排查嵌入式軟件中的bug時，別把代碼改得爛七八糟~

嵌入式資訊精選 2021-12-13 11:34

本文對嵌入式開發中問題排查的思路、方法和經驗總結得不錯，大家可以參考學習。

一、問題復現

穩定復現問題才能正確的對問題進行定位、解決以及驗證。一般來說，越容易復現的問題越容易解決。

1.1 模擬復現條件

有的問題存在於特定的條件下，只需要模擬出現問題的條件即可複現。對於依賴外部輸入的條件，如果條件比較複雜難以模擬可以考慮程序裡預設直接進入對應狀態。

1.2 提高相關任務執行頻率

例如某個任務長時間運行才出現異常則可以提高該任務的執行頻率。

1.3 增大測試樣本量

程序長時間運行後出現異常，問題難以復現，可以搭建測試環境多套設備同時進行測試。

二、問題定位

縮小排查範圍，確認引入問題的任務、函數、語句。

2.1 打印LOG

根據問題的現象，在抱有疑問的代碼處增加LOG輸出，以此來追踪程序執行流程以及關鍵變量的值，觀察是否與預期相符。

2.2 在線調試

在線調試可以起到和打印LOG類似的作用，另外此方法特別適合排查程序崩潰類的BUG，當程序陷入異常中斷(HardFault，看門狗中斷等)的時候可以直接STOP查看call stack以及內核寄存器的值，快速定位問題點。

2.3 版本回退

使用版本管理工具時可以通過不斷回退版本並測試驗證來定位首次引入該問題的版本，之後可以圍繞該版本增改的代碼進行排查。

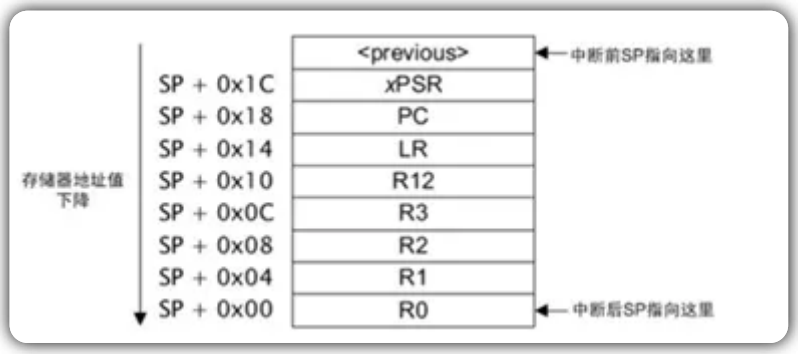
2.4 二分註釋

二分注释即以類似二分查找法的方式註釋掉部分代碼，以此判斷問題是否由註釋掉的這部分代碼引起。

具體方法為將與問題不相干的部分代碼註釋掉一半，看問題是否解決，未解決則註釋另一半，如果解決則繼續將註釋範圍縮小一半，以此類推逐漸縮小問題的範圍。

2.5 保存內核寄存器快照

Cortex M內核陷入異常中斷時會將幾個內核寄存器的值壓入棧中，如下圖：



我們可以在陷入異常中斷時將棧上的內核寄存器值寫入RAM的一段復位後保留默認值的區域內，執行複位操作後再從RAM將該信息讀出並分析，通過PC、LR確認當時執行的函數，通過R0-R3分析當時處理的變量是否異常，通過SP分析是否可能出現棧溢出等。

三、問題分析處理

結合問題現像以及定位的問題代碼位置分析造成問題的原因。

3.1 程序繼續運行

3.1.1 數值異常

3.1.1.1 軟件問題

1、數組越界

寫數組時下標超出數組長度，導致對應地址內容被修改。如下：

```
#define ARRAY_LEN 10
int32_t arr[ARRAY_LEN] = {0};

//不安全的代码
void setArrVal(uint32_t index, int32_t val)
{
    //此处未判断index的值，如果其值超出arr[]的长度则会修改相邻高地址的变量
    arr[index] = val;
    //如果进行下标乘法运算，则超出长度会修改距离较远的高地址变量
    arr[ index * 5 ] = val;
}

//安全的代码
ErrCode setArrVal(const uint32_t index, const int32_t val)
{
    //对下标长度进行判断，超出时返回错误值
    if( index >= ARRAY_LEN ){
        return ERR_CODE_OVERFLOW;
    }

    arr[index] = val;
    return ERR_CODE_NONE;
}
```

此類問題通常需要結合map文件進行分析，通過map文件觀察被篡改變量地址附近的數組，查看對該數組的寫入操作是否存在如上圖所示不安全的代碼，將其修改為安全的代碼。

2、棧溢出

0x20001ff8	**g_val**
0x20002000	棧底
.....	棧空間
0x20002200	棧頂

如上圖，此類問題也需要結合map文件進行分析。假設棧從高地址往低地址增長，如果發生棧溢出，則g_val的值會被棧上的值覆蓋。

出現棧溢出時要分析棧的最大使用情況，函數調用層數過多，中斷服務函數內進行函數調用，函數內部申明了較大的臨時變量等都有可能導致棧溢出。

解決此類問題有以下方法：

- 在設計階段應該合理分配內存資源，為棧設置合適的大小；
- 將函數內較大的臨時變量加”static”關鍵字轉化為靜態變量，或者使用malloc()動態分配，將其放到堆上；
- 改變函數調用方式，降低調用層數。

3、判斷語句條件寫錯

```
//正确判断语句
if( val == 0 )
//错写成赋值语句
if( val = 0 )
//建议的写法
if( 0 == val )
```

判斷語句的條件容易把相等運算符“==”寫成賦值運算符“=”導致被判斷的變量值被更改，該類錯誤編譯期不會報錯且總是返回真。

建議將要判斷的變量寫到運算符的右邊，這樣錯寫為賦值運算符時會在編譯期報錯。還可以使用一些靜態代碼檢查工具來發現此類問題。

4、同步問題

例如操作隊列時，出隊操作執行的過程中發生中斷(任務切換)，並且在中斷(切換後的任務)中執行入隊操作則可能破壞隊列結構，對於這類情況應該操作時關中斷（使用互斥鎖同步）。

5、優化問題

```
uint8_t flg = 0;

void main(void)
{
    flg = 0;
    while(!flg){
        foo();
    }
}

void irq_handler(void)
{
    flg = 1;
}
```

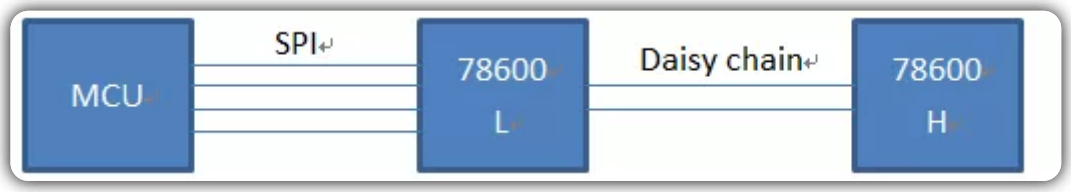
如上圖程序，本意是等待irq中斷之後不再執行foo()函數，但被編譯器優化之後，實際運行過程中flg可能被裝入寄存器並且每次都判斷寄存器內的值而不重新從ram裡讀取flg的值，導致即使irq中斷發生foo()也一直運行，此處需要在flg的申明前加“volatile”關鍵字，強制每次都從ram裡獲取flg的值。

3.1.1.2 硬件問題

1、芯片BUG

芯片本身存在BUG，在某些特定情況下給單片機返回一個錯誤的值，需要程序對讀回的值進行判斷，過濾異常值。

2、通信時序錯誤



例如電源管理芯片Isl78600，假設現在兩片級聯，當同時讀取兩片的電壓採樣數據時，高端芯片會以固定週期通過菊花鏈將數據傳送到低端芯片，而低端芯片上只有一個緩存區.

如果單片機不在規定時間內將低端芯片上的數據讀走那麼新的數據到來時將會覆蓋當前數據，導致數據丟失。此類問題需要仔細分析芯片的數據手冊，嚴格滿足芯片通信的時序要求。

3.1.2 動作異常

3.1.2.1 軟件問題

1、設計問題

設計中存在錯誤或者疏漏，需要重新評審設計文檔。

2、實現與設計不符

代碼的實現與設計文檔不相符需要增加單元測試覆蓋所有條件分支，進行代碼交叉review。

3、狀態變量異常

例如記錄狀態機當前狀態的變量被篡改，分析該類問題的方法同前文數值異常部分。

3.1.2.2 硬件問題

1、硬件失效

目標IC失效，接收控制指令後不動作，需要排查硬件。

2、通信異常

與目標IC通信錯誤，無法正確執行控制命令，需要使用示波器或邏輯分析儀去觀察通信時序，分析是否發出的信號不對或者受到外部干擾。

3.2 程序崩潰

3.2.1 停止運行

3.2.1.1 軟件問題

1、HardFault

以下情況會造成HardFault：

- 在外設時鐘門未使能的情況下操作該外設的寄存器；
- 跳轉函數地址越界，通常發生在函數指針被篡改，排查方法同數值異常；
- 解引用指針時出現對齊問題：

以小端序為例，如果我們聲明了一個強制對齊的結構體如下：

```
typedef __packed struct{
    uint8_t val0;
    uint16_t val1;
    uint8_t val2;
}tstruct;

tstruct a;
a.val0 = 0x12;
a.val1 = 0x3456;
a.val2 = 0x78;
```

地址	0x00000000	0x00000001	0x00000002	0x00000003
變量名	Val0	Val1_low	Val1_high	Val2
值	0x12	0x56	0x34	0x78

此時a.val1的地址為0x00000001，如果以uint16_t類型去解引用此地址則會因為對齊問題進入HardFault，如果一定要用指針方式操作該變量則應當使用memcpy()。

2、中斷服務函數中未清除中斷標誌

中斷服務函數退出前不正確清除中斷標誌，當程序執行從中斷服務函數內退出後又會立刻進入中斷服務函數，表現出程序的“假死”現象。

3、NMI中斷

調試時曾遇到SPI的MISO引腳復用NMI功能，當通過SPI連接的外設損壞時MISO被拉高，導致單片機復位後在把NMI引腳配置成SPI功能之前就直接進入NMI中斷，程序掛死在NMI中斷中。這種情況可以在NMI的中斷服務函數內禁用NMI功能來使其退出NMI中斷。

3.2.1.2 硬件問題

1、晶振未起振

2、供電電壓不足

3、復位引腳拉低

3.2 .2 復位

3.2.2.1 軟件問題

1、看門狗復位

除了餵狗超時導致的複位以外，還要注意看門狗配置的特殊要求，以Freescale KEA單片機為例，該單片機看門狗在配置時需要執行解鎖序列（向其寄存器連續寫入兩個不同的值），該解鎖序列必須在16個總線時鐘內完成，超時則會引起看門狗復位。此類問題只能熟讀單片機數據手冊，注意類似的細節問題。

3.2.2.2 硬件問題

1、供電電壓不穩

2、電源帶載能力不足

四、回歸測試

問題解決後需要進行回歸測試，一方面確認問題是否不再复現，另一方面要確認修改不會引入其他問題。

五、經驗總結

總結本次問題產生的原因及解決問題的方法，思考類似問題今後如何防範，對相同平台產品是否值得借鑒，做到舉一反三，從失敗中吸取經驗。

文章來源:
https://www.cnblogs.com/jozochen/p/8541714.html

END

1.嵌友們，BUG不好找？那是因為宏定義沒用好！
2.GD32V RISC-V MCU調試體驗
3.2022年六大值得關注的邊緣計算趨勢
4.2021年第12期《單片機與嵌入式系統應用》電子刊新鮮出爐！
5.STM32的SPI外設片選只有一個，怎麼破？
6.C語言開發單片機，為什麼都是全局變量形式？

长按指纹“识别二维码”关注

喜歡此內容的人還喜歡

附代碼|Java日誌庫Log4j2注入漏洞復現，看看它危害有多大
碼農小胖哥

吐槽物聯網開發難搞，一群工程師搞了個Toit 語言並宣布開源
嵌入式資訊精選

和一個嵌入式工程師聊代碼，他建議我回去重學C語言

STM32嵌入式開發