) { MAX31865\_CS\_PORT |= (1 << MAX31865\_CS\_PIN); }

static void write\_reg(uint8\_t addr, uint8\_t val) {

cs\_low();

spi\_transfer(addr | 0x80); /\* write flag \*/

spi\_transfer(val);

cs\_high();

}

static uint8\_t read\_reg(uint8\_t addr) {

cs\_low();

spi\_transfer(addr & 0x7F);

uint8\_t v = spi\_transfer(0xFF);

cs\_high();

return v;

}

/\* Read 16‑bit register (RTD value, thresholds). \*/

static uint16\_t read\_reg16(uint8\_t addr) {

cs\_low();

spi\_transfer(addr & 0x7F);

uint8\_t msb = spi\_transfer(0xFF);

uint8\_t lsb = spi\_transfer(0xFF);

cs\_high();

return ((uint16\_t)msb << 8) | lsb;

}

void max31865\_init(uint8\_t three\_wire, uint8\_t filter50Hz) {

/\* Configure CS pin \*/

MAX31865\_CS\_DDR |= (1 << MAX31865\_CS\_PIN);

cs\_high();

spi\_init(); /\* in spi.c \*/

sd\_init();

sd\_clear\_log(1);

SPSR |= (1 << SPI2X);

SPCR &= ~((1 << SPR1) | (1 << SPR0));

uint8\_t cfg = 0;

if (three\_wire) cfg |= MAX31865\_CFG\_3WIRE;

if (filter50Hz) cfg |= MAX31865\_CFG\_FILT50HZ;

write\_reg(MAX31865\_REG\_CONFIG, cfg | MAX31865\_CFG\_BIAS); /\* Enable bias \*/

\_delay\_ms(10);

/\* Start automatic conversions \*/

write\_reg(MAX31865\_REG\_CONFIG,

cfg | MAX31865\_CFG\_BIAS | MAX31865\_CFG\_MODE\_AUTO);

}

uint16\_t max31865\_read\_raw(void) {

uint16\_t v = read\_reg16(MAX31865\_REG\_RTD\_MSB);

return (v >> 1) & 0x7FFF;

}

float max31865\_read\_temperature(void) {

uint16\_t raw = max31865\_read\_raw();

float Rt = (float)raw \* MAX31865\_RREF / 32768.0f;

/\* Callendar‑Van Dusen, simplify for >=0 °C \*/

float Z = Rt / MAX31865\_R0;

float temp;

if (Rt >= MAX31865\_R0) {

/\* quadratic formula \*/

float d = (MAX31865\_A \* MAX31865\_A) - (4 \* MAX31865\_B \* (1 - Z));

temp = (-MAX31865\_A + sqrtf(d)) / (2 \* MAX31865\_B);

} else {

/\* Use full equation with C term for <0 °C \*/

/\* Solve iteratively (Newton) \*/

temp = -50.0f; /\* initial \*/

for (uint8\_t i = 0; i < 6; i++) {

float f = MAX31865\_R0 \*

(1 + MAX31865\_A \* temp + MAX31865\_B \* temp \* temp +

MAX31865\_C \* (temp - 100) \* temp \* temp \* temp) -

Rt;

float df = MAX31865\_R0 \*

(MAX31865\_A + 2 \* MAX31865\_B \* temp +

3 \* MAX31865\_C \* temp \* temp - 200 \* MAX31865\_C \* temp);

temp -= f / df;

}

}

return temp;

}

uint8\_t max31865\_read\_fault(void) {

return read\_reg(MAX31865\_REG\_FAULT\_STATUS);

}

void max31865\_clear\_fault(void) {

uint8\_t cfg = read\_reg(MAX31865\_REG\_CONFIG);

cfg &= ~MAX31865\_CFG\_MODE\_AUTO; /\* halt conv \*/

write\_reg(MAX31865\_REG\_CONFIG, cfg | MAX31865\_CFG\_CLEAR\_FAULT);

/\* restart \*/

write\_reg(MAX31865\_REG\_CONFIG,

cfg | MAX31865\_CFG\_BIAS | MAX31865\_CFG\_MODE\_AUTO);

}

// asd

Файл sdcard.h

#ifndef SDCARD\_H

#define SDCARD\_H

#include <avr/io.h>

#include <stdint.h>

/\* Chip-select \*/

#define SD\_CS\_PORT PORTC

#define SD\_CS\_DDR DDRC

#define SD\_CS\_PIN PC4

void sd\_init(void);

void sd\_dump\_log(void);

uint8\_t sd\_write\_line(const char \*str);

void sd\_iter\_reset(void);

uint8\_t sd\_read\_line(int8\_t dir, char \*dst, uint8\_t dst\_size);

void sd\_flush(void);

uint32\_t sd\_get\_entry\_count(void);

uint8\_t sd\_read\_line\_at(uint32\_t line\_num, char \*dst, uint8\_t dst\_sz);

void sd\_iter\_to\_end(void);

uint8\_t sd\_erase\_sector(uint32\_t lba);

void sd\_clear\_log(uint16\_t n\_sectors);

#endif

Файл sdcard.c

#include <avr/eeprom.h>

#include <string.h>

#include "sdcard.h"

#include "spi.h"

#define F\_CPU 8000000UL

#include <util/delay.h>

/\* -------------------------- SPI helpers -------------------------------- \*/

static inline void sd\_select(void) { SD\_CS\_PORT &= ~(1 << SD\_CS\_PIN); }

static inline void sd\_deselect(void) { SD\_CS\_PORT |= (1 << SD\_CS\_PIN); }

static uint8\_t spi\_x(uint8\_t v) { return spi\_transfer(v); }

/\* ------------------------- Global state -------------------------------- \*/

static uint8\_t sec\_buf[512];

static uint8\_t card\_sdhc = 0;

static uint32\_t next\_sector = 1; /\* куда писать следующую строку \*/

static uint32\_t it\_sector; /\* курсор при чтении \*/

uint32\_t EEMEM eeprom\_next\_sector;

/\* ====================================================================== \*/

static uint8\_t sd\_cmd(uint8\_t cmd, uint32\_t arg, uint8\_t crc) {

sd\_deselect();

spi\_x(0xFF); /\* 8†такт SCK с CS=1 \*/

sd\_select();

spi\_x(cmd | 0x40);

spi\_x(arg >> 24);

spi\_x(arg >> 16);

spi\_x(arg >> 8);

spi\_x(arg);

spi\_x(crc);

for (uint8\_t i = 0; i < 10; i++) {

uint8\_t r = spi\_x(0xFF);

if (!(r & 0x80)) return r;

}

return 0xFF; /\* timeout \*/

}

static uint8\_t wait\_token(uint8\_t token, uint16\_t tout\_ms) {

while (tout\_ms--) {

if (spi\_x(0xFF) == token) return 0;

\_delay\_us(200);

}

return 1;

}

static uint8\_t wait\_ready(uint16\_t tout\_ms) {

while (tout\_ms--) {

if (spi\_x(0xFF) == 0xFF) return 0;

\_delay\_us(200);

}

return 1;

}

static uint8\_t read\_sector(uint32\_t lba) {

if (lba == 0) return 1; /\* защита MBR \*/

uint32\_t addr = card\_sdhc ? lba : (lba << 9);

if (sd\_cmd(17, addr, 0xFF)) return 2; /\* READ\_SINGLE\_BLOCK \*/

if (wait\_token(0xFE, 100)) return 3; /\* data token \*/

for (uint16\_t i = 0; i < 512; i++) sec\_buf[i] = spi\_x(0xFF);

spi\_x(0xFF);

spi\_x(0xFF); /\* dummy CRC \*/

sd\_deselect();

spi\_x(0xFF);

return 0;

}

uint32\_t sd\_get\_entry\_count(void) {

return (next\_sector > 1) ? (next\_sector - 1) : 0;

}

static uint8\_t write\_sector(uint32\_t lba) {

if (lba == 0) return 5; /\* защита MBR \*/

uint32\_t addr = card\_sdhc ? lba : (lba << 9);

if (wait\_ready(250)) return 6;

if (sd\_cmd(24, addr, 0xFF)) return 1; /\* WRITE\_BLOCK \*/

spi\_x(0xFE); /\* start token \*/

for (uint16\_t i = 0; i < 512; i++) spi\_x(sec\_buf[i]);

spi\_x(0xFF);

spi\_x(0xFF); /\* dummy CRC \*/

uint8\_t resp = spi\_x(0xFF) & 0x1F;

if (resp != 0x05) return 2; /\* data?resp error \*/

if (wait\_ready(250)) return 3; /\* busy timeout \*/

sd\_deselect();

spi\_x(0xFF);

return 0;

}

/\* ------------------------- Tail detector -------------------------------- \*/

static void find\_tail(void) {

// uint32\_t s = 1;

// while (1) {

// if (read\_sector(s)) break; /\* ошибка чтени¤†Ц хвост=1 \*/

// if (sec\_buf[0] == 0xFF) break; /\* пустой сектор ? хвост \*/

// ++s;

// }

// next\_sector = s;

next\_sector = 1;

}

/\* ====================================================================== \*/

void sd\_init(void) {

SD\_CS\_DDR |= (1 << SD\_CS\_PIN);

SD\_CS\_PORT |= (1 << SD\_CS\_PIN);

for (uint8\_t i = 0; i < 10; i++) spi\_x(0xFF); /\* 80†тактов на пробуждение \*/

if (sd\_cmd(0, 0, 0x95) != 0x01) return; /\* CMD0 GO\_IDLE \*/

uint8\_t r = sd\_cmd(8, 0x1AA, 0x87); /\* CMD8 \*/

if (r == 0x01) {

for (uint8\_t i = 0; i < 4; i++) spi\_x(0xFF);

}

do { /\* ACMD41 \*/

sd\_cmd(55, 0, 0x65);

r = sd\_cmd(41, 0x40000000, 0x77);

} while (r != 0);

if (sd\_cmd(58, 0, 0xFF) == 0) { /\* CMD58 OCR \*/

uint32\_t ocr = 0;

for (uint8\_t i = 0; i < 4; i++) ocr = (ocr << 8) | spi\_x(0xFF);

card\_sdhc = (ocr & (1UL << 30)) ? 1 : 0;

}

sd\_deselect();

spi\_x(0xFF);

uint32\_t stored = eeprom\_read\_dword(&eeprom\_next\_sector);

if (stored >= 1 && stored < 0xFFFFFFFF) {

next\_sector = stored;

} else {

find\_tail();

}

}

void sd\_clear\_log(uint16\_t n\_sectors) {

memset(sec\_buf, 0xFF, 512);

for (uint16\_t i = 0; i < n\_sectors; i++) write\_sector(1 + i);

next\_sector = 1;

}

/\* --------------------------- WRITE line -------------------------------- \*/

uint8\_t sd\_write\_line(const char \*str) {

uint16\_t len = strlen(str);

if (!len || len > 510) return 4; /\* 0†/ слишком длинно \*/

memset(sec\_buf, 0xFF, 512);

memcpy(sec\_buf, str, len);

sec\_buf[len] = '\n';

uint8\_t res = write\_sector(next\_sector);

if (res == 0) {

++next\_sector;

eeprom\_update\_dword(&eeprom\_next\_sector, next\_sector);

}

return res;

}

void sd\_flush(void) { /\* no?op: всегда пишем полный сектор \*/ }

/\* --------------------------- READ iterator ------------------------------ \*/

void sd\_iter\_reset(void) { it\_sector = 1; }

static uint8\_t copy\_line(char \*dst, uint8\_t dst\_sz) {

uint16\_t i = 0;

while (i < dst\_sz - 1 && i < 512 && sec\_buf[i] != 0xFF &&

sec\_buf[i] != '\n') {

dst[i] = sec\_buf[i];

i++;

}

dst[i] = 0;

return 0;

}

uint8\_t sd\_read\_line(int8\_t dir, char \*dst, uint8\_t dst\_sz) {

/\* -------- вперед -------- \*/

if (dir > 0) {

if (read\_sector(it\_sector)) return 1;

if (sec\_buf[0] == 0xFF) return 2; /\* конец журнала \*/

uint8\_t r = copy\_line(dst, dst\_sz);

++it\_sector;

return r;

}

/\* -------- назад --------- \*/

if (it\_sector <= 1) return 3; /\* уже в начале \*/

do {

--it\_sector;

if (read\_sector(it\_sector)) return 1;

} while (sec\_buf[0] == 0xFF && it\_sector > 1);

return copy\_line(dst, dst\_sz);

}

uint8\_t sd\_read\_line\_at(uint32\_t line\_num, char \*dst, uint8\_t dst\_sz) {

// допустимый диапазон: [1, next\_sector-1]

if (line\_num < 1 || line\_num >= next\_sector) return 1;

// читаем нужный сектор

if (read\_sector(line\_num)) return 2;

// копируем строку из сектора в dst (останавливаемс¤ на '\n' или 0xFF)

return copy\_line(dst, dst\_sz);

}

void sd\_iter\_to\_end(void) { it\_sector = next\_sector; }

uint8\_t sd\_erase\_sector(uint32\_t lba) {

// заполн¤ем буфер сектора 0xFF

memset(sec\_buf, 0xFF, sizeof(sec\_buf));

// записываем буфер в сектор lba

return write\_sector(lba);

}

Файл twi.h

#ifndef TWI\_H

#define TWI\_H

#include <avr/io.h>

enum twi\_status {

TWI\_STATUS\_NOT\_INIT = 0,

TWI\_STATUS\_READY,

TWI\_STATUS\_TX\_BUSY,

TWI\_STATUS\_TX\_COMPLETE,

TWI\_STATUS\_RX\_BUSY,

TWI\_STATUS\_RX\_COMPLETE,

TWI\_STATUS\_ERROR,

};

extern volatile enum twi\_status twi\_status;

void twi\_init(void);

void twi\_write(uint8\_t slaw, uint8\_t reg\_addr, uint8\_t \*buf, uint8\_t size);

void twi\_read(uint8\_t slaw, uint8\_t reg\_addr, uint8\_t \*buf, uint8\_t size);

#endif

/\* End File \*/

Файл twi.c

#include <avr/pgmspace.h>

#include <avr/interrupt.h>

#include "twi.h"

#define LINE\_FREQ 100000UL // Частота обмена в Гц

#define F\_CPU 8000000UL

volatile enum twi\_status twi\_status = TWI\_STATUS\_NOT\_INIT;

static struct twi

{

uint8\_t slaw;

uint8\_t reg\_addr;

uint8\_t \*data;

uint8\_t count;

} twi;

/\*\*

\* Функция инициализации модуля TWI.

\* По окончанию инициализации устанавливается статус TWI\_STATUS\_READY.

\* Частота обмена в Гц: LINE\_FREQ.

\* Используется прерывание по событиям на шине TWI.

\*/

void twi\_init(void)

{

TWBR = (F\_CPU / LINE\_FREQ - 16) / 2;

TWCR |= (1 << TWEN) | (1 << TWIE);

twi\_status = TWI\_STATUS\_READY;

}

/\*\*

\* Функция записи/чтения в/из регистра ведомого устройства

\*

\* \param slaw адрес ведомого устройства SLA+W

\* \param reg\_addr адрес начального регистра для записи/чтения

\* \param buf указатель на буфер для записи/чтения

\* \param size размер буфера для записи/чтения

\*/

static inline void twi\_txrx(

uint8\_t slaw, uint8\_t reg\_addr, uint8\_t \*buf, uint8\_t size)

{

twi.slaw = slaw;

twi.reg\_addr = reg\_addr;

twi.data = buf;

twi.count = size;

TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN) | (1<<TWIE);

}

/\*\*

\* Функция записи в регистр ведомого устройства.

\* Во время записи устанавливается статус TWI\_STATUS\_TX\_BUSY, после записи

\* устанавливается статус TWI\_STATUS\_TX\_COMPLETE.

\*

\* \param slaw адрес ведомого устройства SLA+W

\* \param reg\_addr адрес начального регистра для записи

\* \param buf указатель на буфер для записи

\* \param size размер буфера для записи

\*/

void twi\_write(uint8\_t slaw, uint8\_t reg\_addr, uint8\_t \*buf, uint8\_t size)

{

twi\_txrx(slaw, reg\_addr, buf, size);

twi\_status = TWI\_STATUS\_TX\_BUSY;

}

/\*\*

\* Функция чтения из регистра ведомого устройства.

\* Во время чтения устанавливается статус TWI\_STATUS\_RX\_BUSY, после чтения

\* устанавливается статус TWI\_STATUS\_RX\_COMPLETE.

\*

\* \param slaw адрес ведомого устройства SLA+W

\* \param reg\_addr адрес начального регистра для чтения

\* \param buf указатель на буфер для чтения

\* \param size размер буфера для чтения

\*/

void twi\_read(uint8\_t slaw, uint8\_t reg\_addr, uint8\_t \*buf, uint8\_t size)

{

twi\_txrx(slaw, reg\_addr, buf, size);

twi\_status = TWI\_STATUS\_RX\_BUSY;

}

/\* Конечный автомат линии TWI \*/

enum line\_status

{

LINE\_BUS\_ERROR = 0, // Ошибка на шине в результате некорректного формирования

// состояния СТАРТ или СТОП

LINE\_START = 1, // Сформировано состояние СТАРТ

LINE\_REP\_START = 2, // Сформировано состояние ПОВСТАРТ

LINE\_TX\_SLA\_ACK = 3, // Передан пакет SLA+W и принято подтверждение (ACK)

LINE\_TX\_SLA\_NACK = 4, // Передан пакет SLA+W, а подтверждение не принято (NACK)

LINE\_TX\_DATA\_ACK = 5, // Передан пакет данных и принято подтверждение (ACK)

LINE\_TX\_DATA\_NACK = 6, // Передан пакет данных, а подтверждение не принято (NACK)

LINE\_ARB\_LOST = 7, // Потеря приоритета при передаче пакета адреса или данных

LINE\_RX\_SLA\_ACK = 8, // Передан пакет SLA+R и принято подтверждение (ACK)

LINE\_RX\_SLA\_NACK = 9, // Передан пакет SLA+R и принято неподтверждение (NACK)

LINE\_RX\_DATA\_ACK = 10, // Принят байт данных и передано подтверждение (ACK)

LINE\_RX\_DATA\_NACK = 11, // Принят байт данных и передано неподтверждение (NACK)

LINE\_CNT

};

static void line\_error\_stop(void)

{

TWCR |= (1 << TWSTO);

twi\_status = TWI\_STATUS\_ERROR;

}

static void line\_send\_slaw(void)

{

TWDR = twi.slaw;

TWCR &= ~(1 << TWSTA);

}

static void line\_send\_slar(void)

{

uint8\_t twi\_slar = twi.slaw | 0x01;

TWDR = twi\_slar;

TWCR &= ~(1 << TWSTA);

}

static void line\_send\_reg\_addr(void)

{

TWDR = twi.reg\_addr;

}

static void line\_tx\_data(void)

{

switch (twi\_status)

{

case TWI\_STATUS\_RX\_BUSY:

TWCR |= (1 << TWSTA);

break;

case TWI\_STATUS\_TX\_BUSY:

TWDR = \*twi.data++;

if (twi.count-- == 0)

{

TWCR |= (1 << TWSTO);

twi\_status = TWI\_STATUS\_TX\_COMPLETE;

}

break;

default:

break;

}

}

static void line\_rx\_set\_ack(void)

{

if (--twi.count)

{

TWCR |= (1 << TWEA); // ACK

}

else

{

TWCR &= ~(1 << TWEA); // NACK

}

}

static void line\_rx\_data(void)

{

\*twi.data++ = TWDR;

line\_rx\_set\_ack();

}

static void line\_rx\_last\_data(void)

{

\*twi.data++ = TWDR;

TWCR |= (1 << TWSTO);

twi\_status = TWI\_STATUS\_RX\_COMPLETE;

}

typedef void (\*vfptr\_t)(void);

const vfptr\_t line\_actions\_table[LINE\_CNT] PROGMEM = {

[LINE\_BUS\_ERROR] = line\_error\_stop,

[LINE\_START] = line\_send\_slaw,

[LINE\_REP\_START] = line\_send\_slar,

[LINE\_TX\_SLA\_ACK] = line\_send\_reg\_addr,

[LINE\_TX\_SLA\_NACK] = line\_error\_stop,

[LINE\_TX\_DATA\_ACK] = line\_tx\_data,

[LINE\_TX\_DATA\_NACK] = line\_error\_stop,

[LINE\_ARB\_LOST] = line\_error\_stop,

[LINE\_RX\_SLA\_ACK] = line\_rx\_set\_ack,

[LINE\_RX\_SLA\_NACK] = line\_error\_stop,

[LINE\_RX\_DATA\_ACK] = line\_rx\_data,

[LINE\_RX\_DATA\_NACK] = line\_rx\_last\_data,

};

/\* Прерывание используется для обработки события на линии TWI \*/

ISR(TWI\_vect)

{

enum line\_status status = (TWSR & 0xF8) >> 3;

vfptr\_t func = (vfptr\_t)pgm\_read\_word(&line\_actions\_table[status]);

if (func != 0)

{

func();

}

else

{

TWCR |= (1 << TWSTO);

}

TWCR |= (1 << TWINT);

}

/\* End File \*/

Файл sevseg.h

#ifndef SEVSEG\_H

#define SEVSEG\_H

#include <avr/io.h>

void sevseg\_init(void);

void sevseg\_bin2bcd(uint16\_t number, uint8\_t \*digits);

void sevseg\_display\_process(const volatile uint8\_t \*digits);

void sevseg\_off(void);

void sevseg\_set\_dot(uint8\_t position);

void sevseg\_blank\_leading(uint8\_t on);

#endif

/\* End File \*/

Файл secseg.c

#include <avr/pgmspace.h>

#include "sevseg.h"

#define DATA\_PORT PORTA // Порт данных к которому подключены

#define DATA\_DDR DDRA // семисегментные индикаторы.

#define CTRL\_PORT PORTB // Управляющий порт к которому подключены общие

#define CTRL\_DDR DDRB // катоды/аноды семисегментных индикаторов

#define CTRL\_PIN3 3 // Номер вывода общего а/к четвертого индикатора

#define CTRL\_PIN2 2 // Номер вывода общего а/к третьего индикатора

#define CTRL\_PIN1 1 // Номер вывода общего а/к второго индикатора

#define CTRL\_PIN0 0 // Номер вывода общего а/к первого индикатора

#define CTRL\_PIN\_MASK \

((1 << CTRL\_PIN3) | (1 << CTRL\_PIN2) | (1 << CTRL\_PIN1) | (1 << CTRL\_PIN0))

static uint8\_t dot\_pos = 0xFF; /\* по умолчанию точка выключена \*/

static uint8\_t blank\_leading = 0; /\* гасить ли ведущие нули \*/

#define SEVSEG\_MINUS\_CODE 0b00000010

void sevseg\_set\_dot(uint8\_t position) { dot\_pos = position; }

void sevseg\_blank\_leading(uint8\_t on) { blank\_leading = on; }

const uint8\_t ctrl\_pins[4] PROGMEM = {

(1 << CTRL\_PIN0),

(1 << CTRL\_PIN1),

(1 << CTRL\_PIN2),

(1 << CTRL\_PIN3),

};

/\* Таблица перекодировки символов в памяти программ \*/

const uint8\_t dec2sevseg[10] PROGMEM = {

0b11111100, // код "0"

0b01100000, // код "1"

0b11011010, // код "2"

0b11110010, // код "3"

0b01100110, // код "4"

0b10110110, // код "5"

0b10111110, // код "6"

0b11100000, // код "7"

0b11111110, // код "8"

0b11110110 // код "9"

};

/\*\*

\* Функция инициализации 4-х семисегментных индикаторов

\* Выводы данных подключены к DATA\_PORT

\* Выводы управления подключены к CTRL\_PORT

\*/

void sevseg\_init(void) {

/\* Выводы данных и управления - выходы \*/

DATA\_DDR = 0xFF;

DATA\_PORT = 0x00;

CTRL\_DDR |= CTRL\_PIN\_MASK;

CTRL\_PORT &= ~CTRL\_PIN\_MASK;

// TODO проверить нужна ли данная строчка

CTRL\_PORT |= (1 << CTRL\_PIN0);

}

/\*\*

\* Функция перевода 2-х байтного числа в 4-х разрядное двоично-десятичное число.

\*

\* \param value исходное 2-х байтное число

\* \param digits указатель на 4-х байтный буфер, в который записываются разряды

\* двоично-десятичного числа.

\*/

void sevseg\_bin2bcd(uint16\_t value, uint8\_t \*digits) {

/\* Установка указателя на последний (старший) разряд \*/

digits += 4 - 1;

/\* Вычисление разрядов числа, начиная со старшего \*/

\*digits-- = value / 1000;

value %= 1000;

\*digits-- = value / 100;

value %= 100;

\*digits-- = value / 10;

value %= 10;

\*digits-- = value / 1;

value %= 1; // ?

}

/\*\*

\* Функция циклически инкрементирует номер отображаемого разряда

\* 4-х разрядного числа

\*

\* \return номер следующего отображаемого разряда

\*/

static uint8\_t inc\_number\_digit(void) {

static uint8\_t num\_dig = 0;

num\_dig = (num\_dig + 1) & 0x03;

return num\_dig;

}

/\*\*

\* Функция отображения разряда 4-х разрядного числа на семисегментном

\* индикаторе. Для преобразования разряда числа в код семисегментного индикатора

\* используется таблица перекодировки символов dec2sevseg в памяти программ.

\*

\* \param digits указатель на 4-х байтное двоично-десятичное число

\* \param num\_dig номер отображаемого разряда

\*/

static void display\_digit(const volatile uint8\_t \*digits, uint8\_t num\_dig) {

DATA\_PORT = 0x00;

CTRL\_PORT &= ~CTRL\_PIN\_MASK;

uint8\_t d = digits[num\_dig];

/\* гасим лидирующие нули, кроме младшего разряда \*/

if (blank\_leading && num\_dig > 0) {

uint8\_t higher\_nonzero = 0;

for (uint8\_t i = 3; i > num\_dig; --i)

if (digits[i]) higher\_nonzero = 1;

if (!higher\_nonzero && d == 0) { /\* просто не включаем индикатор \*/

return;

}

}

uint8\_t seg;

if (d <= 9) { /\* цифра \*/

seg = pgm\_read\_byte(&dec2sevseg[d]);

} else if (d == 0x0A) { /\* «минус» \*/

seg = SEVSEG\_MINUS\_CODE;

} else { /\* пусто \*/

return;

}

if (num\_dig == dot\_pos) seg |= 0x01; /\* точка \*/

DATA\_PORT = seg;

CTRL\_PORT |= pgm\_read\_byte(&ctrl\_pins[num\_dig]);

}

/\*\*

\* Функция отображения 4-х разрядного двоично-десятичного числа на

\* 4-х семисегментных индикаторах.

\* При каждом вызове отображается один разряд, поэтому для отображения всего

\* числа, функцию необходимо переодически вызывать.

\*

\* \param digits указатель на 4-х байтное двоично-десятичное число

\*/

void sevseg\_display\_process(const volatile uint8\_t \*digits) {

uint8\_t num\_dig = inc\_number\_digit();

display\_digit(digits, num\_dig);

}

/\*\*

\* Функция выключения семисегментных индикаторов

\*/

void sevseg\_off(void) { DATA\_PORT = 0x00; }

/\* End File \*/

Файл spi.h

/\* ----------------------------- spi.h ------------------------------ \*/

#ifndef SPI\_H

#define SPI\_H

#include <avr/io.h>

void spi\_init(void);

uint8\_t spi\_transfer(uint8\_t data);

#endif

//

Файл spi.c

/\* ----------------------------- spi.c ------------------------------ \*/

#include "spi.h"

#include <avr/io.h>

void spi\_init(void)

{

/\* MOSI – PB5, MISO – PB6, SCK – PB7 \*/

DDRB |= (1<<PB5)|(1<<PB7); /\* outputs \*/

DDRB &= ~(1<<PB6); /\* MISO input \*/

SPCR = (1<<SPE)|(1<<MSTR)|(1<<CPOL)|(1<<CPHA)|(1<<SPR0); /\* fosc/16, mode 1 \*/

}

uint8\_t spi\_transfer(uint8\_t data)

{

SPDR = data;

while(!(SPSR & (1<<SPIF)));

return SPDR;

}

//