Подробный алгоритм функции WRITE_DATA()

Назначение функции

Функция `WRITE_DATA()` выполняет запись управляющих данных в выходные регистры системы на основе текущего состояния флагов и переменных. Она преобразует программные состояния в аппаратные сигналы управления.

Карта выходных регистров

WR2 - Основные управляющие сигналы

```
Бит 0: PITPDP     - управление питанием PDP
Бит 1: PITCKP     - управление питанием CKP
Бит 2: PITPDP_6   - управление питанием PDP 6.2/7.8
Бит 5: F_BKL      - импульс включения реле (инверсия)
Бит 6: F_OTKL     - импульс отключения реле (инверсия)
Бит 7: I_PITPDP   - индикация питания PDP (инверсия)
```

WR3 - Состояние каналов и режимов

```
Биты 0-3: IK1_2   - состояние измерительных каналов (старшие 4 бита)
Бит 4: FI_DU      - индикация режима дистанционного управления
Бит 5: FP1_P2     - управление питанием P1/P2
```

WR4 - Состояние измерительных каналов

```
Биты 0-3: IK1_2   - состояние измерительных каналов (младшие 4 бита)
```

WR5 - Индикация и управление CKP (только в локальном режиме)

```
Бит 0: FISX       - индикация исходного состояния
Бит 1: FI_BKL     - индикация включенного реле
Бит 2: FI_OTKL    - индикация отключенного реле
Бит 3: FI_TEST    - индикация режима тестирования
Биты 4-7: CKP_CODE - код управления контрольно-корректирующим пунктом
```

WR6 - Значение аттенюации (только в локальном режиме)

```
Биты 0-7: ATT     - текущее значение аттенюатора
```

WR7 - Код ответвителя (только в локальном режиме)

```
Биты 0-7: COD_OTV - код конфигурации переключателя-ответвителя
```

Пошаговый алгоритм выполнения

Шаг 1: Инициализация базового буфера WR2

```
unsigned char buffer = 0x60;  // Базовое значение: 0110 0000
                              // Биты 5,6 = 1 (реле в покое, инверсная логика)
```

Шаг 2: Формирование данных для WR2 (управляющие сигналы)

2.1 Проверка блокировки записи

```
if ((flags.F_START1 || flags.F_START2) == 0)  // Если НЕ активны команды реле
{
    // Продолжить формирование WR2
}
```

Назначение: Предотвратить запись в WR2 во время выполнения импульсов управления реле.

2.2 Формирование битов питания

```
// Управление питанием CKP (бит 1)
UPDATE_BUFFER(buffer, flags.FPIT_CKP != 1, PITCKP_MASK, PITCKP_OFF_MASK);
// Если НЕ включено - установить бит 1, иначе - сбросить

// Управление питанием PDP (бит 0)
```

```
UPDATE_BUFFER(buffer, flags.FPIT_PDP != 1, PITPDP_MASK, PITPDP_OFF_MASK);
// Если НЕ включено – установить бит 0, иначе – сбросить
```

Логика работы макроса UPDATE_BUFFER:

```
#define UPDATE_BUFFER(buffer, condition, set_mask, clear_mask) \
        buffer = (condition) ? (buffer | set_mask) : (buffer & clear_mask)

// Если условие истинно – устанавливает биты (OR с set_mask)
// Если условие ложно – сбрасывает биты (AND с clear_mask)
```

2.3 Управление питанием блока 6.7 PDP

```
UPDATE_BUFFER(buffer, flags.F6_7_PDP == 0, PITPDP_6_MASK, PITPDP_7_MASK);
// Если блок выключен – установить бит 2 (режим 6.2)
// Если блок включен – сбросить бит 2 (режим 7.8)
```

2.4 Индикация питания PDP (инверсная логика)

```
UPDATE_BUFFER_AND_OR(buffer, flags.FPIT_PDP != 0, I_PITPDP_MASK, I_PITPDP_OFF_MASK);
// Если PDP включен – сбросить бит 7 (AND с 0x7F)
// Если PDP выключен – установить бит 7 (OR с 0x80)
```

2.5 Запись в регистр WR2

```
perform_actionW(WR2, buffer);  // Отправка данных в аппаратный регистр
```

Шаг 3: Формирование данных для WR3 (каналы и режимы)

3.1 Подготовка данных измерительных каналов

```
buffer = ~var.IK1_2;   // Инверсия состояния каналов IK
buffer = buffer >> 4;  // Сдвиг вправо на 4 бита (старшие биты в младшие)
```

Назначение: Преобразование 8-битного состояния каналов IK в 4-битное для записи в старшие биты WR3.

3.2 Добавление управляющих битов

```
// Управление питанием P1/P2 (бит 5)
UPDATE_BUFFER(buffer, flags.FP1_P2 == 0, FP1_P2_MASK, FP1_P2_2_MASK);
// Если P1/P2 выключен – установить бит 5, иначе – сбросить

// Индикация режима DU (бит 4)
UPDATE_BUFFER(buffer, flags.F_DU == 0, FI_DU_MASK, FI_DU_OFF_MASK);
// Если НЕ дистанционное управление – установить бит 4, иначе – сбросить
```

3.3 Запись в регистр WR3

```
perform_actionW(WR3, buffer);
```

Шаг 4: Формирование данных для WR4 (младшие биты каналов IK)

```
buffer = ~var.IK1_2 & IK_MASK;   // Инверсия + маскирование младших 4 бит (0x0F)
perform_actionW(WR4, buffer);
```

Назначение: Вывод полного 8-битного состояния каналов IK через два 4-битных регистра (WR3[3:0] и WR4[3:0]).

Шаг 5: Запись локальных управляющих данных (только при flags.F_DU == 0)

```
if (flags.F_DU == 0)  // Только в режиме локального управления
{
    update_WR5();                   // Обновление регистра индикации и СКР
    perform_actionW(WR6, ATT);      // Запись значения аттенюатора
    perform_actionW(WR7, COD_OTV);  // Запись кода ответвителя
}
```

Детальный алгоритм функции update_WR5()

Назначение

Формирование регистра WR5, содержащего биты индикации состояния системы и код управления СКР.

Пошаговое выполнение

Шаг 1: Инициализация буфера

```
unsigned char buffer = 0x00;  // Начальное значение
```

Шаг 2: Формирование битов индикации

```
// Индикация исходного состояния (бит 0)
UPDATE_BUFFER(buffer, flags.FISX == 0, FISX_MASK, FISX_OFF_MASK);
// Если НЕ исходное состояние - установить бит 0

// Индикация включенного реле (бит 1)
UPDATE_BUFFER(buffer, flags.FI_BKL == 0, FI_BKL_MASK, FI_BKL_OFF_MASK);
// Если НЕ показывать включение - установить бит 1

// Индикация отключенного реле (бит 2)
UPDATE_BUFFER(buffer, flags.FI_OTKL == 0, FI_OTKL_MASK, FI_OTKL_OFF_MASK);
// Если НЕ показывать отключение - установить бит 2

// Индикация тестирования (бит 3)
UPDATE_BUFFER(buffer, flags.FI_TEST == 0, FI_TEST_MASK, FI_TEST_OFF_MASK);
// Если НЕ режим теста - установить бит 3
```

Шаг 3: Добавление кода СКР

```
buffer |= CKP_CODE;  // Побитовое ИЛИ с кодом управления СКР (биты 4-7)
```

Шаг 4: Запись в регистр

```
perform_actionW(WR5, buffer);
```

Логика инверсных сигналов

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

- Биты питания в WR2 (при выключенном состоянии флагов)
- Биты индикации в WR5 (при выключенном состоянии флагов)
- Состояние каналов ІК

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

- Исходные флаги питания (flags.FPIT_*)
- Флаги индикации (flags.FI_*)
- Индикация питания PDP (бит 7 WR2)

Условия выполнения операций

Операции выполняемые всегда:

1. Формирование WR2 (при отсутствии активных команд реле)
2. Формирование WR3 (состояние каналов и режимов)
3. Формирование WR4 (младшие биты каналов)

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

1. Обновление WR5 (индикация и СКР)
2. Запись WR6 (аттенюатор)
3. Запись WR7 (код ответвителя)

Операции с блокировкой:

1. Запись WR2 блокируется при активных flags.F_START1 или flags.F_START2

Связь с другими функциями

Входные данные:

- Флаги состояния системы (flags.*)
- Состояние измерительных каналов (var.IK1_2)
- Код управления СКР (CKP_CODE)
- Значение аттенюатора (ATT)
- Код ответвителя (COD_OTV)

Выходные данные:

- Аппаратные регистры WR2-WR7
- Управляющие сигналы для внешних устройств

Вызываемые функции:

- `perform_actionW()` - запись в аппаратные регистры
- `update_WR5()` - формирование регистра индикации

Частота вызова

Функция вызывается каждые **100 мс** из обработчика времени `TIME()` в составе основного цикла управления.

Критические аспекты

1. **Приоритет дистанционного управления**: В режиме DU локальные регистры WR5-WR7 не обновляются
2. **Защита от конфликтов**: Запись WR2 блокируется во время выполнения команд управления реле
3. **Инверсная логика**: Многие сигналы требуют инверсии при записи в аппаратуру
4. **Синхронизация**: Все данные записываются атомарно в течение одного цикла
5. **Разделение каналов IK**: 8-битное состояние разделяется между WR3 и WR4 для совместимости с аппаратурой

Временные характеристики

- **Время выполнения**: ~7 операций записи через SPI (~28 мс при 4 мс на операцию)
- **Критичность**: Средняя - влияет на управление аппаратурой
- **Синхронизация**: Должна выполняться после READ_DATA() и обработки логики

Функция обеспечивает преобразование программных состояний в аппаратные управляющие сигналы, являясь ключевым элементом выходного интерфейса системы управления.