Max Schulz

A random corner in the internet

STM32 CAN Identifier Filtering

23 Mar 2017

Introduction

This is a rather specific post that is motivated by the rather unintuitive use of Identifier filtering on an STM32Fxx CAN Bus and by the impossibility to find some easy examples. For a general overview, refer to the official manual from page 1092 onwards.

Identifier filtering

Every CAN message has an associated ID sent with its content. A receiver can decide on a hardware level which IDs to accept by using 27 so called filter banks which can implement either one 32-bit filter or two 16-bit filter. The usage of

these also depends on your scale of CAN Msg IDs which can either be standard IDs (11-bit) or extended IDs (29-bit). In this post we assume the usage of the extended IDs.

Mask Mode

The filter makes use of two 29-bit numbers - composed in a `FilterId` and a `FilterMask` and there are two modes to be used: Identifier Mode and Mask Mode. The identifier mode is pretty simple: both numbers represent IDs that should be caught by the filter. For the Mask Mode that we are presenting here, the `FilterMask` specifies which bits to compare with the `FilterId`, some examples for extended IDs:

- `MASK = 0x1FFFFFFE` and `ID = 0x00000002` will catch IDs 2 and 3.
- `MASK = 0x1FFFFFF8` and `ID = 0x00000000` will catch IDs 0 to 7.
- `MASK = 0x1FFFFFFF` and `ID = 0x00000008` will catch ID 8.
- `MASK = 0x00000001` and `ID = 0x00000001` will catch all odd IDs.

So for “normal” operations, you can only filter either single IDs or ranges in the form `2^N - (2^(N-1) - 1)` but there are also possibilities such as only accepting odd IDs and whatever you can combine with these representations. Each filter is also associated with a number, with generally being the lower the number the higher the priority, other priority rules are given in the documentation. One important point that is not mentioned though: In mask mode, a mask of `0x00` means that all IDs gets passed, be aware that this filter will “grab” all IDs from all other filters and the filter number doesn’t play a role anymore!

Implementation

The actual filters are actually used by configuring the `CAN_FilterInitTypeDef` struct of the `stm32f4xx_can.h` filter, which has various fields. For the specific case of this post, the filter is setup as seen in this snippet, with `filter_id` and `filter_mask` being an input from the user as specified before:

```
    CAN_FilterInitTypeDef filter;

    filter.CAN_FilterIdHigh = ((filter_id << 5)  | (filter_id >> (32 -
5))) & 0xFFFF; // STID[10:0] & EXTID[17:13]
    filter.CAN_FilterIdLow = (filter_id >> (11 - 3)) & 0xFFF8; // EXID
[12:5] & 3 Reserved bits
    filter.CAN_FilterMaskIdHigh = ((filter_mask << 5)  | (filter_mask
>> (32 - 5))) & 0xFFFF;
    filter.CAN_FilterMaskIdLow = (filter_mask >> (11 - 3)) & 0xFFF8;

    filter.CAN_FilterFIFOAssignment = CAN_Filter_FIFO0;
    filter.CAN_FilterNumber = filter_num;
    filter.CAN_FilterMode = CAN_FilterMode_IdMask;
    filter.CAN_FilterScale = CAN_FilterScale_32bit;
```

```
filter.CAN_FilterActivation = ENABLE;
CAN_FilterInit(&filter);
```

The reasoning behind this bit shifting is best understood by having a look at the referenced manual on `p. 1092`. Whereas `High` and `Low` are the higher and lower indices of the filter registers respectively.

Related Posts

Python Threading with External Processes 17 Nov 2016

Short Introduction into Catkin Packages 22 Aug 2016

Subscribe and Access Topic in Rospy regardless of Message Type 18 Jul 2016

Matthew Turner • 7 months ago • edited

I found the above didn't handle 29 bit addresses correctly. It seemed to be shifting the LSB of the extended identifier into the standard identifier. These changes work for me confirmed with Microchip CANBUS analyzer.

```
filter.FilterIdHigh = filter_id >> 13 & 0xFFFF;
filter.FilterIdLow = filter_id << 3 & 0xFFF8;
filter.FilterMaskIdHigh = filter_mask >> 13 & 0xFFFF;
filter.FilterMaskIdLow = filter_mask << 3 & 0xFFF8;
```


sina rahbari • 9 months ago

Thanks man, I got it finally

Jordan • 2 years ago

Very helpful, thanks!