Навигация

Глава VI. Аналого-цифровой преобразователь 2
6.1. Понятие аналого-цифрового преобразователя 2
6.2. Метод преобразования последовательным приближением 3
6.3. Программирование аналого-цифрового преобразователя в микроконтроллерах серии 1986BE9x 5
6.3.1. Инициализация аналого-цифрового преобразователя 6
6.3.2. Однократное преобразование 12
6.3.3. Циклическое преобразование 13
6.3.4. Последовательное переключение каналов 16
6.4. Термисторы 19
6.5. Двухосевые джойстики 21
Описание программных проектов 25
Задачи для самостоятельной работы 26
Контрольные вопросы 26

Глава VI

Аналого-цифровой преобразователь

Цель работы:

– изучение алгоритмов аналого-цифрового преобразования;

– получение навыков работы с аналоговыми устройствами.

Оборудование:

– отладочный комплект для микроконтроллера 1986ВЕ92У;

– программатор-отладчик J-LINK (или аналог);

– модуль расширения для работы с микроконтроллерной техникой;

– цифровой мультиметр;

– персональный компьютер.

Программное обеспечение:

– операционная система Windows 7 / 8 / 10;

– среда программирования Keil µVision MDK-ARM 5.24;

– драйвер программатора J-LINK;

– примеры кода программ.

6.1. Понятие аналого-цифрового преобразователя

Аналого-цифровой преобразователь (**АЦП**, англ. Analog-To-Digital Converter, **ADC**) представляет собой устройство, преобразующее физическую величину в ее числовое представление. Формально, входным сигналом АЦП может быть любая физическая величина – сила тока, напряжение, сопротивление, емкость, частота. Однако в подавляющем большинстве случаев под аналого-цифровым преобразованием подразумевается преобразование **напряжение-код**.

Аналого-цифровое преобразование тесно связано с понятием измерения. Под измерением понимается процесс сравнения измеряемой величины с некоторым эталоном; при аналого-цифровом преобразовании происходит сравнение входного значения с **опорным напряжением**. Таким образом, к преобразованию применимы все понятия метрологии, в частности – погрешности измерений [x].

АЦП является основой многих измерительных приборов: цифровых вольтметров и термометров, электронных весов, датчиков давления и т.д. На базе АЦП также создается цифровая звукозаписывающая аппаратура.

АЦП может быть исполнен в виде отдельной микросхемы (как, например, микросхемы 5101НВ015, 1310НМ025, 1316ПП1У разработки и производства компании «Миландр»), или являться частью какого-либо устройства, как в случае микросхемами серии 1986ВЕ9Х.

Основными характеристиками АЦП являются частота преобразования, разрядность и допустимый диапазон опорных напряжений.

Частота преобразования определяет количество преобразований, которое АЦП способен выполнить в течение секунды. Она зависит от тактовой частоты и алгоритма преобразования.

Разрядность характеризует точность АЦП в смысле округления значения. Например, 10-разрядный АЦП способен различать $2^{10} = 1024$ уровня напряжения, а 12-разрядный – $2^{12} = 4096$.

Надо отметить, что реализация АЦП с высоким быстродействием и большой разрядностью – задача достаточно сложная. Обычно акцент делается на одном параметре, а другой организуется по остаточному принципу.

Опорное напряжение (англ. Reference Voltage) является эталонной величиной, с которой сравнивается входной сигнал. Чем точнее и стабильнее источник опорного напряжения, тем достовернее будет результат аналого-цифрового преобразования. Значение опорного напряжения может варьироваться в зависимости от схемы подключения АЦП. Однако внутренние каскады преобразователей характеризуются предельно допустимыми значениями токов, протекающих через них. Эта характеристика задает допустимый диапазон опорных напряжений. Из этого можно заключить, что **входной сигнал никогда не должен превышать опорного напряжения**. Превышение в пределах допустимого диапазона приведет к неверному результату преобразования; за пределами – к выходу из строя микросхемы.

Ключевым элементом большинства АЦП является **компаратор**. Компаратор – это устройство сравнения аналоговых сигналов. Он имеет два входа – прямой и инверсный – и один выход (рисунок 6.1). Если напряжение на прямом входе больше, чем на инверсном, то на выходе компаратора формируется сигнал высокого уровня; если наоборот – низкого.

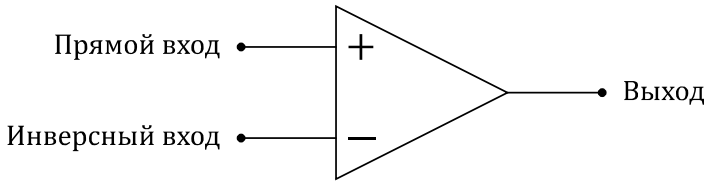

Рисунок 6.1 – Условное графическое обозначение компаратора

Существует достаточно много методов аналого-цифрового преобразования: параллельное преобразование, сигма-дельта преобразование, конвейерное преобразование, комбинирование варианты и т.д. В целом, это процесс довольно творческий. Но чтобы не запутаться в большом объеме информации, рассмотрим наиболее распространенный вариант: **последовательное приближение**. Такой же метод реализован в преобразователях микроконтроллеров серии 1986ВЕ9х.

6.2. Метод преобразования последовательным приближением

В начале XIII века Леонардо Пизанский (также известный как **Фибоначчи**) сформулировал задачу о выборе оптимального набора гирь для взвешивания на рычажных весах. Суть задачи состояла в нахождении наименьшего числа гирь, которые бы позволили определить все веса меньше заданного.

В случае, когда гири разрешается класть лишь на свободную от груза чашу весов, а инерционностью весов можно пренебречь, решение задачи известно: оптимальной является двоичная система гирь $\{1, 2, 4, 8, ..., 2^{n-1}\}$.

Рассмотрим процедуру взвешивания неизвестного груза с помощью системы двоичных гирь. На первом шаге взвешивания следует положить на свободную (правую) чашу весов старшую гирю 2^{n-1}. После этого действия возможны две ситуации:

1) весы остаются в исходном состоянии;
2) весы переходят в противоположное состояние.

Второй шаг будет зависеть от результата первого. В первом случае необходимо добавить на правую чашу весов следующую по старшинству гирю 2^{n-2}; во втором – снять с правой чаши весов гирю 2^{n-1}, и после возвращения весов в исходное состояние положить туда следующую по старшинству гирю 2^{n-2}. Второй шаг так же создаст одну из двух ситуаций, которую нужно обработать аналогичным образом. В итоге за n шагов можно определить массу груза с точностью до значения массы младшей гири [x].

Интерес к данной задаче возник в XX веке в связи с появлением цифровой техники и, следовательно, потребностью преобразования аналоговых величин в цифровой формат. Описанный выше алгоритм взвешивания лежит в основе аналого-цифровых преобразователей последовательного приближения. Структурная схема такого устройства приведена на рисунке 6.2.

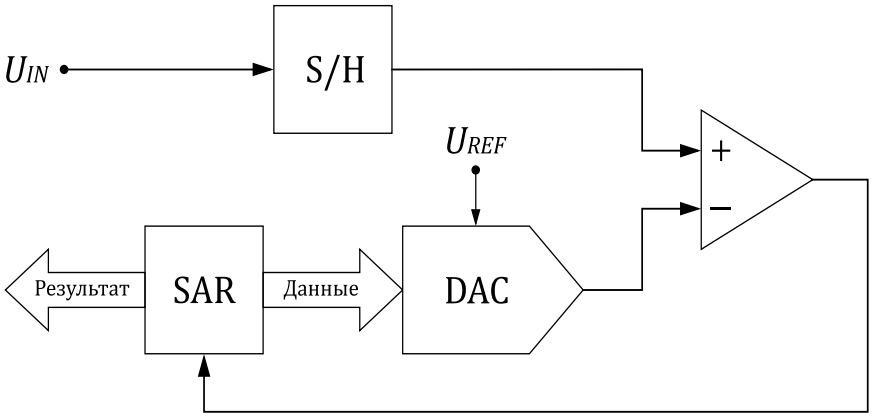

Рисунок 6.2 – Структурная схема преобразователя последовательного приближения

Вся схема состоит из четырех блоков:

1. **Устройство последовательного приближения** (англ. Successive Approximation Register, **SAR**) выполняет описанный выше алгоритм и формирует цифровой сигнал, соответствующий каждой итерации.

2. **Цифро-аналоговый преобразователь** (англ. Digital To Analog Converter, **DAC**) принимает цифрой сигнал от устройства последовательного приближения и преобразует его в аналоговый вид, создавая весовое значение напряжения.

3. **Компаратор** сравнивает сформированное весовое напряжение с входным напряжением U_{IN}. Если входное напряжение больше весового, то на выходе компаратора устанавливается уровень логической единицы, меньше – логического нуля. Выходное значение компаратора фиксируется в качестве одного бита результата преобразования

и направляется на устройство последовательного приближения, как требует того алгоритм.

4. **Устройство выборки и хранения** (англ. Sample And Hold Circuit, **S/H**) запоминает текущее значение входного напряжения и сохраняет его неизменным на протяжении всего цикла преобразования. Это принципиально важно для работы данного типа АЦП.

Для примера рассмотрим 3-разрядный АЦП с опорным напряжением U_{REF} = 4 В, на вход которого подается сигнал U_{IN} = 2.7 В. Процесс преобразования этой величины в цифровой код будет выглядеть следующим образом:

1. На выходе цифро-аналогового преобразователя формируем весовое напряжение ½ U_{REF} = 2 В. U_{IN} > ½ U_{REF}, поэтому на выходе компаратора имеем единицу.

2. Формируем весовое напряжение ½ U_{REF} + ¼ U_{REF} = ¾ U_{REF} = 3 В. U_{IN} < ¾ U_{REF}, поэтому на выходе компаратора имеем ноль.

3. Формируем весовое напряжение ¾ U_{REF} – ⅛ U_{REF} = ⅝ U_{REF} = 2.5 В. U_{IN} > ⅝ U_{REF}, поэтому на выходе компаратора имеем единицу.

В итоге за три итерации (разрядность АЦП) мы получили цифровой код 101_2, соответствующий уровню входного напряжения.

Схема последовательного приближения обладает **сбалансированными показателями быстродействия и точности**. Время преобразования линейно зависит от разрядности устройства. Точность преобразования определятся разрядностью внутреннего цифро-аналогового преобразователя и может достигать 24 битов.

6.3. Программирование аналого-цифрового преобразователя в микроконтроллерах серии 1986ВЕ9х

В микроконтроллерах серии 1986ВЕ9х реализовано два независимых **12-разрядных** аналого-цифровых преобразователя последовательного приближения.

Каждое преобразование занимает не менее **28 тактов**; быстродействие преобразователя таким образом зависит от тактирующей его частоты. Эта частота не может превышать 14 МГц, что позволяет каждому АЦП выполнять до 500 тысяч преобразований в секунду. Преобразователи могут быть могут быть синхронизированы с некоторой разностью фаз для увеличения итогового быстродействия.

Опорное напряжение преобразователей должно находиться в диапазоне **0 ÷ 3.3 В**. В качестве опорного напряжения может выступать либо питание аналоговой части микроконтроллера, либо внешний источник, подключаемый к специальным выводам `ADC0_REF+` и `ADC1_REF–` (рисунок 6.3). Напряжение внешнего источника обычно надежно стабилизируют для увеличения точности преобразования.

Входным сигналом АЦП может быть напряжение с одного из нескольких внешних каналов (16 для 1986ВЕ91Т; 8 для 1986ВЕ92У; 4 для 1986ВЕ93У), внутреннего датчика температуры или внутреннего источника 1.23 В, подключенных через аналоговую матрицу (рисунок 6.3). Для одновременной работы с несколькими сигналами в контроллере преобразователей предусмотрен режим последовательного переключения каналов.

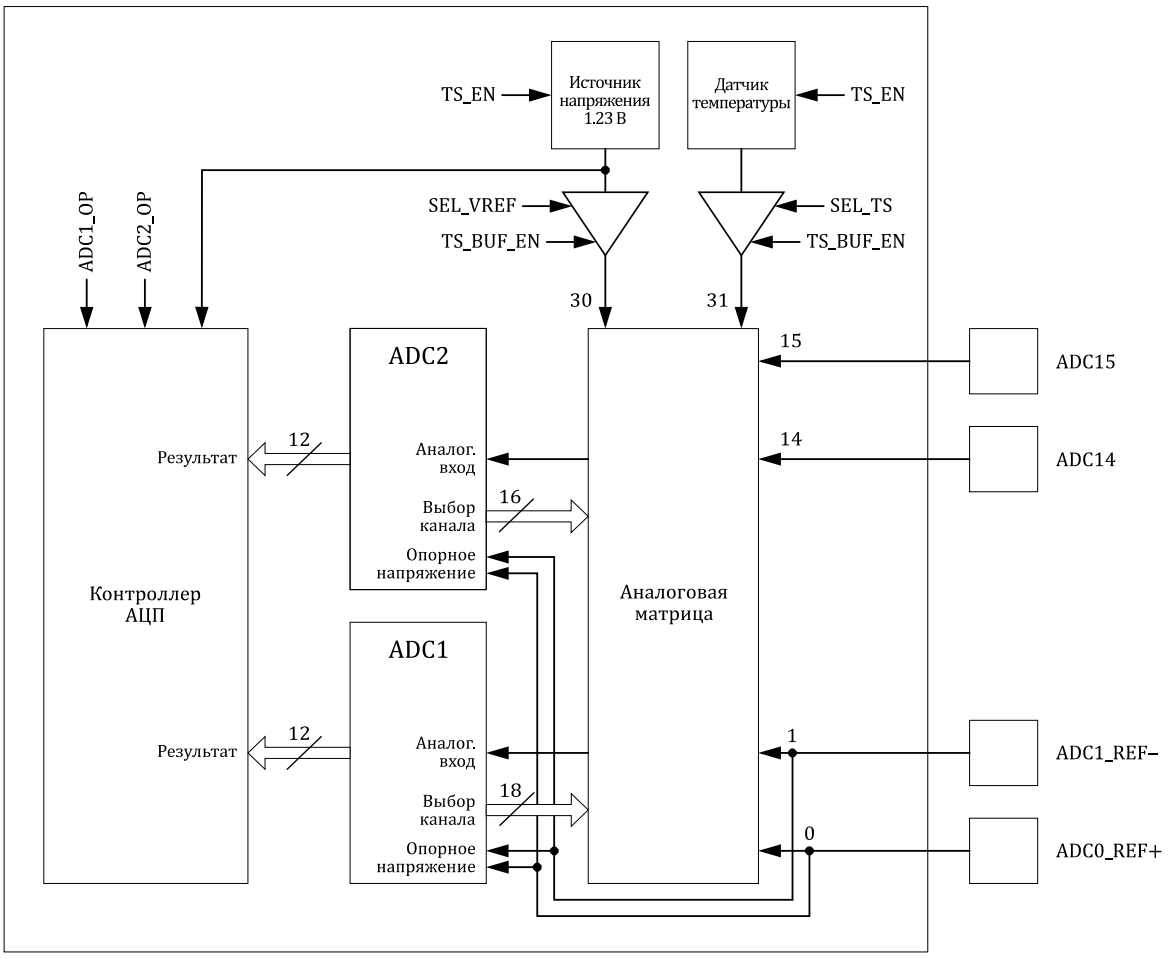

Рисунок 6.3 – Структурная схема контроллера аналого-цифровых преобразователей

6.3.1. Инициализация аналого-цифрового преобразователя

Инициализация контроллера АЦП состоит из следующих шагов:

1. Включение тактирования.
2. Деинициализация.
3. Конфигурация порта ввода-вывода.
4. Конфигурация контроллера.
5. Настройка аппаратных прерываний.

1. Включение тактирования производится в одном регистре для обоих преобразователей сразу:

```
// Включение тактирования АЦП
MDR_RST_CLK->PER_CLOCK |= (1 << RST_CLK_PCLK_ADC_Pos);
```

2. Деинициализация состоит из сброса значений регистров двух АЦП:

```
// Деинициализация первого АЦП
MDR_ADC->ADC1_CFG     = 0;
MDR_ADC->ADC1_H_LEVEL = 0;
MDR_ADC->ADC1_L_LEVEL = 0;
MDR_ADC->ADC1_RESULT;
MDR_ADC->ADC1_STATUS  = 0;
MDR_ADC->ADC1_CHSEL   = 0;
```

```
// Деинициализация второго АЦП
MDR_ADC->ADC2_CFG     = 0;
MDR_ADC->ADC2_H_LEVEL = 0;
MDR_ADC->ADC2_L_LEVEL = 0;
MDR_ADC->ADC2_RESULT;
MDR_ADC->ADC2_STATUS  = 0;
MDR_ADC->ADC2_CHSEL   = 0;
```

Обратите внимание на регистры результата ADCx_RESULT: их чтение осуществляется для сброса флага завершения преобразования. Запись в данный регистр запрещена.

3. Конфигурация порта ввода-вывода. Для корректного выполнения аналого-цифрового преобразования необходимо настроить линии, используемые преобразователем, на аналоговый сигнал и отключить внутренние подтяжки. Остальные настройки порта не оказывают влияния, поэтому им следует задать исходное значение. Например, если предполагается работа с пятым каналом преобразователя (*PD*5), то процедура конфигурации порта должна выглядеть следующим образом:

```
// Конфигурация порта для работы АЦП
void ADC_PortCfg(void)
{
  // Включение тактирования требуемых порта D
  MDR_RST_CLK->PER_CLOCK |= (1 << RST_CLK_PCLK_PORTD_Pos);

  // Конфигурация линии PD5
  MDR_PORTD->OE     &= ~(1 << 5);
  MDR_PORTD->FUNC   &= ~(3 << 10);
  MDR_PORTD->ANALOG &= ~(1 << 5);   // Режим работы линии (аналоговый)
  MDR_PORTD->PULL   &= ~(1 << 5);   // Резистор подтяжки к цепи питания (отключен)
  MDR_PORTD->PULL   &= ~(1 << 21);  // Резистор подтяжки к земле (отключен)
  MDR_PORTD->PD     &= ~(1 << 5);
  MDR_PORTD->PD     &= ~(1 << 21);
  MDR_PORTD->PWR    &= ~(3 << 10);
  MDR_PORTD->GFEN   &= ~(1 << 5);
}
```

4. Конфигурация контроллера преобразователей производится в регистрах ADC1_CFG и ADC2_CFG. Регистры несколько отличаются друг от друга битами 16…24 и 28…31. Описание регистра конфигурации первого АЦП приведено в таблице 6.1.

Таблица 6.1 – Описание регистра MDR_ADC->ADC1_CFG

№ битов	Функциональное имя битов	Описание
31…28	DELAY_ADC[3:0]	Разность фаз между циклами преобразователей: 0000 – 1 такт *HCLK*; 0001 – 2 такта *HCLK*; … 1111 – 16 тактов *HCLK*.
27…25	DELAY_GO[2:0]	Дополнительная задержка перед преобразованием для заряда внутренней емкости: 000 – 1 такт *HCLK*; 001 – 2 такта *HCLK*; … 111 – 8 тактов *HCLK*.

Продолжение таблицы 6.1

№ битов	Функциональное имя битов	Описание
24...21	TR[3:0]	Подстройка внутреннего источника напряжения 1.23 В (рисунок 6.4).
20	SEL_VREF	Преобразование сигнала с внутреннего источника напряжения 1.23 В: 0 – запрещено; 1 – разрешено.
19	SEL_TS	Преобразование сигнала с датчика температуры: 0 – запрещено; 1 – разрешено.
18	TS_BUF_EN	Выходной усилитель для датчика температуры и внутреннего источника напряжения 1.23 В: 0 – отключен; 1 – включен.
17	TS_EN	Блок датчика температуры и внутреннего источника напряжения 1.23 В: 0 – отключен; 1 – включен.
16	SYNC_CONVER	Режим запуска двух преобразователей: 0 – независимый; 1 – синхронный.
15...12	REG_DIVCLK[3:0]	Делитель тактовой частоты: 0000 – *HCLK*; 0001 – *HCLK* / 2; 0010 – *HCLK* / 4; 0011 – *HCLK* / 8; ... 1011 – *HCLK* / 2048; 1111 – *HCLK*.
11	M_REF	Источник опорного напряжения: 0 – питание аналоговой части микроконтроллера; 1 – внешнее напряжение с выводов ADC0_REF+ и ADC1_REF–.
10	REG_RNGC	Контроль границ преобразования: 0 – отключен; 1 – включен (границы задаются в регистрах ADC1_L_LEVEL и ADC1_H_LEVEL).
9	REG_CHCH	Режим последовательного переключения каналов: 0 – отключен (преобразование по одному каналу); 1 – включен (последовательное переключение каналов, заданных в регистре ADC1_CHSEL).
8...4	REG_CHS[4:0]	Выбор целевого канала для преобразования: 00000 – канал 0; 00001 – канал 1; ... 11111 – канал 31.

Продолжение таблицы 6.1

№ битов	Функциональное имя битов	Дескрипция
3	REG_SAMPLE	Способ запуска преобразователя: 0 – однократный; 1 – циклический.
2	REG_CLKS	Источник тактирования: 0 – сигнал *HCLK*; 1 – сигнал *ADC_CLK*.
1	REG_GO	Запуск преобразования: 0 – не имеет влияния; 1 – запуск.
0	REG_ADON	Работа преобразователя: 0 – отключен; 1 – включен.

Рассмотрим подробнее некоторые параметры из таблицы.

В качестве **источника тактовых импульсов** может выступать сигнал *HCLK* с процессора, поделенный на некоторый коэффициент из последовательности $\{2^n\}_{n=0}^{11}$, либо специально сформированный сигнал *ADC_CLK* (рисунок 3.5). Следует помнить, что максимальная тактовая частота АЦП составляет 14 МГц, поэтому при тактировании преобразователя от разогнанного ядра необходимо выбрать соответствующее значение предделителя.

Способ запуска преобразователя может быть однократным и циклическим. **Однократный запуск** подразумевает, что АЦП выполнит лишь одно преобразование и перейдет в режим ожидания следующего запуска. Запуск преобразования осуществляется установкой бита `REG_GO`. При **циклическом запуске**, напротив, АЦП будет выполнять преобразования с момента включения до тех пор, пока не получит команду на остановку.

При включении **режима последовательного переключения каналов** биты выбора канала (4...8) игнорируются; выбор каналов осуществляется в регистре ADCx_CHSEL. При этом запись в регистр значения, например, $8D_{16} = 0100\ 1101_2$ позволит выполнять преобразования для каналов 0, 2, 3 и 6.

Контроль границ преобразования позволяет системе устанавливать специальный флаг (регистр `ADCx_STATUS`, бит `FLG_REG_AWOIFEN`) или формировать аппаратное прерывание при выходе результата аналого-цифрового преобразования за границы, задаваемые в регистрах `ADCx_L_LEVEL` и `ADCx_H_LEVEL`.

Синхронный запуск двух преобразователей используется для увеличения частоты аналого-цифрового преобразования. При этом настройки делителя частоты, источника опорного напряжения и целевых каналов первого АЦП используются и для второго. Биты `DELAY_ADC` задают разность фаз в работе преобразователей в тактах ядра.

В микроконтроллеры серии 1986ВЕ9Х интегрирован **датчик температуры**. Для преобразования его сигнала необходимо установить биты `TS_EN`, `TS_BUF_EN`, `SEL_TS`, и выбрать для преобразования канал 31. Однако микроконтроллеры, к сожалению,

не содержат индивидуальных калибровочных данных этого датчика, поэтому для его полноценной эксплуатации сперва требуется провести измерительные процедуры с использованием климатических камер. При недоступности оных использование датчика не рекомендуется.

Внутренний источник напряжения 1.23 В обладает высокой стабильностью и может быть использован для оценки точности опорного напряжения. Напряжение источника может быть подстроено в диапазоне 1.160 ÷ 1.240 В (рисунок 6.4). Для преобразования сигнала источника следует установить биты `TS_EN`, `TS_BUF_EN`, `SEL_VREF`, и выбрать для преобразования канал 30.

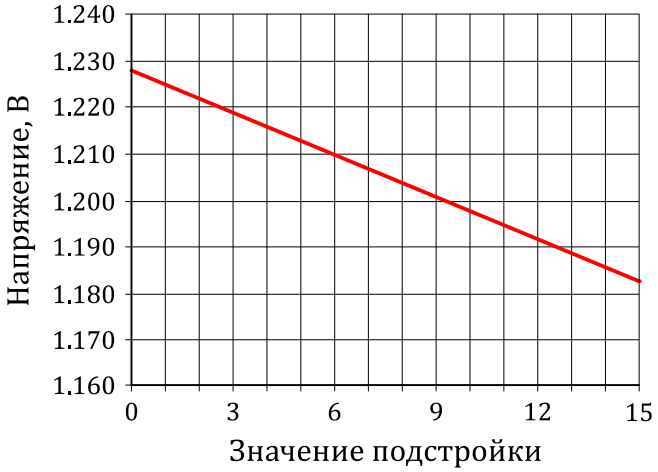

Рисунок 6.4 – Зависимость напряжения внутреннего источника от подстройки

Дополнительная задержка перед преобразованием положительно коррелирует с точностью преобразования. Дело в том, что для выполнения преобразования входной сигнал должен быть зафиксирован устройством выборки и хранения (раздел 6.2), а заряд внутренней емкости этого устройства до уровня внешнего сигнала требует некоторого времени. Задержка перед преобразованием увеличивает время заряда на несколько тактов ядра, позволяя внутреннему и внешнему напряжениям сравняться.

Обратите внимание, что упомянутые задержки задаются именно в **тактах ядра (*HCLK*)**. При этом не имеет значения, на какой частоте работают собственно преобразователи.

Таким образом, конфигурация аналого-цифрового преобразователя для работы с пятым каналом (*PD*5) в простейшем варианте может выглядеть так:

```
// Конфигурация АЦП
MDR_ADC->ADC1_CFG =
    (1 << ADC1_CFG_REG_ADON_Pos)     // Работа АЦП (включен)
  | (0 << ADC1_CFG_REG_CLKS_Pos)     // Источник тактирования АЦП (CPU)
  | (0 << ADC1_CFG_REG_SAMPLE_Pos)   // Способ запуска АЦП (однократный)
  | (5 << ADC1_CFG_REG_CHS_Pos)      // Целевой канал преобразователя (ADC5)
  | (0 << ADC1_CFG_REG_CHCH_Pos)     // Режим последовательного переключения
                                     // каналов (отключен)
  | (0 << ADC1_CFG_REG_RNGC_Pos)     // Контроль границ преобразования (отключен)
  | (0 << ADC1_CFG_M_REF_Pos)        // Источник опорного напряжения (внутренний)
  | (11 << ADC1_CFG_REG_DIVCLK_Pos)  // Делитель тактовой частоты для АЦП (2^11 = 2048)
  | (0 << ADC1_CFG_SYNC_CONVER_Pos)  // Режим запуска двух АЦП (независимый)
```

```
// ...Конфигурация датчика температуры и внутреннего источника напряжения 1.23 В
| (0 << ADC1_CFG_TS_EN_Pos)        // Работа датчика температуры (отключен)
| (0 << ADC1_CFG_TS_BUF_EN_Pos)    // Работа усилителя для датчика температуры и
                                   // внутр. источника напряжения 1.23 В (отключен)
| (0 << ADC1_CFG_SEL_TS_Pos)       // Преобразование сигнала
                                   // с датчика температуры (отключено)
| (0 << ADC1_CFG_SEL_VREF_Pos)     // Преобразование сигнала с внутр.
                                   // источника напряжения 1.23 В (отключено)
| (0 << ADC1_CFG_TR_Pos)           // Подстройка напряжения внутр. источника 1.23 В

// ...Настройка задержек при преобразовании
| (0 << ADC1_CFG_DELAY_GO_Pos)     // Дополнительная задержка перед преобразованием
                                   // для заряда внутренней емкости (1 такт)
| (0 << ADC1_CFG_DELAY_ADC_Pos);   // Разность фаз между циклами
                                   // преобразователей (не используется)
```

5. Настройка аппаратных прерываний контроллера АЦП производится в регистрах ADCx_STATUS (таблица 6.2).

Таблица 6.2 – Описание регистров MDR_ADC->ADCx_STATUS

№ битов	Функциональное имя битов	Описание
31…5	–	–
4	ECOIF_IE	Запрос прерывания при завершении преобразования: 0 – запрещен; 1 – разрешен.
3	AWOIF_IE	Запрос прерывания при выходе за заданные границы преобразования: 0 – запрещен; 1 – разрешен.
2	FLG_REG_EOCIF	Флаг завершения аналого-цифрового преобразования (сброс осуществляется путем чтения регистров ADCx_RESULT): 0 – регистр не содержит готового результата; 1 – регистр содержит готовый результат.
1	FLG_REG_AWOIFEN	Флаг контроля границ преобразования: 0 – результат преобразования в пределах заданных границ; 1 – результат преобразования за пределами заданных границ.
0	FLG_REG_OVERWRITE	Флаг перезаписи результата аналого-цифрового преобразования: 0 – несчитанных результатов не зафиксировано; 1 – осуществлена запись нового результата поверх несчитанного.

Чтобы организовать прерывания по завершении преобразования требуется, во-первых, разрешить отправление запросов контроллером АЦП; во-вторых, настроить контроллер NVIC; и в-третьих, описать обработчик аппаратных прерываний:

```
// Настройка запросов на обработку прерываний от АЦП
MDR_ADC->ADC1_STATUS = (1 << ADC_STATUS_ECOIF_IE_Pos);

// Назначение приоритета прерывания от АЦП
NVIC_SetPriority(ADC_IRQn, 1);

// Разрешение обработки прерывания от АЦП
NVIC_EnableIRQ(ADC_IRQn);

...

// Обработчик прерывания от АЦП
void ADC_IRQHandler(void)
{
  // Получение результата преобразования и сброс флага прерывания
  // (digit – глобальная переменная)
  digit = MDR_ADC->ADC1_RESULT & ADC_RESULT_Msk;
}
```

6.3.2. Однократное преобразование

Однократное преобразование – наиболее простой вид аналого-цифрового преобразования. Для примера рассмотрим, как выполнить преобразование напряжения, подведенного к пятому каналу АЦП (*PD*5) с использованием сервисов операционной системы. Конфигурация АЦП, приведенная в разделе 6.3.1, без модификаций может быть использована для работы в этом режиме.

Однократное преобразование выполняется по следующему алгоритму:

1. Запуск аналого-цифрового преобразователя.
2. Ожидание завершения преобразования.
3. Обработка результата преобразования.
4. Задержка перед началом следующей итерации.

Фрагмент программы, выполняющий эти действия, выглядит так:

```
// Калибровочные значения напряжения
#define CALIBR_CONST_V 3.290F
#define CALIBR_CONST_D 0xFFF

// Макрос для расчета напряжения
#define VOLTAGE(D) CALIBR_CONST_V * (D) / CALIBR_CONST_D

// Поток аналого-цифровых преобразований
void Thread_ADC(void *argument)
{
  uint16_t digit;    // Значение аналого-цифрового преобразования
  float voltage;     // Расчетное значение напряжения

  // Основной цикл
  while (1)
  {
    // Запуск процесса аналого-цифрового преобразования
    MDR_ADC->ADC1_CFG |= (1 << ADC1_CFG_REG_GO_Pos);

    // Ожидание окончания аналого-цифрового преобразования
    osMessageQueueGet(MsgQueId_ADC, &digit, NULL, osWaitForever);
```

```
    // Вычисление напряжения на основе результата преобразования
    voltage = VOLTAGE(digit);

    // Задержка перед началом следующей итерации
    osDelay(250);
  }
}

// Обработчик прерывания АЦП
void ADC_IRQHandler(void)
{
  // Получение результата преобразования и сброс флага прерывания
  uint16_t digit = MDR_ADC->ADC1_RESULT & ADC_RESULT_Msk;

  // Отправление сообщения с результатом аналого-цифрового преобразования
  osMessageQueuePut(MsgQueId_ADC, &digit, osPriorityNormal, NULL);
}
```

Запуск преобразования осуществляется установкой бита REG_GO регистра ADC1_CFG (таблица 6.2). Далее поток попадает в состояние ожидания до тех пор, пока не получит сообщение с результатом аналого-цифрового преобразования. Данное сообщение формируется и отправляется в обработчике аппаратных прерываний.

Затем необходимо интерпретировать полученное цифровое значение. Для этого требуется определить зависимость напряжения от цифрового кода. Имея в виду линейную зависимость, а также принимая во внимание значение опорного напряжения (U_{REF} = 3.3 В) и разрядность АЦП (R = 12), можно вывести такую функцию:

$$U(D) = \frac{U_{REF}}{2^R - 1} \times D = \frac{3.3}{2^{12} - 1} \times D = 8.06 \times 10^{-4} \times D,$$

где D – результат аналого-цифрового преобразования.

Значения 3.3 В и $2^{12} - 1$ в данном случае будет выступать в качестве **калибровочных констант**: при несоответствии фактического значения напряжения результатам измерения, эти константы могут быть скорректированы.

Задержка в конце цикла определяет частоту преобразований АЦП.

6.3.3. Циклическое преобразование

Как известно, аналоговые сигналы имеют свойство постоянно изменяться во времени. Это, в свою очередь, влечет за собой колебания результата аналого-цифрового преобразования. И в большинстве случаев – это явление нежелательное.

Для стабилизации итогового результата преобразования используют метод **децимации путем суммирования и усреднения отсчетов**. Суть метода очень проста: вместо одного преобразования выполняется n преобразований, формируя таким образом выборку результатов; все элементы выборки суммируются, а сумма делится на размер выборки n. Иными словами, находится **среднее арифметическое** выборки. Отрицательной стороной этого метода является снижение эффективной частоты преобразования в n раз.

Для программной реализации данного метода рекомендуется использовать режим циклического преобразования. Задействование при этом прямого доступа к памяти (DMA) позволит значительно снизить нагрузку на процессорное ядро.

Процедура инициализации DMA была подробно описана в разделе 5.5.4. Сейчас остановимся лишь на ключевых параметрах:

1. Для работы с первым АЦП в контроллере DMA зарезервирован канал 8.
2. Режим работы DMA – базовый.
3. Информационный объем передачи – 2 байта.
4. Адрес источника данных – регистр результатов АЦП: `MDR_ADC->ADC1_RESULT`.
5. Адрес приемника данных – 2-байтовый массив в памяти: `uint16_t sample[n]`.
6. Инкремент адреса источника данных отсутствует.
7. Инкремент адреса приемника равен 2 байтам.
8. Количество передач в цикле должно быть равно размеру выборки *n*.

При инициализации АЦП, в отличие от предыдущего варианта, следует изменить биты `ADC1_CFG_REG_ADON_Pos` и `ADC1_CFG_REG_SAMPLE_Pos` регистра `ADC1_CFG` и **не** включать аппаратные прерывания:

```
// Конфигурация АЦП
MDR_ADC->ADC1_CFG =
    (0 << ADC1_CFG_REG_ADON_Pos)    // Работа АЦП (пока отключен)
    ...
  | (1 << ADC1_CFG_REG_SAMPLE_Pos)  // Способ запуска АЦП (циклический)
    ...;
```

Отключение АЦП на этапе конфигурации связано с особенностью циклического режима: преобразования запускаются при активации АЦП, без явной команды. Для правильной работы системы процедура полной инициализации должна предшествовать началу аналого-цифровых преобразований, поэтому запуск преобразователя выносится за пределы процедуры. Индикацией завершения цикла преобразований в данном проекте будет служить прерывание от контроллера DMA, ввиду чего в прерывании от АЦП нет потребности.

Далее приведен фрагмент программы, выполняющий аналого-цифровые преобразования в циклическом режиме:

```
// Калибровочные значения напряжения
#define CALIBR_CONST_V 3.290F
#define CALIBR_CONST_D 0xFFF

// Макрос для расчета напряжения
#define VOLTAGE(D) CALIBR_CONST_V * (D) / CALIBR_CONST_D

// Размер выборки
#define ADC_SAMPLE_SIZE 128

// Буфер для хранения результатов преобразования
volatile uint16_t sample[ADC_SAMPLE_SIZE];

// Поток аналого-цифровых преобразований
void Thread_ADC(void *argument)
{
  uint32_t digit;  // Значение аналого-цифрового преобразования
  float voltage;   // Расчетное значение напряжения
  uint32_t i;
```

```
  // Основной цикл
  while (1)
  {
    // Запуск цикла аналого-цифровых преобразований
    MDR_ADC->ADC1_CFG |= (1 << ADC1_CFG_REG_ADON_Pos);

    // Ожидание завершения цикла аналого-цифровых преобразований
    osEventFlagsWait(EventId_ADC, EVENT_ADC_EOC, osFlagsWaitAny, osWaitForever);

    // Сложение элементов выборки
    for (i = 0, digit = 0; i < ADC_SAMPLE_SIZE; i++)
      digit += sample[i] & ADC_RESULT_Msk;

    // Усреднение результата
    digit /= ADC_SAMPLE_SIZE;

    // Вычисление напряжения на основе результата преобразования
    voltage = VOLTAGE(digit);

    // Задержка перед началом следующей итерации
    osDelay(250);
  }
}

// Обработчик прерывания контроллера DMA
void DMA_IRQHandler(void)
{
  // Отключение АЦП
  MDR_ADC->ADC1_CFG &= ~(1 << ADC1_CFG_REG_ADON_Pos);

  // Установка флага завершения цикла передач
  osEventFlagsSet(EventId_ADC, EVENT_ADC_EOC);

  // Формирование нового цикла передач
  // путем модификации полей структуры управляющих данных
  DMA_ControlData[8].DMA_Control |= (1 << DMA_CHANNEL_CFG_CYCLE_CTRL_Pos)
                                  | ((ADC_SAMPLE_SIZE - 1) << DMA_CHANNEL_CFG_N_MINUS_1_Pos);

  // Разрешение работы канала 8
  MDR_DMA->CHNL_ENABLE_SET = (1 << 8);
}
```

В первую очередь необходимо создать глобальный массив, который будет хранить выборку результатов преобразования. Данный массив следует объявлять со спецификатором `volatile`, чтобы избежать возможных проблем при компиляции.

Цикл аналого-цифровых преобразований запускается при включении АЦП. После этого поток переходит в режим ожидания до завершения цикла.

Контроллер DMA последовательно передает результаты преобразований в массив до тех пор, пока количество передач не сравняется с размером выборки. После этого формируется аппаратное прерывание, в обработчике которого производится отключение АЦП, оповещение потока о завершении цикла преобразований и подготовка следующего цикла передач.

Обработка полученной выборки заключается в суммировании всех ее элементов и делении суммы на ее размер. Затем остается лишь пересчитать полученное цифровое значение в напряжение уже рассмотренным способом.

6.3.4. Последовательное переключение каналов

Последовательное переключение каналов применяется для оцифровки сигналов с нескольких источников. Идея состоит в том, что при настройке преобразователя, например, на каналы 2, 5 и 7, при первом его запуске будет совершено преобразование по каналу 2, при втором – по каналу 5, при третьем – по каналу 7, при четвертом – вновь по каналу 2 и т.д. Реализованное количество каналов в микроконтроллерах серии 1986BE9x различно; для микроконтроллера 1986BE92У оно равно 8.

Последовательное переключение каналов чаще всего используется совместно с циклическим режимом для непрерывного контроля сигналов. В таком случае, перед обработкой выборки результатов может возникнуть вопрос: как определить, к какому каналу относится каждый из элементов выборки? В действительности, достаточно просто: регистры результатов преобразования ADCx_RESULT содержат, помимо собственно результатов, номер рабочего канала (таблица 6.3). При этом, однако, нужно обратить внимание, что тип массива, формируемого для хранения выборки, должен быть 32-битным (16-битного теперь будет недостаточно), а контроллер DMA должен быть настроен на передачу 4 байтов с равным инкрементом источника.

Таблица 6.3 – Описание регистров MDR_ADC->ADCx_RESULT

№ битов	Функциональное имя битов	Дескрипция
31...21	–	–
20...16	CHANNEL[4:0]	Номер активного канала: 00000 – канал 0; 00001 – канал 1; ... 11111 – канал 31.
15...12	–	–
11...0	RESULT[11:0]	Результат аналого-цифрового преобразования.

Что касается процедуры инициализации АЦП в режиме последовательного переключения каналов, то она имеет две особенности.

Во-первых, в связи с тем, что при переключении каналов нередко встречаются резкие перепады напряжения (например, переход от 0.1 В до 3.2 В), рекомендуется максимально эксплуатировать функцию задержки между преобразованиями для заряда внутренней емкости, т.е. задать ей значение, соответствующее 8 тактам ядра (биты DELAY_GO регистра ADCx_CFG).

Во-вторых, следует помнить, что целевые каналы преобразования задаются не в регистре конфигурации ADCx_CFG, в специальном регистре ADCx_CHSEL.

Так, например, выглядит фрагмент инициализации АЦП для работы с тремя каналами:

```
// Конфигурация АЦП
MDR_ADC->ADC1_CFG =
    (0 << ADC1_CFG_REG_ADON_Pos)    // Работа АЦП (пока отключен)
    ...
  | (1 << ADC1_CFG_REG_SAMPLE_Pos)  // Способ запуска АЦП (циклический)
  | (0 << ADC1_CFG_REG_CHS_Pos)     // Целевой канал преобразователя (не указывается)
  | (1 << ADC1_CFG_REG_CHCH_Pos)    // Режим последовательного
                                    // переключения каналов (включен)
    ...
  | (7 << ADC1_CFG_DELAY_GO_Pos)    // Дополнительная задержка перед преобразованием
                                    // для заряда внутренней емкости (8 тактов)
    ...;

// Выбор каналов для режима последовательного переключения
MDR_ADC->ADC1_CHSEL = (1 << 2)   // Канал 2
                    | (1 << 5)   // Канал 5
                    | (1 << 7);  // Канал 7
```

Дальнейшее управление преобразователем может иметь такой вид:

```
// Калибровочные значения напряжения
#define CALIBR_CONST_V 3.290F
#define CALIBR_CONST_D 0xFFF

// Макрос для расчета напряжения
#define VOLTAGE(D) CALIBR_CONST_V * D / CALIBR_CONST_D

// Размер выборки
#define ADC_SAMPLE_SIZE 256

// Буфер для хранения результатов преобразования
volatile uint32_t sample[ADC_SAMPLE_SIZE];

// Поток аналого-цифровых преобразований
void Thread_ADC(void *argument)
{
  // Значения преобразований по каналам 2, 5, 7 соответственно
  uint32_t digit2, digit5, digit7;

  // Количество преобразований по каналам 2, 5, 7 соответственно
  uint16_t amount2, amount5, amount7;

  // Расчетные значения напряжений на каналах 2, 5, 7 соответственно
  float voltage2, voltage5, voltage7;

  uint32_t i;

  // Основной цикл
  while (1)
  {
    // Запуск цикла аналого-цифровых преобразований
    MDR_ADC->ADC1_CFG |= (1 << 0);

    // Ожидание завершения цикла аналого-цифровых преобразований
    osEventFlagsWait(EventId_ADC, EVENT_ADC_EOC, osFlagsWaitAny, osWaitForever);
```

```
      // Сложение элементов выборки преобразования
      for (i = 0, digit2  = 0, digit5  = 0, digit7  = 0,
                  amount2 = 0, amount5 = 0, amount7 = 0;
           i < ADC_SAMPLE_SIZE; i++)
      {
        // Определение канала, по которому выполнение преобразование
        switch ((sample[i] & ADC_RESULT_CHANNEL_Msk) >> ADC_RESULT_CHANNEL_Pos)
        {
          // Канал 2
          case 2:
            digit2 += sample[i] & ADC_RESULT_Msk;
            amount2++;
            break;

          // Канал 5
          case 5:
            digit5 += sample[i] & ADC_RESULT_Msk;
            amount5++;
            break;

          // Канал 7
          case 7:
            digit7 += sample[i] & ADC_RESULT_Msk;
            amount7++;
            break;
        }
      }

      // Усреднение результатов
      if (amount2)
        digit2 /= amount2;

      if (amount5)
        digit5 /= amount5;

      if (amount7)
        digit7 /= amount7;

      // Вычисление напряжения на основе результатов преобразования
      voltage2 = VOLTAGE(digit2);
      voltage5 = VOLTAGE(digit5);
      voltage7 = VOLTAGE(digit7);

      // Задержка перед началом следующей итерации
      osDelay(250);
    }
  }
```

Наибольший интерес в приведенном фрагменте программы представляет обработка результатов преобразования.

В первую очередь необходимо объявить по три переменные для каждого целевого канала: одна будет хранить результат аналого-цифрового преобразования, вторая – количество преобразований по каналу в каждом цикле, третья – расчетное значение напряжения.

После завершения цикла аналого-цифровых преобразований следует вычислить три значения, каждое из которых должно представлять собой сумму преобразований по определенному каналу. Рациональнее всего это сделать с помощью оператора выбора `switch`. Обратите внимание, что каждая итерация суммирования регистрируется с помощью

инкремента специальной переменной, чтобы иметь возможность следующим действием найти среднее арифметическое. Проверка существования регистрации при расчете среднего арифметического требуется для того, чтобы исключить возможность деления на ноль.

Обработчик прерывания контроллера DMA не приведен, т.к. его содержание идентично рассмотренному в предыдущем разделе.

6.4. Термисторы

Термистор – это полупроводниковый прибор, электрическое сопротивление которого изменяется в зависимости от температуры. Термисторы находят множество применений в электронике; в частности, они могут быть использованы в качестве простейшего датчика температуры: зная температурную зависимость термистора и измерив его сопротивление можно определить температуру окружающей среды.

По типу зависимости сопротивления от температуры различают термисторы с отрицательным (**NTC**) и положительным (**PTC**) температурным коэффициентом. При отрицательном температурном коэффициенте сопротивление термистора снижается с ростом температуры, при положительном – наоборот.

С целью измерения температуры обычно применяют NTC-термисторы. Температурная зависимость сопротивления термисторов является экспоненциальной, и при отрицательном температурном коэффициенте в некотором приближении может быть описана т.н. β-уравнением:

$$R(T) = R_0 \times e^{\beta \times \left(\frac{1}{T} - \frac{1}{T_0}\right)}, \tag{6.3}$$

где R_0 – номинальное сопротивление при номинальной температуре T_0,

β – специальная константа термистора (указывается в документации),

значения температур при этом берутся по абсолютной шкале (в кельвинах).

В практических целях, однако, удобнее использовать зависимость $T(R)$, которую можно получить из формулы 6.3:

$$T(R) = \frac{\beta \times T_0}{T_0 \times \ln\left(\frac{R}{R_0}\right) + \beta}. \tag{6.4}$$

Определить мгновенное значение сопротивления термистора можно с помощью аналого-цифрового преобразователя. Однако для этого термистор должен быть подключен по схеме делителя напряжения (рисунок 6.5).

Значение напряжения в средней точке данной схемы будет равно:

$$U_{OUT} = U_{REF} \frac{R_B}{R_T + R_B}, \tag{6.5}$$

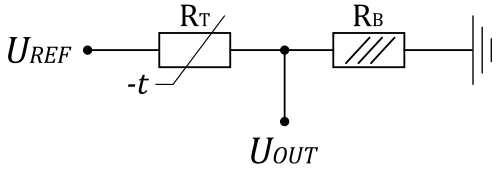

Рисунок 6.5 – Схема делителя напряжения с использованием термистора
(U_{REF} – делимое напряжение, U_{OUT} – напряжение средней точки,
R_T – термистор, R_B –балансировочный резистор)

Номинал балансировочного резистора R_B следует выбирать равным сопротивлению термистора в середине целевого температурного диапазона. Например, если система предназначена для измерения температур в диапазоне 0...40 °C, и применяется термистор с номинальным сопротивлением 10 КОм при температуре 25 °C и β = 3950 К, то по формуле 6.3 требуемое значение балансировочного резистора составит:

$$R_B = 10^4 \times e^{3950\times\left(\frac{1}{293.15}-\frac{1}{298.15}\right)} \approx 12.5 \text{ КОм}.$$

Измерив напряжение в средней точке схемы, по формуле 6.4 можно определить соответствующее значение сопротивления термистора:

$$R_T = R_B \times \left(\frac{U_{REF}}{U_{OUT}} - 1\right). \tag{6.6}$$

Если опорное напряжение аналого-цифрового преобразователя и напряжение схемы делителя равны, то выражение 6.6 может быть записано так:

$$R_T = R_B \times \left(\frac{2^R - 1}{D} - 1\right), \tag{6.7}$$

где R – разрядность аналого-цифрового преобразователя,

D – результат преобразования.

Если пойти еще дальше, то используя выражения 6.4 и 6.7 можно определить зависимость температуры от результата аналого-цифрового преобразования:

$$T(D) = \frac{\beta \times T_0}{T_0 \times \ln\left(\frac{R_B \times \frac{2^R - 1}{D} - 1}{R_0}\right) + \beta}. \tag{6.8}$$

На модуле расширения установлен NTC-термистор **MF52F103J3950** с такими параметрами: R (25 °C) = 10 КОм; β = 3950 К. Номинал балансировочного резистора равен 10 КОм. Способ подключения термистора приведен в разделе *Описание программных проектов* текущей главы.

При работе с микроконтроллером все параметры схемы рекомендуется задать в виде констант, что позволит возложить бóльшую часть приведенных вычислений на препроцессор.

Фрагмент программы, реализующий измерение температуры с использованием термистора, может иметь такой вид:

```
// Подключение стандартной библиотеки
// для выполнения математических операций
#include <math.h>

// Сопротивление балансировочного резистора (в омах)
#define RB    10000

// Номинальные значения термистора
#define R0    10000    // Сопротивление (в омах)
#define T0    298.15F  // Температура (в кельвинах)
#define BETA  3950     // Бета-коэффициент

// Максимальное значение аналого-цифрового преобразования
#define D_MAX 4095

// Макрос для расчета температуры (в градусах Цельсия)
#define TEMPERATURE(D) (BETA * T0 /                                    \
                       (T0 * log(RB * (D_MAX / (D) - 1) / R0) + BETA)) + 273.15
```

Приведенный макрос соответствует формуле 6.8, но с той разницей, что возвращает значение температуры в градусах Цельсия.

6.5. Двухосевые джойстики

Двухосевой джойстик – это электромеханическое устройство, предназначенное для управления объектом по двум координатам (рисунок 6.6). Такие устройства широко используются в манипуляторах для игровых приставок *PlayStation* и *XBOX*.

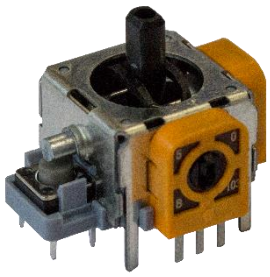

Рисунок 6.6 – Внешний вид двухосевого джойстика White Label

Джойстик состоит из двух потенциометров и механической кнопки. Рычаг, изменяющий состояние потенциометров, закреплен на шарнире и связан с пружинами, возвращающими его в центральное положение после отклонения. Кнопка механически связана с рычагом: при вертикальном нажатии на рычаг контакты кнопки замыкаются.

Электрическая схема джойстика приведена на рисунке 6.7.

При центральном положении рычага потенциометры делят входное напряжение пополам. Смещение рычага влево увеличивает сопротивление потенциометра R_X, смещение вправо – уменьшает. Ситуация аналогична для потенциометра R_Y с той разницей, что речь идет о смещении вниз/вверх. Таким образом, сигналы с потенциометров после программной обработки могут интерпретироваться в качестве управляющих воздействий.

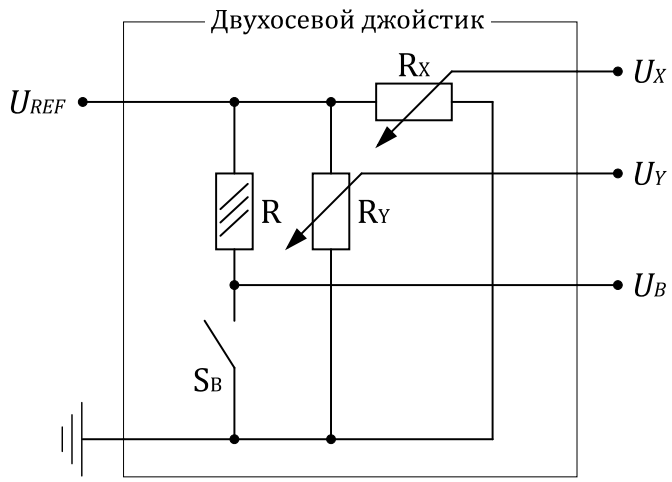

Рисунок 6.7 – Электрическая схема двухосевого джойстика

Необходимо отметить, что вследствие трения в механических деталях, пружины джойстика не возвращают рычаг точно в центральное положение. По этой причине напряжение потенциометров в состоянии покоя будет отличаться от половины опорного напряжения на некоторую величину.

Для создания эргономичного управления необходимо правильно настроить чувствительность джойстика, иначе отклик будет происходить даже при небольших отклонениях рычага. С этой целью программно формируется **мертвая зона**, т.е. диапазон отклонений, который не будет обработан системой. Ширина мертвой зоны подбирается эмпирически и зависит от конкретного экземпляра джойстика и требований управления. Практика показывает, что в большинстве случаев она должна быть достаточно велика – примерно в четверть всего диапазона напряжения.

Для постоянного отслеживания сигналов с джойстика рационально использовать циклический режим с последовательным переключением каналов. Количество передач в цикле не имеет смысла делать большим, т.к. наличие мертвой зоны снижает требования к стабильности сигнала.

Фрагмент программы, изменяющий координаты некоторого объекта в зависимости от сигналов с джойстика, может выглядеть так:

```
// Максимальное значение аналого-цифрового преобразования
#define D_MAX 4095

// Константа для задания мертвой зоны (в цифровых единицах)
#define DELTA 1000

// Размер выборки
#define ADC_SAMPLE_SIZE 16

// Буфер для хранения результатов преобразования
volatile uint32_t sample[ADC_SAMPLE_SIZE];
```

```
// Инициализация аналого-цифрового преобразователя
void ADC_Init(void)
{
  ...

  // Конфигурация АЦП
  MDR_ADC->ADC1_CFG =
    (0 << ADC1_CFG_REG_ADON_Pos)    // Работа АЦП (пока отключен)
    ...
    | (1 << ADC1_CFG_REG_SAMPLE_Pos)  // Способ запуска АЦП (циклический)
    ...
    | (1 << ADC1_CFG_REG_CHCH_Pos)    // Режим последовательного
                                      // переключения каналов (включен)
    ...;

  // Выбор каналов для режима последовательного переключения
  MDR_ADC->ADC1_CHSEL = (1 << 5)   // Канал потенциометра RX
                      | (1 << 6);  // Канал потенциометра RY
}

// Поток аналого-цифровых преобразований
void Thread_ADC(void *argument)
{
  uint32_t digitX,  digitY;   // Значения преобразований по каналам потенциометров
  uint16_t amountX, amountY;  // Количество преобразований по каналам потенциометров
  uint32_t i;

  // Основной цикл
  while (1)
  {
    // Включение аналого-цифрового преобразователя
    MDR_ADC->ADC1_CFG |= (1 << ADC1_CFG_REG_ADON_Pos);

    // Ожидание завершения цикла аналого-цифровых преобразований
    osEventFlagsWait(EventId_ADC, EVENT_ADC_EOC, osFlagsWaitAny, osWaitForever);

    // Сложение элементов выборки результатов
    for (i = 0, digitX  = 0, digitY  = 0,
                amountX = 0, amountY = 0;
         i < ADC_SAMPLE_SIZE; i++)
    {
      // Определение канала, по которому выполнение преобразование
      switch ((sample[i] & ADC_RESULT_CHANNEL_Msk) >> ADC_RESULT_CHANNEL_Pos)
      {
        // Канал напряжения
        case 5:
          digitX += sample[i] & ADC_RESULT_Msk;
          amountX++;
          break;

        // Канал измерения температуры
        case 6:
          digitY += sample[i] & ADC_RESULT_Msk;
          amountY++;
          break;
      }
    }
```

```
    // Усреднение результатов
    if (amountX)
      digitX /= amountX;

    if (amountY)
      digitY /= amountY;

    // Смещение джойстика влево
    if (digitX < (D_MAX / 2 - DELTA)
      x--;

    // Смещение джойстика вправо
    if (digitX > (D_MAX / 2 + DELTA)
      x++;

    // Смещение джойстика вниз
    if (digitY < (D_MAX / 2 - DELTA)
      y--;

    // Смещение джойстика вверх
    if (digitY > (D_MAX / 2 + DELTA)
      y++;

    // Задержка перед началом следующей итерации
    osDelay(100);
  }
}
```

Описание программных проектов

Для корректного исполнения программных проектов требуется подключить модуль расширения к отладочной плате согласно таблицам 6.4 и 6.5.

Таблица 6.4 – Подключение модуля расширения к отладочной плате для работы проектов *Sample 6.1* и *Sample 6.2*

№ п/п	Модуль расширения		Отладочная плата	
	Имя контакта	Имя разъема	Имя контакта	Имя разъема
1	GND	XP3	1	X26
2	3V3		3	
3	SHAFT		10	
4	TR_OUT		11	

Таблица 6.5 – Подключение модуля расширения к отладочной плате для работы проектов *Sample 6.3* и *Sample 6.4*

№ п/п	Модуль расширения		Отладочная плата	
	Имя контакта	Имя разъема	Имя контакта	Имя разъема
1	GND	XP3	1	X26
2	3V3		3	
3	JS_X		10	
4	JS_Y		11	
5	JS_B		13	

Проект **Sample 6.1** выполняет одиночные преобразования сигнала с внешнего потенциометра. Цифровой результат и расчетное значение напряжения отображается на дисплей. При повороте ручки потенциометра напряжение меняется в диапазоне 0...3.3 В.

Проект **Sample 6.2** выполняет циклические преобразования сигнала с термистора с последующим расчетом температуры окружающей среды. Для стабилизации показаний реализована децимация цифровых значений; выборка формируется через прямой доступ к памяти.

Проект **Sample 6.3** выполняет циклические преобразования сигналов с двухосевого джойстика в режиме последовательного переключения каналов. На дисплей отображаются цифровые значения результатов преобразования, а также состояние кнопки джойстика.

Проект **Sample 6.4** представляет собой небольшую игру «*Catch The Doggie*». Смысл игры заключается в том, чтобы находить указателем появляющуюся в случайных местах «собачку». Управление указателем осуществляется с помощью джойстика. На партию отводится 1 минута. Рекордный результат помещается в регистр аварийного сохранения и отображается на дисплей после каждой партии. Перезапуск игры осуществляется кнопкой *RESET*.

Задачи для самостоятельной работы

[*проект Sample 6.1*]

1. Организуйте отображение на дисплей, наряду с напряжением, угла поворота ручки потенциометра в градусах (диапазон – 30...330°; точность – 1°).

2. Стабилизируйте результат измерений; для этого в цикле формируйте выборку из 8 цифровых значений, находите их среднее арифметическое и используйте его для последующих вычислений.

[*проект Sample 6.2*]

3. Модифицируйте проект таким образом, чтобы по нажатию кнопки *SEL* изменялись единицы измерения температуры:

градусы Цельсия (°C) → Кельвины (К) → градусы Фаренгейта (°F).

[*проект Sample 6.3*]

4. Добавьте в функционал проекта преобразование напряжения с подстроечного резистора *TRIM*, расположенного на отладочной плате (маленькая синяя коробочка). Резистор схемотехнически подведен к каналу *ADC7*. Обратите внимание, что для его подключения необходимо установить перемычку *ADC_IN_SEL* в положение *TRIM*.

[*проект Sample 6.4*]

5. Наберите 30 очков в игре *Catch The Doggie*! ☺

Контрольные вопросы

1. Для каких целей используются аналого-цифровые преобразователи (АЦП)?
2. Какими основными характеристиками обладают АЦП?
3. Какой диапазон напряжений может оцифровывать АЦП?
4. Сколько квантовых уровней реализовано в 8-разрядном АЦП?
5. АЦП какого типа интегрированы в микроконтроллеры серии 1986ВЕ9х?
6. Какие основные параметры необходимо настроить при инициализации АЦП?
7. В чем отличие однократного и циклического режимов работы АЦП?
8. Как определить, по какому каналу выполнено аналого-цифровое преобразование?
9. Как выполнить измерение температуры с использованием термистора?
10. Как организовать управление системой с использованием двухосевого джойстика?