Подробный алгоритм функции READ_DATA()

Назначение функции

Функция `READ_DATA()` выполняет чтение и обработку состояния всех входных сигналов системы, включая:

- Состояние измерительных каналов IK
- Режим работы (дистанционное управление DU)
- Состояние переключателей питания
- Кнопки управления аттенюатором
- Кнопки управления реле
- Логические состояния системы

Карта регистров чтения

RD0 - Состояние переключателей питания

```
Бит 0: FPIT_CKP  – питание контрольно-корректирующего пункта
Бит 1: FPIT_PDP  – питание измерительного блока
Бит 2: F6_7_PDP  – питание блока 6.7 PDP
Биты 3–7: COD_OTV – код ответвителя (младшие 5 бит)
```

RD1 - Кнопки и управление

```
Бит 0: COD_OTV[5] – код ответвителя (бит 5) = 0x20
Бит 1: COD_OTV[6] – код ответвителя (бит 6) = 0x40
Бит 2: COD_OTV[7] – код ответвителя (бит 7) = 0x80
Бит 3: COD_OTV[8] – код ответвителя (бит 8) = 0xFF
Бит 4: FS_ATT_UP – кнопка "Аттенюатор ВВЕРХ"
Бит 5: FS_ATT_DOWN – кнопка "Аттенюатор ВНИЗ" (инверсная логика)
Бит 6: Логический переключатель (для детектирования изменений)
Бит 7: FP1_P2 – состояние питания P1/P2
```

RD2 - Режим дистанционного управления

```
Бит 0: F_DU – режим дистанционного управления (0=локально, 1=дистанционно)
```

RD3 - Состояние измерительных каналов

```
Биты 0–7: Состояние каналов IK (инверсная логика)
```

RD4 - Кнопки реле и системные сигналы

```
Бит 3: F_KVP_LOG – логический сигнал KVP (инверсная логика)
Бит 4: FS_OTB – кнопка "Отбой" (инверсная логика)
Бит 5: FS_BKL – кнопка "Включить реле" (инверсная логика)
Бит 6: FS_OTKL – кнопка "Отключить реле" (инверсная логика)
Бит 7: FS_TEST – кнопка "Тест/Диагностика"
```

Пошаговый алгоритм выполнения

Шаг 1: Чтение измерительных каналов IK

```
buffer = perform_actionR(RD3);  // Чтение регистра RD3
var.IK1_2 |= ~buffer;           // Инверсия и накопление состояния каналов
```

Назначение: Определение активных измерительных каналов. Используется инверсная логика (0 = канал активен).

Шаг 2: Определение режима работы

```
buffer = perform_actionR(RD2);                  // Чтение регистра режима
flags.F_DU = (buffer & (1 << 0)) ? 1 : 0;       // Бит 0 = режим DU
```

Назначение: Определение режима работы системы:

- `flags.F_DU = 0` - локальное управление
- `flags.F_DU = 1` - дистанционное управление от ТАКТ_52

Шаг 3: Чтение переключателей питания и управления

```
buffer = perform_actionR(RD0);   // Переключатели питания и код ответвителя
buffer1 = perform_actionR(RD1);  // Кнопки и дополнительные биты кода
```

Шаг 4: Обработка переключателей питания (только в локальном режиме)

```
if (flags.F_DU == 0)   // Только если НЕ дистанционное управление
{
    // Чтение состояния питания узлов
    flags.FPIT_CKP = (buffer & (1 << 0)) ? 1 : 0;    // Питание СКР
    flags.FPIT_PDP = (buffer & (1 << 1)) ? 1 : 0;    // Питание PDP
    flags.F6_7_PDP = (buffer & (1 << 2)) ? 1 : 0;    // Питание 6.7 PDP

    // Управление остановкой измерений
    StopM = (flags.FPIT_PDP == 1) ? 0 : 1;  // Если PDP выключен – остановить

    // Если измерения остановлены – обнулить индикаторы
    if(StopM)
    {
        INDICATE_U(0);  // Обнуление индикации напряжения
        INDICATE_I(0);  // Обнуление индикации тока
    }
}
```

Шаг 5: Формирование кода ответвителя (только в локальном режиме)

```
if (flags.F_DU == 0)
{
    buffer >>= 3;                        // Сдвиг для получения битов 3–7
    if (buffer & 0x1F) COD_OTV = buffer; // Биты 3–7 из RD0 (маска 0x1F = 5 бит)

    // Дополнительные биты кода из RD1
    if (buffer1 & (1 << 0)) COD_OTV = 0x20;  // Бит 0 RD1 -> код 0x20
    if (buffer1 & (1 << 1)) COD_OTV = 0x40;  // Бит 1 RD1 -> код 0x40
    if (buffer1 & (1 << 2)) COD_OTV = 0x80;  // Бит 2 RD1 -> код 0x80
    if (buffer1 & (1 << 3)) COD_OTV = 0xFF;  // Бит 3 RD1 -> код 0xFF
}
```

Коды ответвителя: Значения от 0x01 до 0xFF, определяющие конфигурацию переключателя-разветвителя.

Шаг 6: Повторное чтение RD1 для кнопок

```
buffer1 = perform_actionR(RD1);  // Повторное чтение для актуального состояния кнопок
```

Шаг 7: Обработка кнопок аттенюатора

```
if ((flags.F_DU == 0) && (flags.FS_ATT_UPDOWN == 0))  // Локальный режим + не зажаты обе кнопки
{
    flags.FS_ATT_UP   = (buffer1 & (1 << 4)) ? 1 : 0;  // Кнопка "Вверх"
    flags.FS_ATT_DOWN = (buffer1 & (1 << 5)) ? 0 : 1;  // Кнопка "Вниз" (инверсная логика!)
}
```

Особенность: Кнопка "Вниз" использует инверсную логику (0 = нажата, 1 = отпущена).

Шаг 8: Детектирование изменений логического переключателя

```
if((flags.F_DU == 0) && (Nlogical == 0))  // Первое чтение
{
    F_N0 = (buffer1 & (1 << 6)) ? 1 : 0;  // Сохранение текущего состояния
    Nlogical = 1;                          // Переключение на следующее чтение
}
else                                       // Второе чтение
{
    F_N1 = (buffer1 & (1 << 6)) ? 1 : 0;  // Сохранение нового состояния
    Nlogical = 0;                          // Возврат к первому чтению
}

// Детектирование изменения состояния
if((flags.F_DU == 0) && (F_N0 != F_N1))
    F_N = 1;  // Флаг изменения для обновления индикации
```

Назначение: Определение переключения режима индикации (обычный/жгут 673).

Шаг 9: Чтение состояния питания P1/P2

```
if (flags.F_DU == 0)
    flags.FP1_P2 = (buffer1 & (1 << 7)) ? 1 : 0;  // Состояние питания P1/P2
```

Шаг 10: Чтение системных кнопок и сигналов

```
if (flags.F_DU == 0)  // Только в локальном режиме
{
    buffer1 = perform_actionR(RD4);  // Чтение регистра RD4

    flags.F_KVP_LOG = (buffer1 & (1 << 3)) ? 0 : 1;  // Логический KVP (инверсная)
    flags.FS_OTB    = (buffer1 & (1 << 4)) ? 0 : 1;  // Кнопка "Отбой" (инверсная)
    flags.FS_TEST   = (buffer1 & (1 << 7)) ? 1 : 0;  // Кнопка "Тест"

    // flags.FS_TEST = 0;  // Закомментированное принудительное отключение теста
}
```

Логика работы с инверсными сигналами

Прямая логика (1 = активно, 0 = неактивно):

- `flags.FPIT_CKP`, `flags.FPIT_PDP`, `flags.F6_7_PDP`
- `flags.FS_ATT_UP`, `flags.FP1_P2`
- `flags.FS_TEST`

Инверсная логика (0 = активно, 1 = неактивно):

- `flags.FS_ATT_DOWN`
- `flags.F_KVP_LOG`
- `flags.FS_OTB`
- `var.IK1_2` (каналы IK)

Условия выполнения операций

Операции только в локальном режиме (`flags.F_DU == 0`):

1. Чтение переключателей питания
2. Формирование кода ответвителя
3. Обработка кнопок аттенюатора
4. Детектирование изменений логического переключателя
5. Чтение состояния P1/P2
6. Чтение системных кнопок

Операции выполняемые всегда:

1. Чтение каналов IK
2. Определение режима DU

Связь с другими функциями

Входные данные:

- Аппаратные регистры RD0-RD4
- Глобальные переменные: `Nlogical`, `F_N0`, `F_N1`

Выходные данные:

- Флаги состояния системы (`flags.*`)
- Переменная остановки измерений (`StopM`)
- Код ответвителя (`COD_OTV`)
- Состояние каналов (`var.IK1_2`)
- Флаги изменений (`F_N`)

Вызываемые функции:

- `perform_actionR()` - чтение аппаратных регистров
- `INDICATE_U()`, `INDICATE_I()` - обнуление индикации при остановке

Частота вызова

Функция вызывается каждые **50 мс** из обработчика таймера `TIME()` в составе периодических задач системы.

Критические аспекты

1. **Приоритет дистанционного управления**: При `flags.F_DU = 1` большинство локальных входов игнорируются
2. **Автоматическая остановка**: При отключении питания PDP автоматически останавливаются измерения
3. **Инверсная логика**: Многие сигналы используют инверсную логику, что требует внимательной обработки
4. **Детектирование изменений**: Специальный алгоритм для определения переключения режимов индикации

Функция является критически важной для корректной работы всей системы, обеспечивая связь между аппаратными входами и программной логикой управления.