Частотомер на основе микропроцессора STM32

[1 Введение 1](#_Toc104886192)

[2 Программа для частотомера на основе STM32 4](#_Toc104886193)

[3 Комментарии к программе 10](#_Toc104886194)

[3.1 Определение переменных 10](#_Toc104886195)

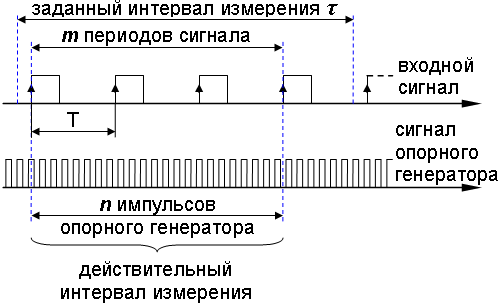
[3.2 Определение функций 13](#_Toc104886196)

1. Введение

**Частотомер** — [радиоизмерительный прибор](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B4%D0%B8%D0%BE%D0%B8%D0%B7%D0%BC%D0%B5%D1%80%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%B8%D0%B1%D0%BE%D1%80%D1%8B) для определения [частоты периодического процесса](https://ru.wikipedia.org/wiki/%D0%A7%D0%B0%D1%81%D1%82%D0%BE%D1%82%D0%B0_%D0%BF%D0%B5%D1%80%D0%B8%D0%BE%D0%B4%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%B0) или частот гармонических составляющих [спектра](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D0%BA%D1%82%D1%80) сигнала.

**Идея**: с помощью счетчика подсчитываем количество импульсов эталонного генератора за время, равное одному или нескольким периодам исследуемого входного сигнала.

Необходимо  определить период входного сигнала. А точнее, среднее значение периода за время измерения. Так, если за m периодов входного сигнала мы насчитали n импульсов эталонного генератора, частота которого равна F0, то среднее значение для одного периода входного сигнала T = n / (m F0). Однако, зная период, нетрудно вычислить и частоту (см. Рис. 1):



1. Определение частоты входного сигнала

В процессе измерений нам нужно выделять один или несколько периодов входного сигнала. Для этого выберем некоторые характерные точки сигнала, которые примем за начала периодов. Тогда период будет заключён между двумя последовательными такими точками (между началом этого периода и началом следующего). Для цифрового (импульсного) сигнала естественно принять фронт или спад сигнала за начало периода. Договоримся, для определённости, что далее началом периода будем считать фронт сигнала.

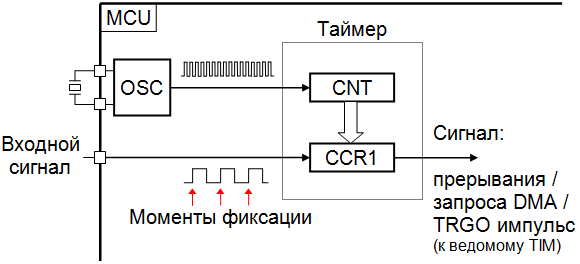
Перед выполнением измерения следует задать желаемый интервал измерения τ, т.е. время, отводимое на одно измерение. Эту величину можно изменять в широких пределах, в зависимости от требований к скорости измерения и точности результата. Относительная погрешность измерения без учёта ошибки, возникающей из-за нестабильности эталонного генератора (только погрешность счёта) может быть оценена как

Здесь мы учли то, что абсолютная точность подсчёта *n* составляет ±1. Как видим, увеличение интервала измерения снижает относительную погрешность счёта (этого же можно добиться при неизменном интервале, увеличивая частоту эталонного генератора).

Также от выбранного интервала измерения зависит минимальная частота сигнала, которую частотомер способен измерить (нижний предел измерения). Для гарантированно успешного измерения, интервал измерения должен быть хотя бы вдвое больше периода измеряемого сигнала (для большей надёжности лучше, если условие выполняется с запасом), т.е.

При выполнении этого условия, независимо от начальной фазы входного сигнала, в интервал измерения попадёт, как минимум, два фронта сигнала (что позволит выделить хотя бы один период сигнала). Например, при нижнем пределе 10 Гц, интервал измерение должен быть не менее 0.2 с. А при выборе интервала в 1 мс (для проведения высокоскоростных измерений), нижний предел прибора составит 2000 Гц.

Таким образом, задача состоит в измерении двух величин: количества периодов входного сигнала m, определяющих время счёта и количества импульсов n опорного генератора за это время. Зная частоту опорного генератора, рассчитываем частоту сигнала.



1. Вычисление частоты на микроконтроллере STM32

Таймеры микроконтроллеров STM32 прекрасно подходят для реализации описанной выше идеи измерения частота (рис. 2). Таймеры общего назначения этих микроконтроллеров имеют так называемые каналы, которые способны фиксировать текущее значение счётчика таймера по фронту внешнего сигнала.

Если упрощённо описывать процесс измерения, то всё что нужно, это настроить таймер и запустить его. При этом, в зависимости от настроек, на счётный вход таймера может быть подан как сигнал от отдельного внешнего опорного генератора, так и тактовый сигнал с шины. То есть, в простейшем случае, можно обойтись тактовым сигналом микроконтроллера для использования в качестве сигнала эталонного генератора.

По каждому фронту входного сигнала таймер будет фиксировать текущее значение счётчика в специализированном регистре канала. Наша задача - сохранить результат первой фиксации *n1*, затем дождаться, пока истечёт интервал измерения, сохранить результат последней фиксации *n2* и общее количество фиксаций *c*. Тогда количество импульсов от первой до последней фиксации составит *n2 - n1*. Количество периодов входного сигнала между первой и последней фиксацией будет равно *c - 1* (так как количество отрезков разбиения на 1 меньше количества точек разбиения). Частоту сигнала вычисляем как

Если *c < 2*, значит, нам не удалось выделить полный период за выбранный интервал измерения (слишком низкая частота входного сигнала для данного интервала измерения).

1. Программа для частотомера на основе STM32

// Частотомер 1Гц..10МГц.

// Отлажено на MCU: STM32F100RB; кварц 8МГц; тактовая частота 24МГц.

// Вход прибора: PA1 вывод MCU (цифровой TTL/CMOS3.3V совместимый).

// (PA1 - вход канала TIM2\_CH2 или TIM1\_CH1).

#include "stm32f10x.h"

// Объявление функций для инициализации средств отображения и

// вывода результатов (определения функций - в отдельном файле).

**int** display\_init();

**int** display(uint32\_t fclk, uint32\_t m, uint32\_t n, uint32\_t err);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Глобальные переменные, используемые обработчиком прерывания

// TIM2\_IRQHandler для собственных нужд, а также для взаимодействия

// обработчика и основной программы.

// Определение этих переменных как глобальных упрощает выполнение их

// начальной инициализации, которая необходима перед каждым запуском

// измерения.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Предыдущее значение количества подсчитанных фронтов входного сигнала:

// сравнивая с текущим количеством (подсчитанных с начала измерения),

// обработчик TIM2\_IRQHandler определяет, были ли с момента предыдущего

// вызова обнаружены новые фронты входного сигнала.

// При каждом вызове обработчик обновляет переменную текущим значением.

**volatile** uint32\_t N1\_prev=0;

// Корректирующее значение для счётчика TIM1->CNT, учитывающее

// переполнения счётчика; действительное значение количества импульсов

// равно (N1\_correction + TIM1->CNT)

**volatile** uint32\_t N1\_correction = 0;

// Корректирующее значение для счётчика TIM2->CNT, учитывающее

// переполнения счётчика.

**volatile** uint32\_t N2\_correction = 0;

// Корректирующее значение счётчика для момента, когда фиксируется

// значение N2b\_naked (фиксация по первому фронту входного сигнала).

**volatile** uint32\_t N2b\_correction = 0;

// Значение счётчика TIM2->CNT в момент обнаружения первого фронта

// входного сигнала (соответствующая ему поправка на переполнения

// хранится в переменной N2b\_correction).

// Значение записывается в переменную системой DMA.

**volatile** uint32\_t N2b\_naked = 0;

// Количество учтённых фронтов сигнала за интервал измерения

// (истинное значение, с учётом переполнений счётчика).

**volatile** uint32\_t N1e = 0;

// Значение счётчика таймера TIM2 в момент последней фиксации (по

// последнему учитываемому фронту входного сигнала); скорректировано

// с учётом количества переполнений счётчика.

**volatile** uint32\_t N2e = 0;

// Флаг, устанавливаемый в true после того как в переменные N1e, N2e

// записываются результаты, полученные в процессе измерения

// (устанавливается в false перед началом измерения в функции measure).

**volatile** **bool** fN2e = **true**;

// Время, отводимое на измерение в тактах сигнала на счётном входе TIM2.

// За точную выдержку длительности интервала измерения отвечает канал 3

// таймера TIM2.

uint32\_t time\_slot = 0;

// Учёт переполнений счётчика в таймере TIM1.

**extern** "C" **void** TIM1\_UP\_TIM16\_IRQHandler()

{

**if**(TIM1->SR & TIM\_SR\_UIF)

{

TIM1->SR = ~TIM\_SR\_UIF;

N1\_correction += TIM1->ARR + 1; // По умолчанию 0x10000.

}

}

// Таймер TIM2 настроен таким образом, что прерывание возникает

// при переполнении счётчика и при достижении счётчиком половины

// переполняющего значения (за это отвечает канал 2).

// Это необходимо для правильного определения корректирующего значения

// в момент выполнения фиксации в канале 1 (для учёта переполнений).

// Обработчик прерываний от TIM2 выполняет основную работу по измерению.

// Вызывается при переполнении счётчика; "полупереполнении" и

// при срабатывании схемы сравнения в канале 3.

**extern** "C" **void** TIM2\_IRQHandler()

{

// Если отведённое время истекло, завершаем измерение.

**if**(TIM2->SR & TIM\_SR\_CC3IF)

{

TIM2->SR = ~TIM\_SR\_CC3IF; // Сброс флага.

uint32\_t t = N2\_correction + TIM2->CCR3;

// Учёт только что произошедшего переполнения.

**if**((TIM2->SR & TIM\_SR\_UIF) &&

TIM2->CCR3 < TIM2->CNT)

t += TIM2->ARR + 1;

**if**(t >= time\_slot)

{

// Отключаем канал 1 таймера (с этого момента прекращаются

// фиксации значения счётчика по фронту входного сигнала).

TIM2->CCER &= ~TIM\_CCER\_CC1E;

// Останавливаем таймеры.

TIM2->CR1 &= ~TIM\_CR1\_CEN;

TIM1->CR1 &= ~TIM\_CR1\_CEN;

// При следующем измерении необходимо разрешить генерацию

// запросов DMA от таймера, но перед этим её обязательно

// следует запретить (цикл запрет/разрешение обязателен, чтобы

// избежать ложного срабатывания канала DMA

// при последующем его включении).

TIM2->DIER &= ~TIM\_DIER\_CC1DE;

// Также отключаем канал DMA до следующего измерения

// (для нового измерения нужно будет перенастроить канал DMA,

// что возможно только при отключённом канале).

DMA1\_Channel5->CCR &= ~DMA\_CCR5\_EN;

}

**else** // Если время не истекло, выходим из обработчика, а

**return**; // если истекло - продолжаем завершающие действия.

}

// Маска для интересующих нас флагов

// (переполнение и сравнение в канале 2).

**const** uint16\_t mask = TIM\_SR\_UIF | TIM\_SR\_CC2IF;

// Запоминаем значение из регистра состояния таймера TIM2.

uint16\_t sr = TIM2->SR;

// Сразу же сбрасываем установленные обрабатываемые флаги.

TIM2->SR = ~(sr & mask);

// Если произошло переполнение счётчика, учитываем его.

**if**(sr & TIM\_SR\_UIF)

N2\_correction += TIM2->ARR + 1;

// Определяем текущее значение счётчика фиксаций в канале 1 таймера

// (подсчёт количества фиксаций ведёт счётчик таймера TIM1).

uint32\_t N1 = N1\_correction + TIM1->CNT;

// Корректируем текущее значение при обнаружении переполнения.

**if**(N1 < N1\_prev)

N1 += TIM1->ARR + 1;

// Проверяем наличие новых фиксаций с момента последнего выполнения

// данного обработчика.

**if**(N1 != N1\_prev)

{

// Если обнаружена первая фиксация, определяем и сохраняем

// корректирующее значение для неё.

**if**(N1\_prev == 0)

{

N2b\_correction = N2\_correction;

// Проверяем, не произошла ли эта фиксация до переполнения.

**if**((sr & TIM\_SR\_UIF) &&

N2b\_naked > TIM2->CNT)

N2b\_correction -= TIM2->ARR + 1;

}

// Любая фиксация может оказаться последней, поэтому всегда

// обновляем переменную N2e (при этом учитываем, что

// фиксация могла произойти и до последнего переполнения).

uint32\_t ccr = TIM2->CCR1;

N2e = N2\_correction + ccr;

**if**((sr & TIM\_SR\_UIF) &&

ccr > TIM2->CNT)

N2e -= TIM2->ARR + 1;

}

// Значение N1 становится "предыдущим"

// при следующем вызове обработчика.

N1\_prev = N1;

// Если канал 1 отключён, значит, измерение уже завершено и можно

// готовить результаты (счётчики остановлены, можно не опасаться

// несогласованности значений).

**if**(!(TIM2->CCER & TIM\_CCER\_CC1E))

{

// Количество учтённых фронтов входного сигнала.

N1e = N1\_correction + TIM1->CNT;

**if**(TIM1->SR & TIM\_SR\_UIF)

{

TIM1->SR = ~TIM\_SR\_UIF;

N1e += TIM1->ARR + 1;

}

// Устанавливаем флаг завершения измерения.

fN2e = **true**;

}

}

// Функция записывает значения в аргументы, передаваемые по ссылке:

// N1 - количество учтённых периодов входного сигнала в интервале

// измерения;

// N2 - количество импульсов опорного генератора (тактовых импульсов

// таймера TIM2) за N1 периодов входного сигнала;

// тогда частота входного сигнала f=f0\*N1/N2, где f0 - частота опорного

// генератора.

// Важно!

// Если измерение после вызова measure ещё не завершено, функция ждёт

// завершения измерения (ждёт установки флага fN2e).

// Возвращает код ошибки:

// 0 - без ошибок;

// -1 - не удалось измерить частоту, так как в интервал измерения

// попало менее двух фронтов сигнала (слишком низкая частота сигнала

// для выбранного интервала измерения).

**int** get\_result(uint32\_t &N1, uint32\_t &N2)

{

N1 = 0;

N2 = 0;

// Если процесс измерения ещё не завершён, ждём его завершения.

**while**(!fN2e) {}

// Слишком низкая частота для выбранного интервала измерения.

**if**(N1e < 2)

**return** -1;

// Количество периодов сигнала на 1 меньше количества подсчитанных

// фронтов.

N1 = N1e - 1;

// Количество тактов опорного генератора за N1 период входного

// сигнала равно разности между значениями счётчика таймера TIM2

// в момент последней и первой фиксации (с учётом переполнений).

N2 = N2e - (N2b\_correction + N2b\_naked);

**return** 0;

}

// Запустить процесс измерения.

// Функция не дожидается завершения измерения; запустив измерение в

// фоновом режиме, она сразу завершает своё выполнение.

// Само измерение происходит за счёт работы таймеров и их обработчиков

// прерываний.

// Если функция вызвана до завершения предыдущего измерения, она

// возвращает false; выполнение ранее запущенного измерения продолжается.

// ticks задаёт желаемый интервал измерения в тактах сигнала на счётном

// входе таймера TIM2.

**bool** measure(uint32\_t ticks)

{

// Ошибка, если предыдущее измерение ещё не завершилось.

**if**(!fN2e)

**return** **false**;

// Выполняем начальную инициализацию переменных.

time\_slot = ticks;

N1\_prev = 0;

N1\_correction = 0;

N2\_correction = 0;

N2b\_correction = 0;

N2b\_naked = 0;

N2e = 0;

fN2e = **false**;

// На всякий случай сбрасываем флаг фиксации в канале 1 таймера TIM2.

TIM2->SR = ~TIM\_SR\_CC1IF;

// Последовательность отключение/включение генерации запроса DMA

// сбрасывает те запросы, которые были сформированы при

// остановленном канале DMA.

TIM2->DIER |= TIM\_DIER\_CC1DE;

// Разрешаем счёт таймеру TIM1 (импульсы на счётный вход не будут

// поступать, пока не включен канал 1 таймера TIM2).

TIM1->CNT = 0;

TIM1->CR1 |= TIM\_CR1\_CEN;

// Готовим канал DMA к пересылке одного значения из TIM2->CCR1 в

// переменную N2b\_naked (основная инициализация DMA выполняется

// один раз при старте программы; при каждом измерении остаётся

// только указать количество пересылок и включить DMA).

DMA1\_Channel5->CNDTR = 1;

DMA1\_Channel5->CCR |= DMA\_CCR5\_EN;

// Запускаем с 0 счёт в таймере TIM2 (на счётный вход поступает

// тактовый сигнал микроконтроллера).

TIM2->CNT = 0;

TIM2->CCR2 = (TIM2->ARR + 1) >> 1; // По умолчанию 0x8000.

TIM2->CCR3 = ticks & 0xFFFF;

TIM2->CR1 |= TIM\_CR1\_CEN;

// Разрешаем работу каналов 1, 2 и 3 в таймере TIM2.

// В канале 1 фиксируется значение счётчика таймера по каждому

// фронту входного сигнала (а также при этом на 1 увеличивается

// счётчик в TIM1);

// канал 2 используется для генерации дополнительного прерывания

// посередине между переполнениями;

// канал 3 служит для своевременного завершения измерения.

TIM2->CCER |= TIM\_CCER\_CC1E | TIM\_CCER\_CC2E | TIM\_CCER\_CC3E;

**return** **true**;

}

// Начальная инициализация используемых для измерения частоты

// периферийных устройств микроконтроллера.

**void** fm\_init()

{

// Настраиваем приоритеты прерываний и разрешаем их обработку

// (предполагается, что прерывание от TIM2 будет более

// высокоприоритетным).

NVIC\_SetPriority(TIM1\_UP\_TIM16\_IRQn, 1);

NVIC\_SetPriority(TIM2\_IRQn, 0);

NVIC\_EnableIRQ(TIM1\_UP\_TIM16\_IRQn);

NVIC\_EnableIRQ(TIM2\_IRQn);

// Включаем тактовый сигнал для используемых устройств

// (DMA, TIM1, TIM2, GPIOA).

RCC->AHBENR |= RCC\_AHBENR\_DMA1EN;

RCC->APB1ENR |=

RCC\_APB1ENR\_TIM2EN;

RCC->APB2ENR |=

RCC\_APB2ENR\_TIM1EN |

RCC\_APB2ENR\_IOPAEN;

// Настраиваем вход PA1 (TIM2\_CH2) (как цифровой вход с подтяжкой).

GPIOA->CRL =

GPIOA->CRL & ~0xF0|

0x80;

// Настраиваем таймер TIM2:

// таймер тактируется от шины (источник опорного

// сигнала - системный тактовый сигнал);

// по переднему фронту исследуемого сигнала производим

// фиксацию состояния счётчика таймера;

// сигнал захвата в канале 1 таймера используем в качестве TRGO.

TIM2->ARR = 0xFFFF;

TIM2->CR2 |= 0x3 << 4; // The trigger output (TRGO) send

// a positive pulse when the CC1IF flag is to be set.

TIM2->CCMR1 = 2; // CC1 channel is configured as input, IC1 is mapped on TI2.

TIM2->DIER |=

TIM\_DIER\_CC1DE | // DMA запрос при фиксации в канале 1.

TIM\_DIER\_UIE | // Прерывание при переполнении.

TIM\_DIER\_CC2IE | // Прерывание, если CNT==CCR2.

TIM\_DIER\_CC3IE; // Прерывание, если CNT==CCR3.

// Базовая настройка DMA для сохранения первого результата фиксации.

// TIM2\_CH1 - DMA1 Ch5

// Сброс значимых битов регистра CCR.

DMA1\_Channel5->CCR &= ~0x7FFF;

// Устанавливаем нужные биты.

// PL[1:0].MSIZE[1:0].PSIZE[1:0]=11.10.01

// (priority level = very high;

// MSIZE=32bits, PSIZE=16bits).

DMA1\_Channel5->CCR |= 0x39 << 8;

DMA1\_Channel5->CMAR = (uint32\_t)&N2b\_naked;

DMA1\_Channel5->CPAR = (uint32\_t)&TIM2->CCR1;

// Таймер TIM1 будет подсчитывать количество захватов в TIM2.

TIM1->ARR = 0xFFFF;

TIM1->SMCR |= (1 << 4) | 0x7; //TS=1 (ITR1 selected - TRGO of TIM2); SMS=bx111.

TIM1->DIER |= TIM\_DIER\_UIE; // Прерывание по переполнению.

}

**int** main(**void**)

{

// Инициализируем частотомер.

fm\_init();

// Инициализируем средство отображения информации.

display\_init();

// Готово. Можно приступать к измерениям частоты.

**while**(**true**)

{

// Запуск очередного измерения, интервал измерения 3 с.

measure(72000000);

// Ожидание завершения измерения, считывание результата и

// его отображение.

uint32\_t N1, N2;

**int** err = get\_result(N1, N2);

display(SystemCoreClock, N1, N2, err);

}

}

1. Комментарии к программе
   1. Определение переменных

volatile uint32\_t N1\_prev;

Вообще, переменные с именами вида N1xxx используются для сохранения результатов счёта таймера TIM1 в определённые моменты времени (соответственно, переменные N2xxx относятся к результатам счёта TIM2). В частности, N1\_prev используется для хранения значения таймера TIM1, считанного при предыдущем выполнении обработчика прерывания от TIM2. Переменная служит для того, чтобы обработчик TIM2\_IRQHandler мог определить, происходили ли фиксации в канале TIM2 (которые и подсчитывает TIM1) с момента предыдущего считывания значения счётчика TIM1 (во время предыдущего вызова этого обработчика). Для определения, считываем текущее значение счётчика, сравниваем с предыдущим из N1\_prev. Если значения отличаются - были фиксации, которые нужно учесть в обработчике. Если не отличаются - фиксаций не было. После выполнения проверки, присваиваем переменной N1\_prev уже считанное текущее значение, которое станет "предыдущим" при следующем вызове обработчика.

Естественно, при определении текущего результат счёта таймера TIM1, требуется учёт переполнений счётчика. Выполняется это просто: если текущее значение больше предыдущего, то переполнения не было и значение корректно; если текущее значение меньше - переполнение было и требуется корректировка на коэффициент пересчёта счётчика в таймере TIM1, т.е. на TIM1->ARR + 1 (по умолчанию это 0xFFFF+1). Благодаря такому алгоритму, при выполнении в обработчике фрагмента

uint32\_t N1 = N1\_correction + TIM1->CNT;

который считывает текущее (истинное) значение счётчика в TIM1, совершенно не важно, успел ли обработчик переполнения от TIM1 обновить переменную N1\_correction или не успел - в любом случае, мы обнаружим неучтённое переполнение и при необходимости скорректируем значение.

Теперь о том, зачем такие сложности с обнаружением фиксаций, если, как известно, существует флаг, устанавливаемый аппаратно при выполнении фиксации (TIM2->SR, бит TIM\_SR\_CC1IF). Дело в том, что, во-первых, при выполнении первой фиксации, происходит пересылка данных средствами DMA и DMA самостоятельно сбрасывает бит. Это уже требует введения дополнительных программных проверок, кроме проверки флага. Во-вторых, между чтением флага и его сбросом (для обнаружения последующих фиксаций), может произойти ещё одна (или даже не одна) фиксация. Но мы сбросим флаг, и если после сброса не будет фиксаций, при последующем вызове обработчика, проверив флаг, ошибочно придём к выводу, что фиксаций не было. Если это происходит в конце измерения (для последней фиксации), она не будет должным образом скорректирована. Это очень серьёзная ошибка, сильно искажающая результат, но возникающая не всегда, а потому трудно выявляемая.

В предлагаемом же варианте, фактически мы каждый раз вычисляем "флаг" фиксации, поэтому пропуски невозможны: мы сначала считываем текущее количество фиксаций и запоминаем, а затем приступаем к обработке фактически последней. Возможна ситуация, когда после считывания счётчика TIM1 и обнаружения неучтённых фиксаций, происходят новые фиксации, и мы обрабатываем результат последней из них. Тем не менее, при следующем вызове обработчика TIM2\_IRQHandler мы опять обнаружим наличие необработанных фиксаций, хотя фактически обработка была выполнена. Но повторная обработка не представляет никакой опасности, она не изменяет результата; важно, что нет пропусков события.

volatile uint32\_t N1\_correction;

Переменная содержит поправку для значения счётчика TIM1, учитывающая переполнения счётчика (т.е. при переполнении мы увеличиваем переменную сразу на коэффициент пересчёта счётчика); истинное значение количества подсчитанных импульсов к данному моменту будет равно

N1\_correction + TIM1->CNT;

volatile uint32\_t N2\_correction;  
Поправка для таймера счётчика TIM2; истинное значение количества импульсов, подсчитанных таймером, составляет

N2\_correction + TIM2->CNT;

volatile uint32\_t N2b\_correction;

Переменная содержит корректирующее значение для результата первой фиксации (корректирующее значение для счётчика в TIM2 в тот момент, когда происходит первая фиксация). Переменная необходима, потому как при низкочастотном сигнале счётчик может успеть переполниться до того, как произойдёт первая фиксация.

volatile uint32\_t N2b\_naked;

В эту переменную система DMA помещает результат первой фиксации, т.е. значение счётчика таймера в момент, соответствующий первому фронту входного сигнала. Значение должно быть скорректировано с учётом переполнений счётчика. Истинное значение составит

N2b\_correction + N2b\_naked;

volatile uint32\_t N1e;

Когда измерение завершается, обработчик прерывания TIM2\_IRQHandler помещает в эту переменную подсчитанное за интервал измерения количество фронтов входного сигнала (иначе говоря, количество фиксаций от первой учтённой до последней, включая первую и последнюю). Это значение на 1 больше периодов входного сигнала, образующих истинный интервал измерения.

volatile uint32\_t N2e;

Количество импульсов опорного генератора, подсчитанных таймером TIM2 на момент последней учтённой фиксации (верное значение, с учётом переполнений).

volatile bool fN2e;

Присваивая этой переменной значение **true**, обработчик прерывания TIM2\_IRQHandler тем самым сообщает основной программе, что измерение завершено и результаты готовы для считывания.

uint32\_t time\_slot;

Переменная содержит заданное время измерения в тактах сигнала на счётном входе TIM2.

* 1. Определение функций

extern "C" void TIM1\_UP\_TIM16\_IRQHandler();

Функция является обработчиком прерываний от таймера TIM1, она просто учитывает переполнения счётчика таймера, обновляя значение глобальной переменной N1\_correction. Заметим, что переменная содержит не количество переполнений, а корректирующее значение, сложив которое с текущим значением счётчика, сразу получим истинное значение результата счёта. Так как переполняющее счётчик значение равно TIM1->ARR + 1, то при каждом переполнении для обновления переменной выполняется код

N1\_correction+=TIM1->ARR + 1;

extern "C" void TIM2\_IRQHandler();

Функция является обработчиком прерываний от таймера TIM2, на эту функцию возложена основная работа по измерению частоты входного сигнала; измерение идёт в фоновом режиме, поэтому одновременно с ним микроконтроллер может заниматься другими делами.

При настройке NVIC для прерываний от TIM2 задаётся более высокий приоритет, чем для прерываний от TIM1 для того, чтобы при переполнении счётчика таймера TIM1, не прерывалась работа обработчика TIM2\_IRQHandler.

Обработчик TIM2\_IRQHandler ведёт учёт переполнений счётчика таймера TIM2, обновляя переменную N2\_correction. Но это лишь малая часть того, что делает обработчик. Самая главная его задача - отслеживать уже произошедшие в канале 1 таймера фиксации и сопоставлять соответствующие им корректирующие значения, учитывающие переполнения счётчика. Также обработчик следит за временем, прошедшим с начала измерения и останавливает измерение, когда заданное время истекает. Время может отмерять отдельный таймер, но здесь эту функцию выполняет третий канал, который работает в режиме сравнения. Если следует остановить измерение после ticks импульсов на счётном входе таймера, то с учётом переполнений, понятно, что в этот момент в счётчике будет значение ticks % 0x10000 (остаток от деления ticks на модуль пересчёта, здесь оно также оказывается равным ticks & 0xFFFF). Поэтому проверять, не истекло ли время, следует всякий раз, когда значение в счётчике становится равным ticks & 0xFFFF. Для этого в регистр сравнения канала 3 таймера, перед запуском измерения, помещаем значение ticks & 0xFFFF и, обрабатывая прерывание от канала, сравниваем прошедшее время (в тактах) с заданным.

Если время на измерение вышло, останавливаем счёт: сначала отключаем канал 1 таймера, тем самым запрещая дальнейшее выполнение фиксаций в канале по фронтам входного сигнала, после чего можем остановить счётчики таймеров (с этого момента можем быть уверены, что больше переполнения счётчиков происходить не будут), обработать результаты последней фиксации и установить флаг завершения измерения (присвоить **true** глобальной переменной fN2e).

Как уже отмечалось, самая важная задача обработчика - определять корректирующие значения (учитывающие переполнения) для первой и последней фиксации. Корректирующие значения определяются всякий раз, когда обнаруживается, что с момента последнего вызова обработчика изменилось количество фронтов, подсчитанных счётчиком TIM1. Смотрим, если прерывание произошло по "полупереполнению", то, несомненно, последней фиксации соответствует текущее корректирующее значение. Если обрабатывается прерывание по переполнению, уточняем, произошла ли фиксация до переполнения или уже после (по величине зафиксированного значения в сравнении с текущим значением счётчика). Если фиксация была после переполнения, за корректирующее значение для фиксации выбираем текущее корректирующее значение, иначе - предыдущее (до переполнения). Определив корректирующее значение, складываем его с результатом последней фиксации и сохраняем результат в глобальной переменной N2e. Измерение может быть остановлено в любой момент, и последняя учтённая фиксация может оказаться вообще последней на интервале измерения. Но регулярное обновление N2e гарантирует, что переменная будет содержать актуальное значение.

Корректирующее значение первой фиксации определяется аналогичным образом (поскольку первая фиксация была выполнена между предыдущим вызовом обработчика и данным). Разумеется, это делается только один раз за измерение. Обработчик обнаруживает то, что первая фиксация произошла по выполнению условия  
N1\_prev == 0 && N1 != 0.  
Для первой фиксации корректирующее значение и нескорректированный результат фиксации сохраняются в отдельные переменные (нескорректированный результат фиксации сохраняется с помощью DMA).

int get\_result(uint32\_t &N1, uint32\_t &N2);

Функция помещает в свои аргументы, передаваемые по ссылке:  
N1 - количество периодов входного сигнала, образующих действительный интервал измерения;  
N2 - количество импульсов эталонного генератора (в нашем случае - тактового сигнала микроконтроллера) за этот интервал, образованный N1 периодами входного сигнала.

Возвращаемое значение:

0 - в случае успешного завершения;

-1 - в случае ошибки, когда в заданный интервал измерения не "поместился" даже один период входного сигнала (период, ограниченный двумя последовательными фронтами); это означает, что выбранный интервал слишком мал для измерения столь низкой частоты.

Если измерение не завершено, функция ожидает его завершения для получения результатов.

Значения N1, N2 функция определяет по результатам работы обработчика прерываний от TIM2, которые помещаются обработчиком в глобальные переменные.  
N1 - это просто уменьшенный на 1 результат счёта таймера TIM1 с учётом поправки на переполнения (счётчик подсчитывает фронты сигнала; количество периодов, заключённых между первым и последним фронтом будет на единицу меньше; учёт начального значения не требуется, поскольку счётчик изначально устанавливается на 0).  
N2 определяется как разность между результатами последней и первой фиксации в канале 1 таймера TIM2 (с учётом переполнений).

bool measure(uint32\_t ticks);

Функция запускает новое измерение (если в данный момент не происходит ещё не завершившееся измерение) и сразу завершается, освобождая микроконтроллер для выполнения других вычислений, пока идёт измерение. Для запуска измерения, функция выполняет инициализацию глобальных переменных, используемых в процессе работы обработчиков прерываний от таймеров; выполняет те действия по инициализации оборудования, которые требуется выполнять каждый раз заново (например, каждый раз нужно задавать количество передаваемых с помощью DMA данных); разрешает счёт для таймеров; разрешает работу каналов таймера TIM2. С момента включения каналов TIM2 измерение начинается, а функция возвращает управление вызвавшей программе.

void fm\_init();

Функция отвечает за начальную инициализацию периферийных устройств микроконтроллера, задействованных в измерении частоты. Вызывается однократно, до первого измерения. Например, в начале основной функции main.

int main();

Основная функция программы. Выполняет начальную инициализацию частотомера путём вызова fm\_init, затем приступает к циклическому измерению частоты с выводом результатов каждого измерения на дисплей.

int dispaly\_init();

int display(uint32\_t fclk, uint32\_t m, uint32\_t n, uint32\_t err);

Функции для начальной инициализации индикатора и вывода на него результатов измерений. Здесь они только объявлены, но не определены. Определяются функции в отдельном файле; их реализация зависит от используемого индикатора. Можно выводить результат на семисегментный индикатор достаточной разрядности; текстовый или графический дисплей с интерфейсом SPI или I2C; можно передавать данные на компьютер, например, через UART.