CAN-шина и stm32 - часть вторая - фильтры и два CAN

STM32

Здравствуйте.

Первая часть закончилась на том, что я обещал рассказать про настройку фильтров и работу двух CAN-интерфейсов на одном камне. Начнём с фильтров.

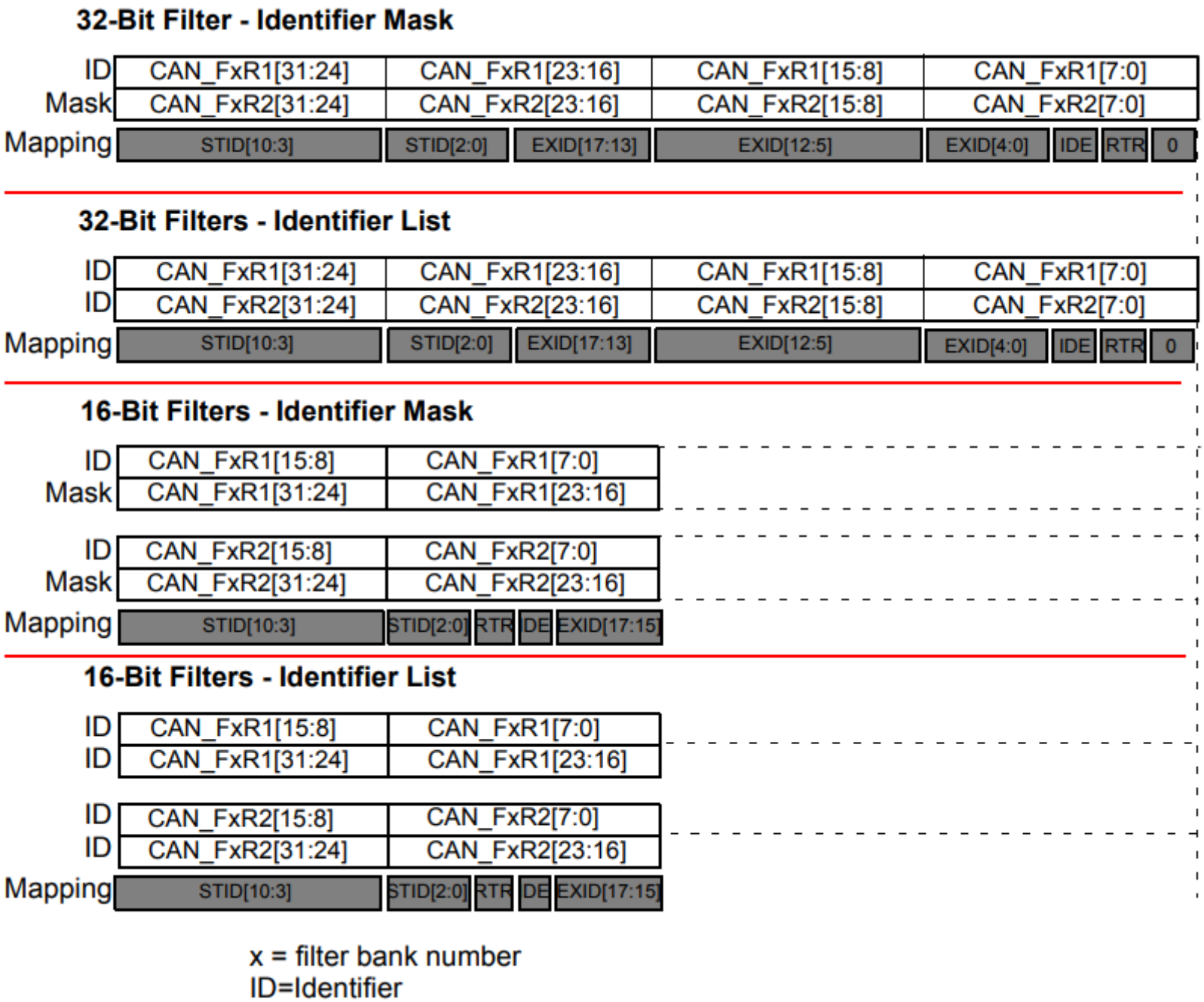
Фильтры

Как уже говорилось ранее, у каждого CAN-интерфейса есть 14 фильтров. *Фильтры так же называются фильтрами-банками или просто банками.* Каждый фильтр имеет порядковый номер *(счёт идёт он нуля)*, и представляет из себя два 32-х битных регистра *(далее буду называть их «регистры фильтра», первый и второй)*. *Каждый фильтр имеет свои «регистры фильтра».*

В двух словах, фильтр работает так: в первый «регистр фильтра» мы прописываем некий идентификатор пакета, а во второй некую маску, в результате чего CAN будет принимать только некоторый диапазон кадров, а все остальные отбрасывать. Это первый способ. Второй способ: в один или оба «регистра фильтра» мы прописываем конкретные идентификаторы кадров, которые хотим получать — всё остальное будет отброшено.

Фильтр нельзя настроить как исключающий, то есть если нужно принимать все кадры кроме какого-то одного, например 0x0296, придётся настроить два фильтра, так чтобы они принимали кадры с 0x0000 по 0x0295, и с 0x0297 по 0x07FF.

Для гибкой настройки фильтрации есть четыре варианта использования «регистров фильтра» для каждого фильтра. Картинка из референс мануала иллюстрирует это…



Красными полосками я разделил варианты использования «регистров фильтра».

Первый и второй варианты подходят для фильтрации как стандартных *(11 бит)*, так и расширенных *(29 бит)* идентификаторов, а

третий и четвёртый только для стандартных. Теперь давайте рассмотрим всё это подробно.

Первый вариант — 32-Bit Filter — Identifier Mask

Заранее предупреждаю — будет сложно, так что приготовьтесь 🤔

В первый «регистр фильтра» *(ID на картинке)* записываться идентификатор кадра, а во второй *(Mask на картинке)* маска, которая будет накладываться на этот идентификатор и сравниваться с этим идентификатором. *Выглядит запутано, но не пугайтесь, ниже всё разъяснится.* Таким образом фильтр будет содержать определённый диапазон идентификаторов, которые будут приняты *(не отброшены)*.

Поскольку наш CAN-интерфейс принимает все кадры, а потом уже аппаратно отбрасывает те, которые не соответствуют настройкам фильтра, то далее по тексту под словом «принимать» я буду подразумевать, что кадр не был отброшен.

Для примера будем настраивать фильтр №0.

Указываем номер фильтра…

```
sFilterConfig.FilterBank = 0;
```

Указываем режим работы фильтра…

```
sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
```

Режим идентификатора и маски. Обращаю ваше внимание на этот параметр, если записать сюда другой макрос (CAN_FILTERMODE_IDLIST), тогда фильтр будет работать по другому. Поскольку эти макросы схожие, будьте внимательны.

Указываем масштаб *(размерность)* фильтра…

```
sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
```

32 бита, говорит о том, что фильтроваться могут либо стандартные (11 бит) идентификаторы, либо расширенные (29 бит).

Далее, в прошлой части мы писали такой код…

```
sFilterConfig.FilterIdHigh = 0x0000;      // старшая часть первого "регистра фильтра"
sFilterConfig.FilterIdLow = 0x0000;       // младшая часть первого "регистра фильтра"

sFilterConfig.FilterMaskIdHigh = 0x0000;  // старшая часть второго "регистра фильтра"
sFilterConfig.FilterMaskIdLow = 0x0000;   // младшая часть второго "регистра фильтра"
```

Это два «регистра фильтра», каждый из которых условно разделён на старшую и младшую часть. *Везде записаны нули чтоб принимать все кадры.*

Сейчас мы настроим фильтр для работы с идентификаторами стандартных кадров, то есть с 11-битными идентификаторами.

Не смотря на то, что размерность фильтра у нас указана 32 бита (CAN_FILTERSCALE_32BIT), мы можем настраивать фильтр для работы и со стандартными (11 бит), и с расширенными (29 бит) идентификаторами. Если бы мы указали размерность фильтра 16 бит (CAN_FILTERSCALE_16BIT), тогда фильтр работал бы только со стандартными кадрами, об этом речь пойдёт ниже.

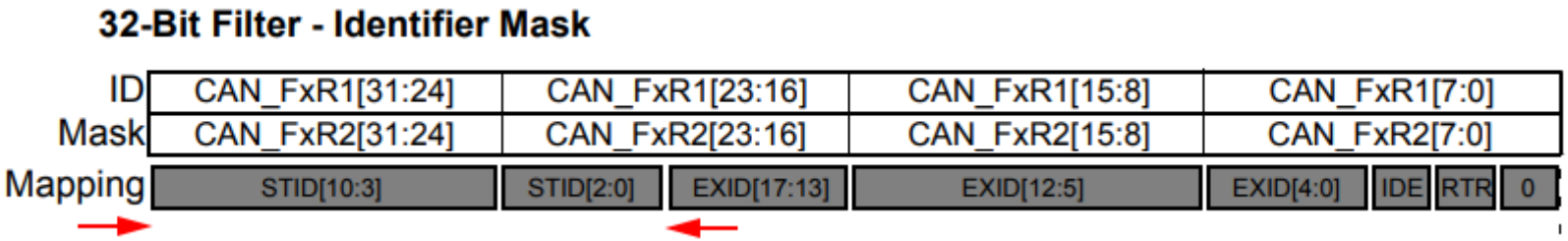
Предположим что мы хотим принимать идентификаторы с **0x0100** по **0x0107**, тогда в старшую часть первого «регистра фильтра» записываем начальный идентификатор, со сдвигом…

```
sFilterConfig.FilterIdHigh = 0x0100<<5; // старшая часть первого "регистра фильтра"
sFilterConfig.FilterIdLow = 0x0000;     // младшая часть первого "регистра фильтра"
```

В младшую часть пишем нули.

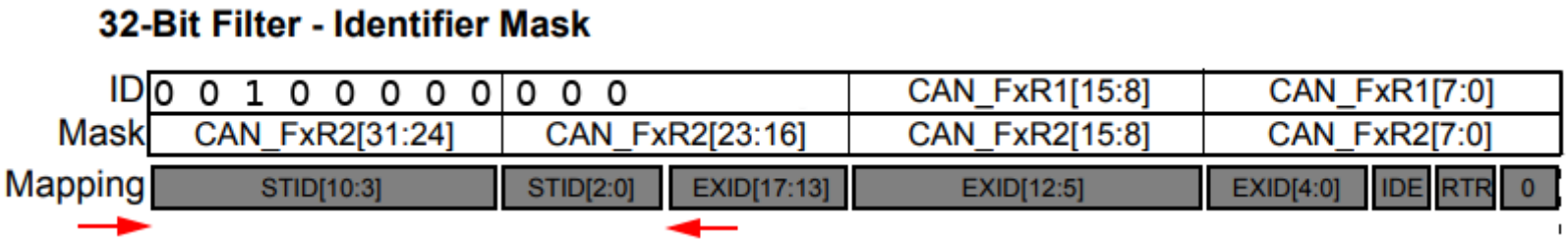
Сдвиг делается потому, что идентификатор у нас 11-ти битный, и согласно рисунку для него выделена старшая часть, старшей части «регистра фильтра». *Такое вот получилось «масло масляное».* 🙂

Вот эта область…



Mapping показывает что здесь должен лежать стандартный идентификатор — STDID — восемь бит в первом октете (слева) и три во втором.

То есть получилось так…



*В бинарном формате число **0x0100** выглядит так — **0000000100000000**, а поскольку количество стандартных идентификаторов не может превышать **0x07FF** (0000011111111111), то первые пять нулей никому не нужны, поэтому «выкидываются на мороз».*

В целом, все эти картинки с распределением ноликов и единичек вам не так уж и нужны, однако для общего развития и упрощения понимания будут полезны.

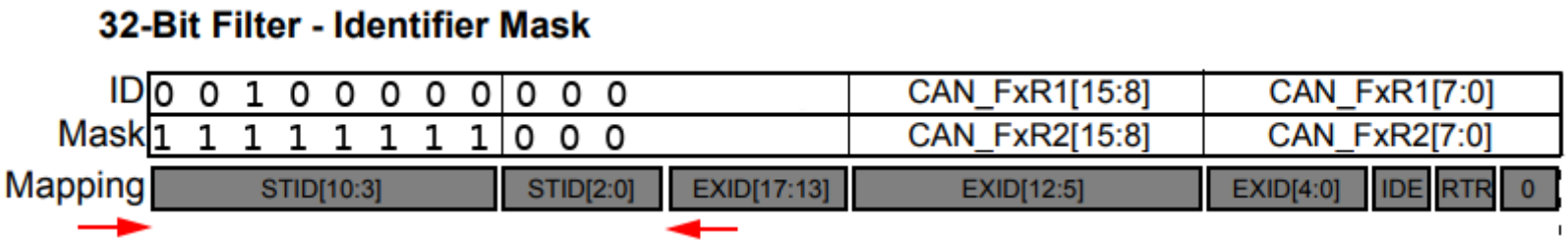
Переходим к старшей части второго «регистра фильтра». Сюда мы записываем *(опять же со сдвигом)* число **0x07F8**…

```
sFilterConfig.FilterMaskIdHigh = 0x07F8<<5; // старшая часть второго "регистра фильтра"
sFilterConfig.FilterMaskIdLow = 0x0000;     // младшая часть второго "регистра фильтра"
```

Вот это поворот, скажет неискушенный читатель, наверное автор что-то перепутал. Но нет, всё правильно, и сейчас будем с

этим разбираться.

В младшую часть второго «регистра фильтра» мы опять записали нолик, а число 0x07F8 *(0000011111111000)* легло у нас в старшую часть второго «регистра фильтра» вот так…



Первые (слева) пять ноликов опять же выкинуты, по описанным выше причинам.

Таким образом, число 0x07F8 у нас является маской. *Немного забегая вперёд, скажу что это число нужно подбирать самостоятельно, экспериментальным путём.* Теперь давайте разбираться как это работает.

Теперь смотрите что получается. Когда прилетает какой-то идентификатор, он попадает в систему фильтрации, и происходит следующее: между прилетевшим идентификатором и маской выполняется побитовая операция «И» *(она же AND, она же &)*, и далее этот результат сравнивается с тем, что записано в «первом регистре» фильтра, в нашем случае это число **0x0100**. Если результат совпадает, тогда идентификатор *(кадр с этим идентификатором)* принимается, если нет — отбрасывается.

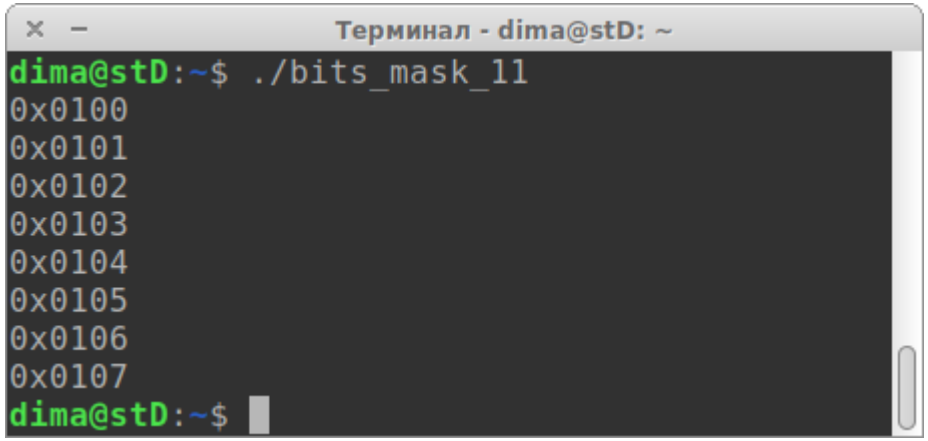
Чтоб внести ясность можно описать этот механизм простой программой…

```c
#include <stdio.h>
#include <stdint.h>

int main()
{
    uint32_t ID = 0x0100;
    uint32_t Mask = 0x07F8;

    for(uint32_t i = 0; i < 0x0800; i++)
    {
        if((i & Mask) == ID)
        {
            printf("0x%04X\n", i);
        }
    }

    return 0;
}
```

ID — это то, что мы записали в первый «регистр фильтра».
Mask — маска, записанная во второй «регистр фильтра».

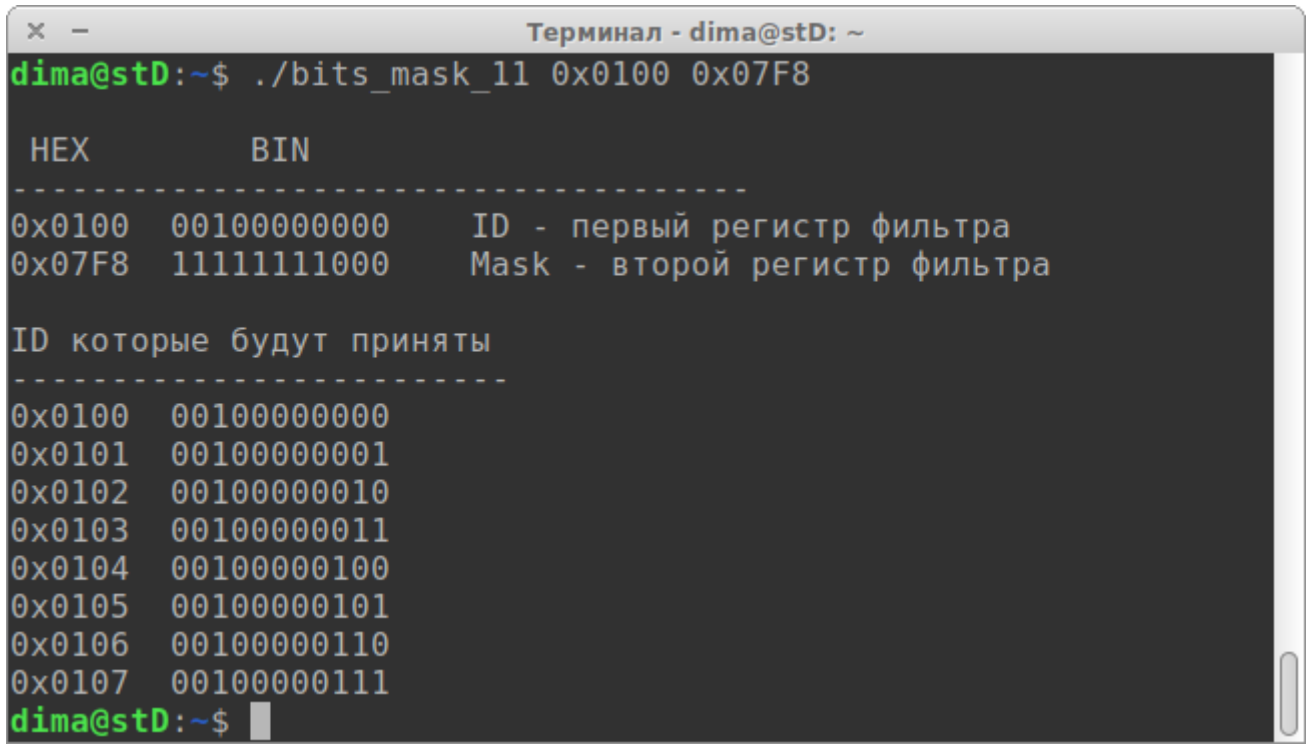
Цикл for имитирует прилетающие идентификаторы, от 0 до 0x07FF *(0x0800 — 1, максимальное кол-во стандартных идентификаторов)*. В условии if на прилетевший идентификатор накладывается маска, и если полученный результат равен 0x0100, прилетевший идентификатор будет принят CAN'ом *(выведен на печать)*.

Результатом работы программы будет вот такой вывод…



Идентификаторы с **0x0100** по **0x0107**, которые мы планировали принимать.

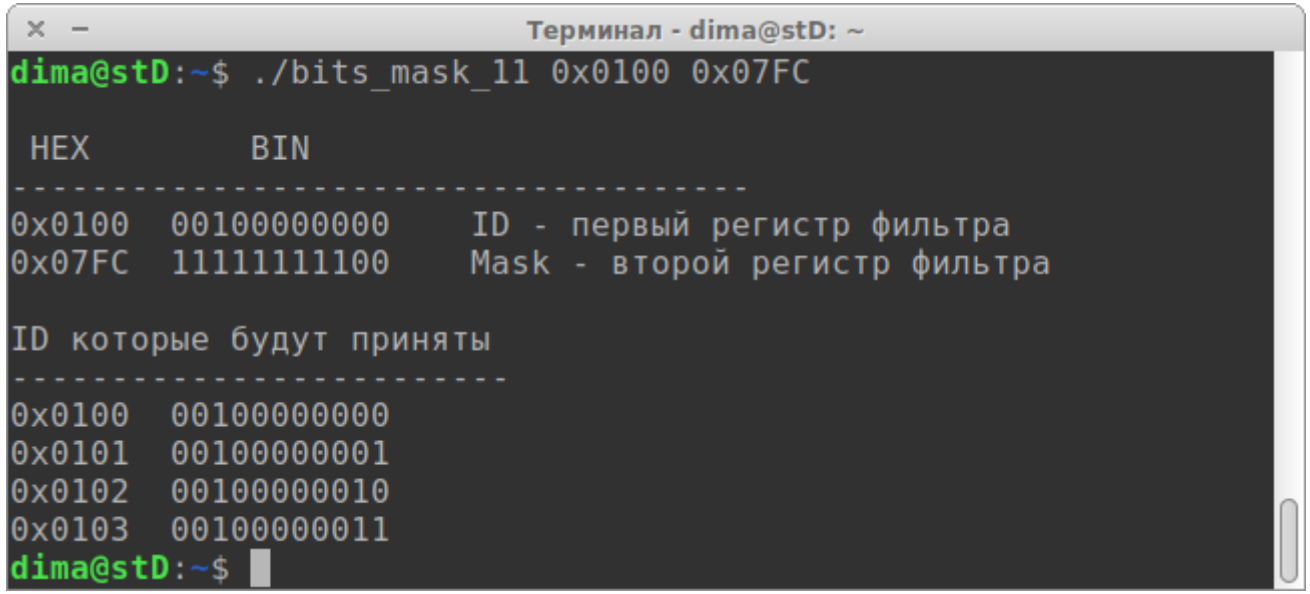
По ссылке можно качнуть программу, которая выводит значения в HEX и бинарном формате…



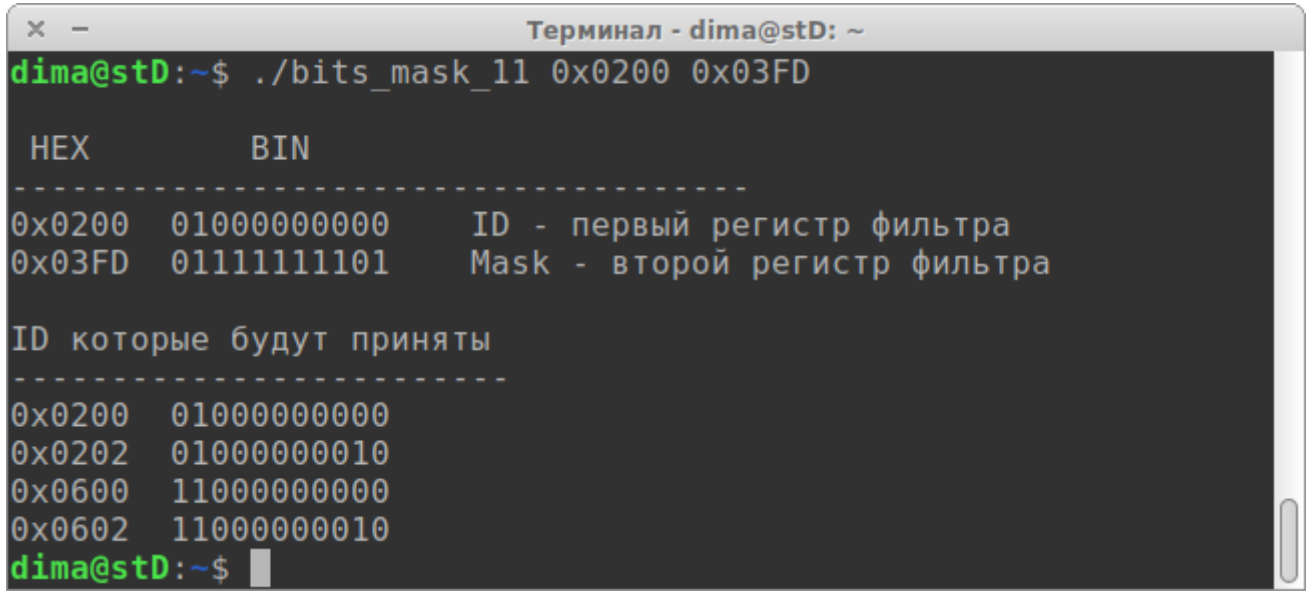
С её помощью можно методом «тыка» подбирать маску. *Программа написана для Линукса, но должна скомпилиться и для Windows.*

Либо просто перенесите её на stm.

Если мы изменим в маске последнюю цифру с 0x07F8 на 0x07FC, то наш CAN будет принимать идентификаторы с 0x0100 по 0x0103…



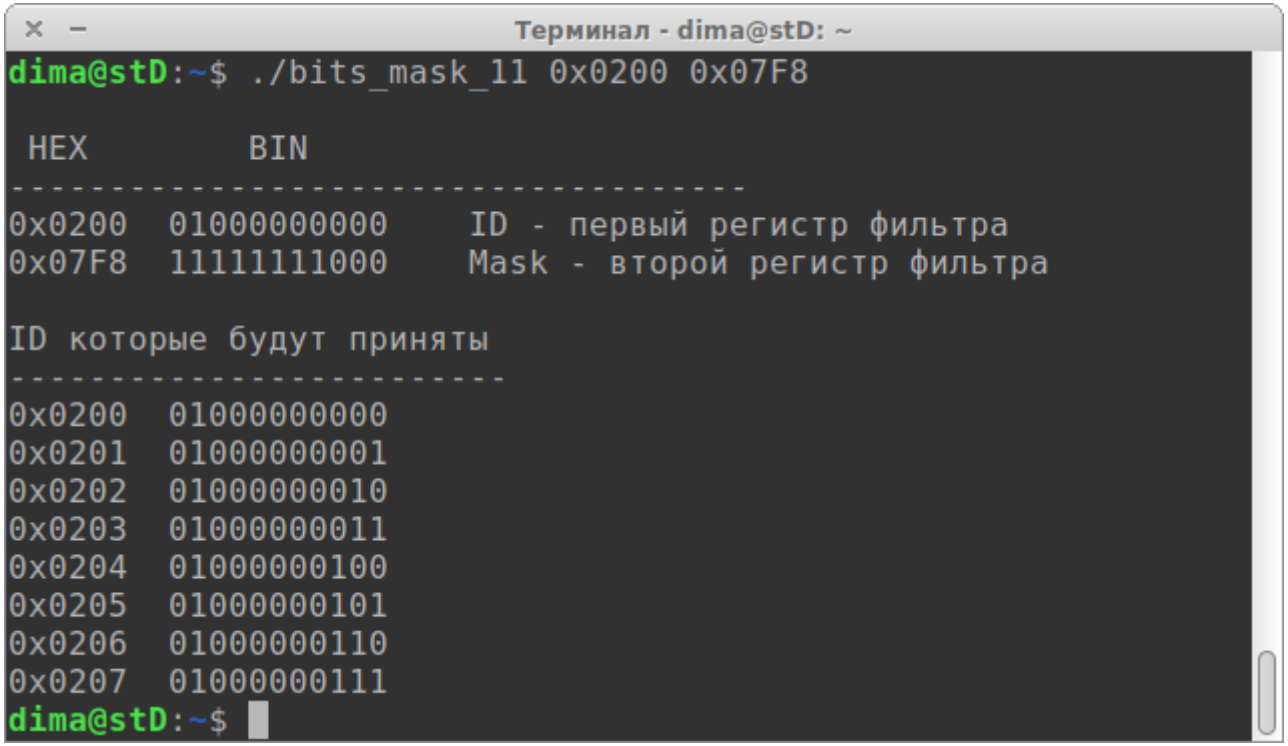
Совсем не обязательно настраивать фильтр так, чтобы принимаемые идентификаторы были последовательны в порядке возрастания, можно делать как угодно, всё зависит от ID и Mask. Например если в ID записать 0x0200, а маску 0x03FD, тогда будут приниматься идентификаторы 0x0200, 0x0202, 0x0600 и 0x0602…



Чтобы понять как подбирать маску взгляните на последнюю картинку. Если бит в маске равен единице значит и соответствующий бит в прилетевшем идентификаторе тоже должен быть равен единице. Если же бит в маске равен нулю, тогда не важно чему равен соответствующий бит в прилетевшем идентификаторе.

Первый бит в маске (слева направо) равен нулю, соответственно не важно чему равен первый бит у идентификаторов, которые будут приняты, может быть ноль или единица. Второй бит в маске равен единице, поэтому принимаются только те идентификаторы, у которых второй бит равен единице. Десятый бит в маске равен нулю, значит не важно чему равен этот бит у идентификаторов, которые будут приняты.

Для закрепления понимания давайте сделаем маску, у которой все биты кроме трёх последних будут равны единице…



Вуаля. Теперь мы принимаем идентификаторы в диапазоне с 0x0200 по 0x0207. Обратите внимание, что маска такая же как и для идентификаторов с 0x0100 по 0x0107. Это ничего не значит, просто к слову.

В общем думаю более менее всё понятно. Берёте какой-нибудь идентификатор, придумываете маску, скармливаете это программе и смотрите какие идентификаторы будут приняты.

Теперь давайте мысленно вернёмся к началу статьи, вспомним что мы хотели принимать идентификаторы с **0x0100** по **0x0107**, и закончим настройку фильтра №0.

В результате у нас получится следующий код…

```
sFilterConfig.FilterBank = 0;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
sFilterConfig.FilterIdHigh = 0x0100<<5; // старшая часть первого "регистра фильтра"
sFilterConfig.FilterIdLow = 0x0000;     // младшая часть первого "регистра фильтра"
sFilterConfig.FilterMaskIdHigh = 0x07F8<<5; // старшая часть второго "регистра фильтра"
sFilterConfig.FilterMaskIdLow = 0x0000;     // младшая часть второго "регистра фильтра"
sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;
sFilterConfig.FilterActivation = ENABLE;

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}
```

Последний параметр просто включает фильтр, а вот про предпоследний нужно сказать пару слов. Как вы помните у CAN'а есть два приёмных буфера CAN_RX_FIFO0 и CAN_RX_FIFO1. В данном случае мы указали CAN_RX_FIFO0, это означает что все принятые кадры прошедшие через фильтр №0 попадут CAN_RX_FIFO0. Отсюда вытекает, что настроив несколько фильтров, мы можем указать в какой именно буфер будут сваливаться принятые кадры. Теперь давайте настроим ещё один фильтр.

Возьмём последний пример и будем принимать идентификаторы с **0x0200** по **0x0207** через фильтр №1…

```
sFilterConfig.FilterBank = 1;
sFilterConfig.FilterIdHigh = 0x0200 << 5;     // старшая часть первого "регистра фильтра"
sFilterConfig.FilterIdLow = 0x0000;           // младшая часть первого "регистра фильтра"
sFilterConfig.FilterMaskIdHigh = 0x07F8 << 5; // старшая часть второго "регистра фильтра"
sFilterConfig.FilterMaskIdLow = 0x0000;       // младшая часть второго "регистра фильтра"

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}
```

Указываем только номер фильтра и прописываем значения в «регистры фильтра». Все остальные настройки будут автоматически взяты из настроек фильтра №0.

Таким образом инициализация CAN'а и фильтров будет выглядеть так…

```
...
/* USER CODE BEGIN CAN_Init 0 */
CAN_FilterTypeDef  sFilterConfig;
/* USER CODE END CAN_Init 0 */

hcan.Instance = CAN1;
hcan.Init.Prescaler = 4;
hcan.Init.Mode = CAN_MODE_NORMAL;
hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;
hcan.Init.TimeSeg1 = CAN_BS1_13TQ;
hcan.Init.TimeSeg2 = CAN_BS2_2TQ;
hcan.Init.TimeTriggeredMode = DISABLE;
hcan.Init.AutoBusOff = ENABLE;
hcan.Init.AutoWakeUp = DISABLE;
hcan.Init.AutoRetransmission = ENABLE;
hcan.Init.ReceiveFifoLocked = DISABLE;
hcan.Init.TransmitFifoPriority = ENABLE;

if (HAL_CAN_Init(&hcan) != HAL_OK)
{
   Error_Handler();
}

/* USER CODE BEGIN CAN_Init 2 */
sFilterConfig.FilterBank = 0;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
sFilterConfig.FilterIdHigh = 0x0100<<5; // старшая часть первого "регистра фильтра"
sFilterConfig.FilterIdLow = 0x0000;     // младшая часть первого "регистра фильтра"
sFilterConfig.FilterMaskIdHigh = 0x07F8<<5; // старшая часть второго "регистра фильтра"
sFilterConfig.FilterMaskIdLow = 0x0000;     // младшая часть второго "регистра фильтра"
sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;
sFilterConfig.FilterActivation = ENABLE;

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}

sFilterConfig.FilterBank = 1;
```

```
sFilterConfig.FilterIdHigh = 0x0200 << 5;     // старшая часть первого "регистра фильтра"
sFilterConfig.FilterIdLow = 0x0000;           // младшая часть первого "регистра фильтра"
sFilterConfig.FilterMaskIdHigh = 0x07F8 << 5; // старшая часть второго "регистра фильтра"
sFilterConfig.FilterMaskIdLow = 0x0000;       // младшая часть второго "регистра фильтра"

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}
```

Если мы хотим помещать кадры прошедшие через фильтр №1 в буфер CAN_RX_FIFO1, тогда это нужно указать. *И вообще, если работаете с разными буферами, то не лишним будет указывать это везде чтоб не запутаться, хуже не будет.*

```
sFilterConfig.FilterBank = 1;
sFilterConfig.FilterIdHigh = 0x0200 << 5;     // старшая часть первого "регистра фильтра"
sFilterConfig.FilterIdLow = 0x0000;           // младшая часть первого "регистра фильтра"
sFilterConfig.FilterMaskIdHigh = 0x07F8 << 5; // старшая часть второго "регистра фильтра"
sFilterConfig.FilterMaskIdLow = 0x0000;       // младшая часть второго "регистра фильтра"

sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO1;

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}
```

*Таким образом кадры с **0x0100** по **0x0107** будут падать в буфер CAN_RX_FIFO0, а кадры с **0x0200** по **0x0207** в буфер CAN_RX_FIFO1.*

Далее если есть необходимость вы можете настроить ещё сколько нужно фильтров — максимальный №13.

Режим фильтра *(CAN_FILTERMODE_IDMASK)* и размерность *(CAN_FILTERSCALE_32BIT)* тоже можно настраивать для каждого фильтра отдельно, но об этом позже.

Теперь рассмотрим фильтрование 29-ти битных идентификаторов. *Вы же помните, что мы всё ещё разбираем первый вариант*

использования «регистров фильтра», который подразумевает фильтрование 11-ти и 29-ти битных идентификаторов 😉

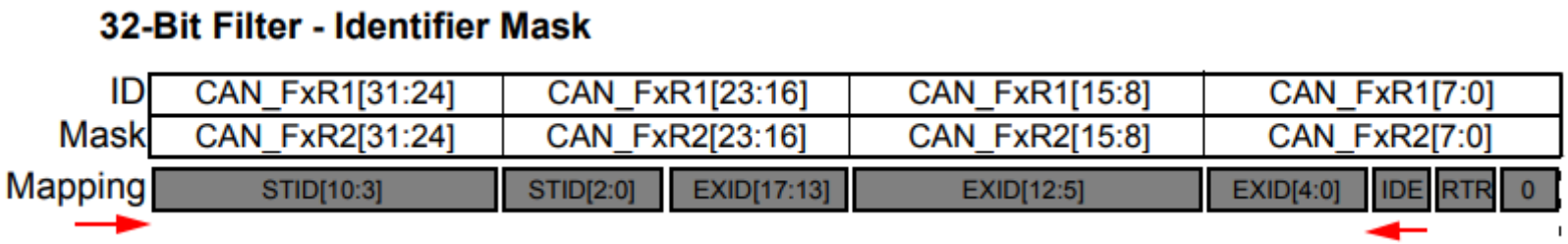
Поскольку настройки режима и размерности фильтра здесь те же самые, что и в предыдущих примерах, мы настроим только номер фильтра и «регистры фильтра», ну и укажем буфер CAN_RX_FIFO1 *(можно CAN_RX_FIFO0, как хотите). Разумеется, если вы настраиваете только один фильтр, тогда нужно прописать все значения. При этом совсем не важно какой номер фильтра вы укажите, можно любой с 0 до 13.*

Настроим фильтр №2, а принимать будем кадры с идентификаторами в диапазоне от **0x00010000** до **0x00010007**…

```
sFilterConfig.FilterBank = 2;
sFilterConfig.FilterIdHigh = (uint16_t)(0x00010000 >> 13);          // старшая часть первого "регистра фильтра"
sFilterConfig.FilterIdLow = (uint16_t)(0x00010000 << 3) | 0x04;     // младшая часть первого "регистра фильтра"
sFilterConfig.FilterMaskIdHigh = (uint16_t)(0x1FFFFFF8 >> 13);      // старшая часть второго "регистра фильтра"
sFilterConfig.FilterMaskIdLow = (uint16_t)(0x1FFFFFF8 << 3) | 0x04; // младшая часть второго "регистра фильтра"

sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO1;

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}
```

Здесь у нас «регистры фильтра» используются полностью, тоже делается сдвиг, чтоб биты встали в нужные места, и ещё в младшие части добавляется | 0x04.

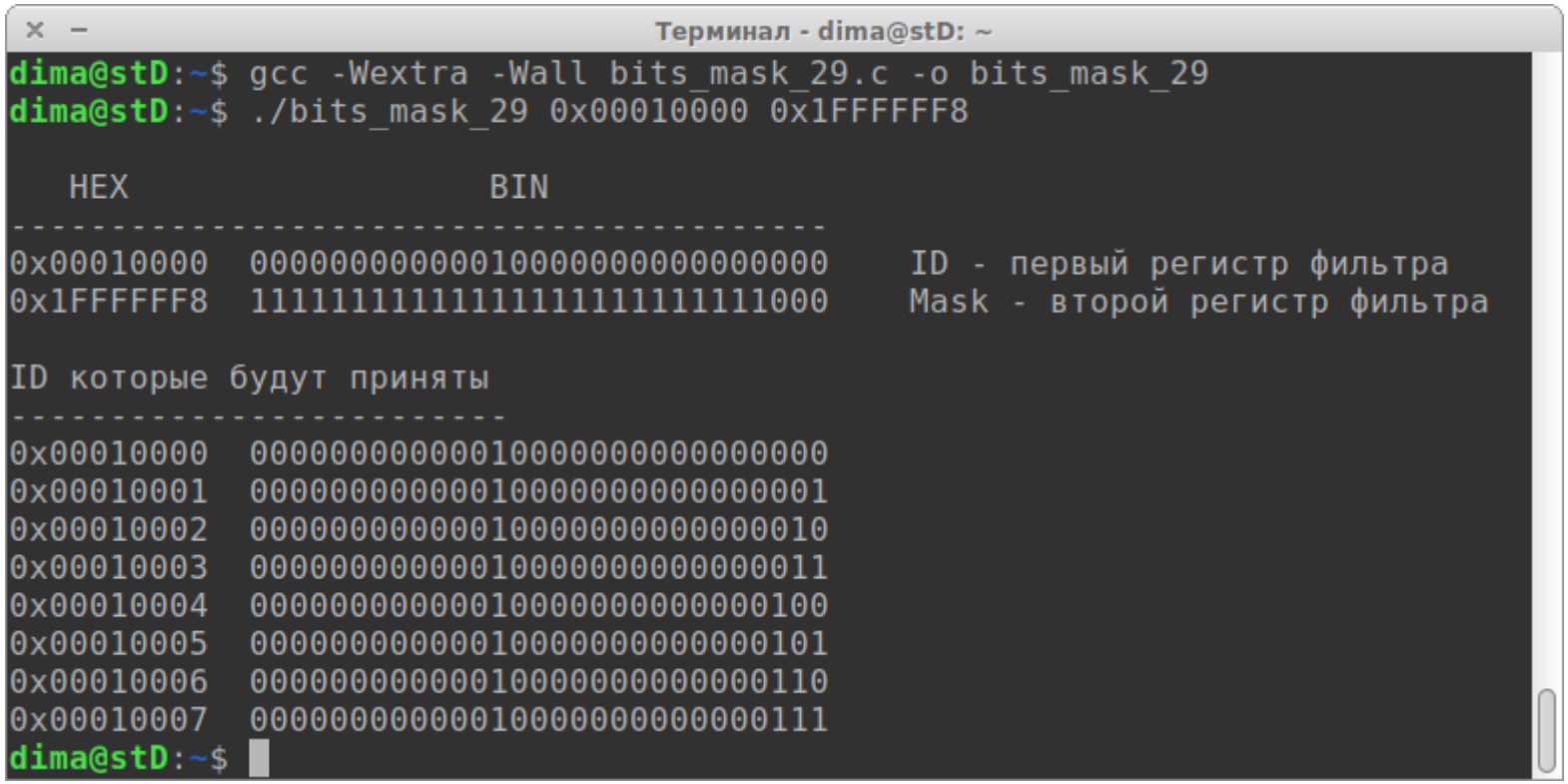
ID и Mask лягут в области STID и EXID…



… а в бит **IDE** будет записана единичка с помощью | 0x04 *(00000100)*. Это сообщит системе, что фильтр настроен на работу с расширенным идентификатором. *То есть, всегда, когда настраиваем фильтр на работу с расширенным кадром, добавляем | 0x04.*

Бит **RTR** не трогаем, однако если захотим настроить фильтр на работу с Remote Frame, тогда в него надо записать единицу. Это касается и стандартных, и расширенных кадров.

Операции с ID и маской здесь точно такие же как и в предыдущем примере, и вот такая же программка, только для 29-ти битных идентификаторов…

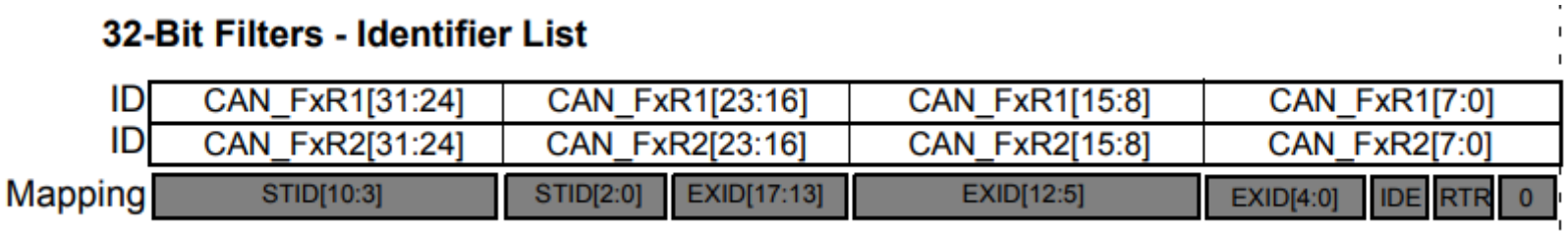


Дальше вы уже знаете что с этим делать.

Вы может настроить несколько фильтров для расширенных кадров, и совмещать их со стандартными, как мы это только что сделали. Сейчас наш CAN принимает стандартные идентификаторы с 0x0100 по 0x0107 и с 0x0200 по 0x0207, складывая их в CAN_RX_FIFO0, и расширенные с 0x00010000 по 0x00010007, складывая их в CAN_RX_FIFO1.

На этом первый вариант закончен, самое трудное позади, дальше будет легко.

Второй вариант — 32-Bit Filters — Identifier List



Здесь всё просто, фильтр можно настроить на приём только одного или двух конкретных идентификатора, они просто записываются в первый и второй «регистры фильтра», так же со сдвигом. Единственное важное отличие от предыдущего варианта, это режим фильтра, тут он будет CAN_FILTERMODE_IDLIST.

Фильтр №3 будет принимать два стандартных идентификатора — 0x06D9 и 0x04C6…

```
sFilterConfig.FilterBank = 3;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDLIST;
sFilterConfig.FilterIdHigh = 0x06D9 << 5;
sFilterConfig.FilterIdLow = 0x0000;
sFilterConfig.FilterMaskIdHigh = 0x04C6 << 5;
sFilterConfig.FilterMaskIdLow = 0x0000;
sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
  Error_Handler();
}
```

А фильтр №4 будет принимать два расширенных идентификатора — 0x000006D9 и 0x000004C6…

```
sFilterConfig.FilterBank = 4;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDLIST;
sFilterConfig.FilterIdHigh = (uint16_t)(0x000006D9 >> 13);
sFilterConfig.FilterIdLow = (uint16_t)(0x000006D9 << 3) | 0x04;
sFilterConfig.FilterMaskIdHigh = (uint16_t)(0x000004C6 >> 13);
sFilterConfig.FilterMaskIdLow = (uint16_t)(0x000004C6 << 3) | 0x04;
sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}
```

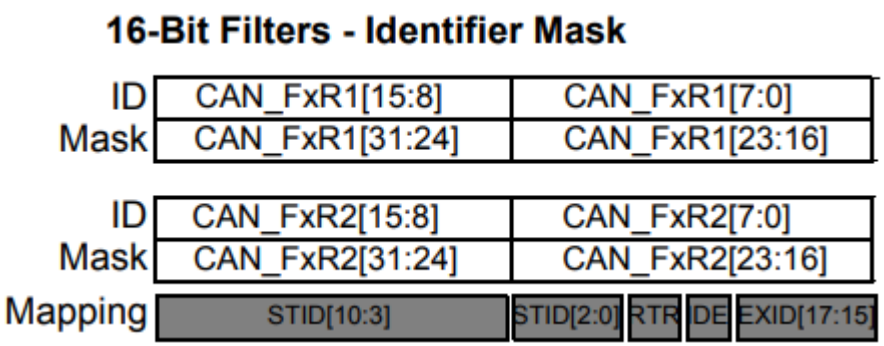
Если нужно принимать только один идентификатор, тогда во второй «регистр фильтра» нужно прописать то же, что и в первом…

```
sFilterConfig.FilterIdHigh = (uint16_t)(0x000006D9 >> 13);
sFilterConfig.FilterIdLow = (uint16_t)(0x000006D9 << 3) | 0x04;
sFilterConfig.FilterMaskIdHigh = (uint16_t)(0x000006D9 >> 13);
sFilterConfig.FilterMaskIdLow = (uint16_t)(0x000006D9 << 3) | 0x04;
```

Получится что CAN будет два раза фильтровать одно и то же, но другого варианта видимо нет, так как если записать туда просто нули, будет приниматься идентификатор 0x0000.

Со вторым вариантом всё.

Третий вариант — 16-Bit Filters — Identifier Mask



Этот вариант, как и первый, работает с маской, но может фильтровать только стандартные *(11 бит)* кадры. Достоинство этого варианта перед первым в том, что один фильтр можно настроить на два диапазона идентификаторов. Это достигается за счёт использования «регистров фильтра» полностью, а не только старшие части.

Будем принимать кадры в диапазонах с **0x0320** по **0x0323**, и с **0x0480** по **0x0487**.

Настроим фильтр №5…

```
sFilterConfig.FilterBank = 5;
```

Режим фильтра на работу с маской…

```
sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
```

Важное отличие от предыдущих примеров — размерность фильтра указываем 16 бит…

```
sFilterConfig.FilterScale = CAN_FILTERSCALE_16BIT;
```

В старшую и младшую часть первого «регистра фильтра» записываем начальные ID первого и второго диапазона…

```
sFilterConfig.FilterIdHigh = 0x0320 << 5;     // ID 1
sFilterConfig.FilterIdLow = 0x0480 << 5;      // ID 2
```

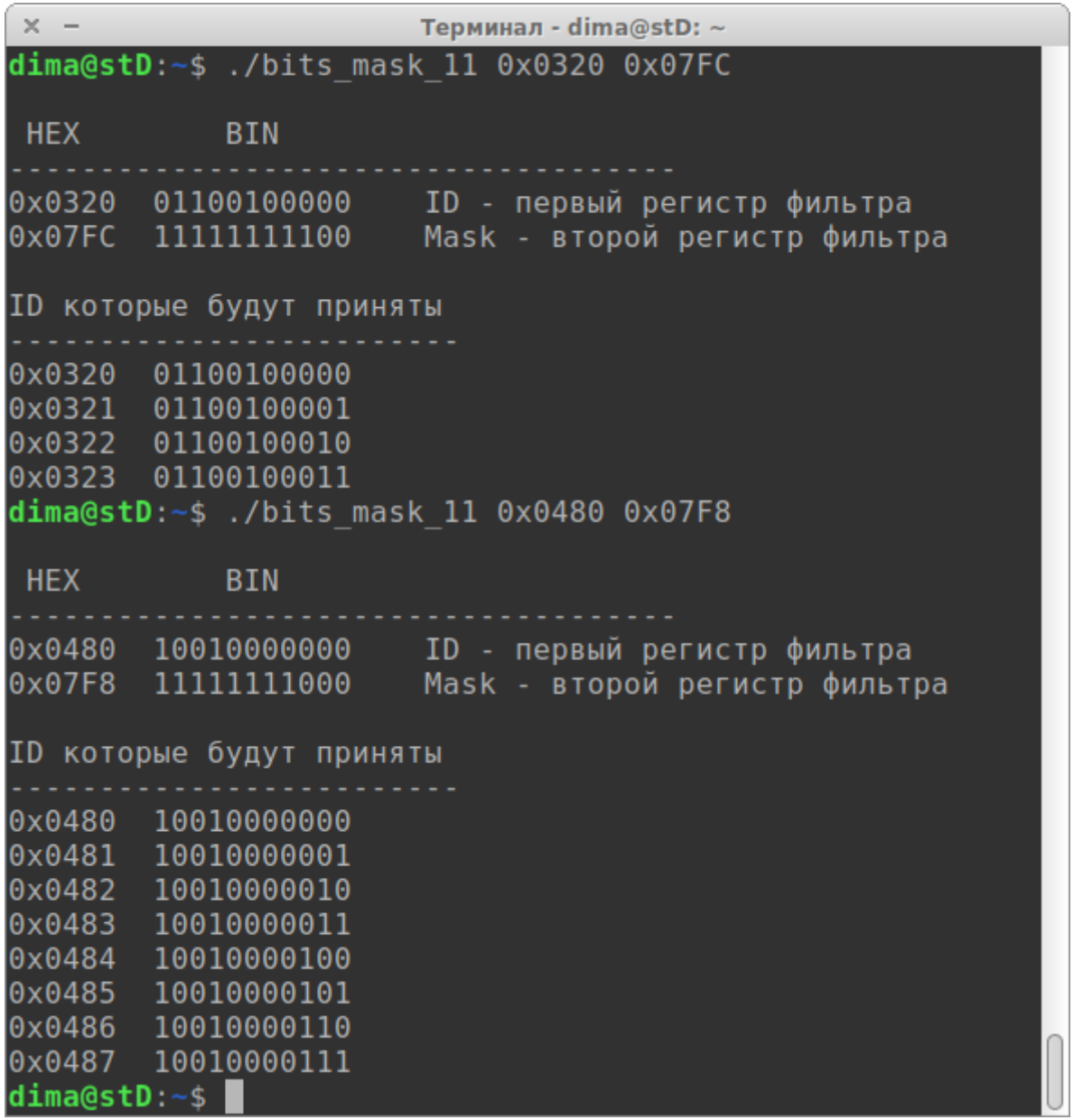
А в старшую и младшую часть второго «регистра фильтра» записываем соответствующие маски…

```
sFilterConfig.FilterMaskIdHigh = 0x07FC << 5; // Mask 1
sFilterConfig.FilterMaskIdLow = 0x07F8 << 5;  // Mask 2
```

Целиком будет так…

```
sFilterConfig.FilterBank = 5;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
sFilterConfig.FilterScale = CAN_FILTERSCALE_16BIT;
sFilterConfig.FilterIdHigh = 0x0320 << 5;     // ID 1
sFilterConfig.FilterIdLow = 0x0480 << 5;      // ID 2
sFilterConfig.FilterMaskIdHigh = 0x07FC << 5; // Mask 1
sFilterConfig.FilterMaskIdLow = 0x07F8 << 5;  // Mask 2

sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}
```

Проверяем в программе…



Если нужно принимать только один диапазон, тогда вместо второго идентификатора и маски нужно прописать то же что и для первого…

```
sFilterConfig.FilterIdHigh = 0x0320 << 5;     // ID 1
sFilterConfig.FilterIdLow = 0x0320 << 5;      // ID 2
sFilterConfig.FilterMaskIdHigh = 0x07FC << 5; // Mask 1
sFilterConfig.FilterMaskIdLow = 0x07FC << 5;  // Mask 2
```

Опять же, получится что CAN будет выполнять лишнюю работу, но если записать туда нули, тогда будут приниматься вообще все стандартные кадры.

Надеюсь тут всё понятно.

Четвёртый вариант — 16-Bit Filters — Identifier List

Этот вариант похож на второй, только опять же, мы можем фильтровать не два конкретных идентификатора, а четыре. И как вы уже наверно догадались, они могут быть только стандартными.

Будем принимать идентификаторы **0x0690**, **0x0693**, **0x0696** и **0x0699**.

Настраиваем фильтр №6, и меняем режим на CAN_FILTERMODE_IDLIST…

```
sFilterConfig.FilterBank = 6;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDLIST;
sFilterConfig.FilterScale = CAN_FILTERSCALE_16BIT;
sFilterConfig.FilterIdHigh = 0x0690 << 5;     // ID 1
sFilterConfig.FilterIdLow = 0x0693 << 5;      // ID 2
sFilterConfig.FilterMaskIdHigh = 0x0696 << 5; // ID 3
sFilterConfig.FilterMaskIdLow = 0x0699 << 5;  // ID 4

sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}
```

Опять же, если вместо какого-то идентификатора вписать 0x0000, то будет приниматься кадр 0x0000.

С последним вариантом, равно как и с разбором фильтров, покончено.

В довершение осталось сказать что фильтр настроенный как CAN_FILTERSCALE_32BIT имеет более высокий приоритет чем CAN_FILTERSCALE_16BIT.

То есть если мы настроим два фильтра вот так…

```
sFilterConfig.FilterBank = 0;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
sFilterConfig.FilterScale = CAN_FILTERSCALE_16BIT; // отличие
sFilterConfig.FilterIdHigh = 0x0000;
sFilterConfig.FilterIdLow = 0x0000;
sFilterConfig.FilterMaskIdHigh = 0x0000;
sFilterConfig.FilterMaskIdLow = 0x0000;
sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0; // буфер 0
sFilterConfig.FilterActivation = ENABLE;

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}

sFilterConfig.FilterBank = 1;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT; // отличие
sFilterConfig.FilterIdHigh = 0x0000;
sFilterConfig.FilterIdLow = 0x0000;
sFilterConfig.FilterMaskIdHigh = 0x0000;
sFilterConfig.FilterMaskIdLow = 0x0000;
sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO1; // буфер 1

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}
```

… все кадры будут складываться в буфер 1.

Так же и номера фильтров имеют приоритет. Чем меньше номер фильтра, тем выше у него приоритет. В примере ниже, все кадры будут складываться опять же в буфер 1.

```
sFilterConfig.FilterBank = 1;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
sFilterConfig.FilterIdHigh = 0x0000;
sFilterConfig.FilterIdLow = 0x0000;
sFilterConfig.FilterMaskIdHigh = 0x0000;
sFilterConfig.FilterMaskIdLow = 0x0000;
sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0; // буфер 0
sFilterConfig.FilterActivation = ENABLE;

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}

sFilterConfig.FilterBank = 0;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
sFilterConfig.FilterIdHigh = 0x0000;
sFilterConfig.FilterIdLow = 0x0000;
sFilterConfig.FilterMaskIdHigh = 0x0000;
sFilterConfig.FilterMaskIdLow = 0x0000;
sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO1; // буфер 1

if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
{
   Error_Handler();
}
```

А если поменять номера фильтров, то в буфер 0. Всё.

Два CAN'а на одном камне

Единственной особенностью CAN2 является то, что он не может работать при отключённом CAN1, во всём остальном настройки CAN2 такие же как у CAN1, кроме номеров фильтров. Чтоб CAN2 заработал, достаточно просто активировать CAN1 и настроить у него хотя бы один фильтр.

То есть инициализация обоих CAN'ов будет такая…

```
////////////// CAN 1 ///////////////
static void MX_CAN1_Init(void)
{
  CAN_FilterTypeDef  sFilterConfig;

  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 4;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan1.Init.TimeSeg1 = CAN_BS1_15TQ;
  hcan1.Init.TimeSeg2 = CAN_BS2_2TQ;
  hcan1.Init.TimeTriggeredMode = DISABLE;
  hcan1.Init.AutoBusOff = ENABLE;
  hcan1.Init.AutoWakeUp = DISABLE;
  hcan1.Init.AutoRetransmission = DISABLE;
  hcan1.Init.ReceiveFifoLocked = DISABLE;
  hcan1.Init.TransmitFifoPriority = ENABLE;

  if(HAL_CAN_Init(&hcan1) != HAL_OK)
  {
    Error_Handler();
  }

  /* USER CODE BEGIN CAN1_Init 2 */

  ////////////////// Filter CAN 1 ////////////////////
  sFilterConfig.FilterBank = 0;
  sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
  sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
  sFilterConfig.FilterIdHigh = 0x0000;
  sFilterConfig.FilterIdLow = 0x0000;
  sFilterConfig.FilterMaskIdHigh = 0x0000;
  sFilterConfig.FilterMaskIdLow = 0x0000;
  sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;
  sFilterConfig.FilterActivation = ENABLE;

  if(HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig) != HAL_OK)
  {
    Error_Handler();
  }
}
```

```
/////////////// CAN 2 ///////////////
static void MX_CAN2_Init(void)
{
  CAN_FilterTypeDef  sFilterConfig;

  hcan2.Instance = CAN2;
  hcan2.Init.Prescaler = 4;
  hcan2.Init.Mode = CAN_MODE_NORMAL;
  hcan2.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan2.Init.TimeSeg1 = CAN_BS1_15TQ;
  hcan2.Init.TimeSeg2 = CAN_BS2_2TQ;
  hcan2.Init.TimeTriggeredMode = DISABLE;
  hcan2.Init.AutoBusOff = ENABLE;
  hcan2.Init.AutoWakeUp = DISABLE;
  hcan2.Init.AutoRetransmission = DISABLE;
  hcan2.Init.ReceiveFifoLocked = DISABLE;
  hcan2.Init.TransmitFifoPriority = ENABLE;

  if(HAL_CAN_Init(&hcan2) != HAL_OK)
  {
    Error_Handler();
  }

  /* USER CODE BEGIN CAN2_Init 2 */

  ////////////////// Filter CAN 2 ///////////////////////
  sFilterConfig.FilterBank = 14;
  sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
  sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
  sFilterConfig.FilterIdHigh = 0x0000;
  sFilterConfig.FilterIdLow = 0x0000;
  sFilterConfig.FilterMaskIdHigh = 0x0000;
  sFilterConfig.FilterMaskIdLow = 0x0000;
  sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;
  sFilterConfig.FilterActivation = ENABLE;
  sFilterConfig.SlaveStartFilterBank = 14;

  if(HAL_CAN_ConfigFilter(&hcan2, &sFilterConfig) != HAL_OK)
  {
    Error_Handler();
```

```
  }
}
```

Для CAN1 фильтры с 0 по 13, для CAN2 фильтры с 14 по 27. У CAN2 есть дополнительный параметр — sFilterConfig.SlaveStartFilterBank = 14, сообщающий системе с какого номера начинаются фильтры CAN2. Добавляются фильтры так же как описано выше.

Разумеется CAN'ы можно настраивать на разные скорости. Собственно не знаю что ещё сказать.

Не смотря на то, что в мануалах CAN2 называют Slave, это полностью самостоятельный интерфейс, просто он не может работать без включённого CAN1, так как именно CAN1 включает всю систему буферов, фильтров, почтовых ящиков, и т.д. Видимо это какая-то особенность проектирования камня. Наверно сначала не планировали два CAN'а делать, а потом передумали и приладили 🙂

Если CAN1 не используется и к нему не подключён трансивер, тогда ножку RX нужно подтянуть к плюсу, можно внутреннюю подтяжку пользовать.

Запускаются оба CAN'а так…

```
HAL_CAN_Start(&hcan1);
HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING | CAN_IT_ERROR | CAN_IT_BUSOFF | CAN_IT_LAST_ERROR_CODE);

HAL_CAN_Start(&hcan2);
HAL_CAN_ActivateNotification(&hcan2, CAN_IT_RX_FIFO0_MSG_PENDING | CAN_IT_ERROR | CAN_IT_BUSOFF | CAN_IT_LAST_ERROR_CODE);
```

Если CAN1 не используется, тогда его запускать не нужно.

CAN_IT_RX_FIFO0_MSG_PENDING — это может быть CAN_IT_RX_FIFO**1**_MSG_PENDING, зависит от того какой буфер вы указали в настройках фильтра *(CAN_RX_FIFO0 или CAN_RX_FIFO1)*.

Колбек для ошибок…

```
void HAL_CAN_ErrorCallback(CAN_HandleTypeDef *hcan)
{
	if(hcan->Instance == CAN1)
	{
		uint32_t er = HAL_CAN_GetError(&hcan1);
		sprintf(trans_str,"ER CAN_1 %lu %08lX", er, er);
		trans_to_usart1(trans_str);
	}
	else if(hcan->Instance == CAN2)
	{
		uint32_t er = HAL_CAN_GetError(&hcan2);
		sprintf(trans_str,"ER CAN_2 %lu %08lX", er, er);
		trans_to_usart1(trans_str);
	}
}
```

Колбек для приёма, если у обоих CAN'ов задействован буфер CAN_RX_FIFO0…

```
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
	//////////////////////////// CAN1 ////////////////////////////////
	if(HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &RxHeader_1, RxData_1) == HAL_OK)
	{

	}

	//////////////////////////// CAN2 ////////////////////////////////
	if(HAL_CAN_GetRxMessage(&hcan2, CAN_RX_FIFO0, &RxHeader_2, RxData_2) == HAL_OK)
	{

	}
}
```

Если для обоих CAN'ов задействован буфер CAN_RX_FIFO1…

```
void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
	//////////////////////////// CAN1 ////////////////////////////////
	if(HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO1, &RxHeader_1, RxData_1) == HAL_OK)
	{

	}

	//////////////////////////// CAN2 ////////////////////////////////
	if(HAL_CAN_GetRxMessage(&hcan2, CAN_RX_FIFO1, &RxHeader_2, RxData_2) == HAL_OK)
	{

	}
}
```

*Обратите внимание, что в названии функции ...RxFifo**0**… поменялось на ...RxFifo**1**...*

Если для разных CAN'ов настроены разные буферы…

```
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
	//////////////////////////// CAN1 ////////////////////////////////
	if(HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &RxHeader_1, RxData_1) == HAL_OK)
	{

	}
}
```

```
void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
	//////////////////////////// CAN2 ////////////////////////////////
	if(HAL_CAN_GetRxMessage(&hcan2, CAN_RX_FIFO1, &RxHeader_2, RxData_2) == HAL_OK)
	{

	}
}
```

Если CAN1 не используется, тогда для него ничего писать не нужно.

Ну и отправка…

```
if(HAL_CAN_GetTxMailboxesFreeLevel(&hcan1) > 0)
{
	HAL_CAN_AddTxMessage(&hcan1, &TxHeader_1, TxData_1, &TxMailbox_1);
}


if(HAL_CAN_GetTxMailboxesFreeLevel(&hcan2) > 0)
{
	HAL_CAN_AddTxMessage(&hcan2, &TxHeader_2, TxData_2, &TxMailbox_2);
}
```

Статью писал долго, с перерывами, так что если что упустил/забыл — пишите в чат.

Это всё, всем спасибо

Телеграм-чат istarik

Телеграм-чат STM32

CAN, Filter, Фильтры, два CAN, CAN2

0 15 октября 2021, 19:26 stD 5635

Поддержать автора

- ○ Яндекс.Деньгами
- ○ Банковской картой
- ○ С баланса телефона

сумма ₽ | Перевести

Telegram-чат istarik

Задать вопрос по статье