目录

QPC移植

GIT使用

QM的使用

QPC程序移植

移植步骤

1、滴答定时器

2、BSP\_init(void)

3、QF\_onStartup(void)

4、QV\_onIdle(void)

QPC程序设计主要的函数

1、QActive\_Subscribe() 事件订阅

2、QF\_PUBLISH() 发布信号

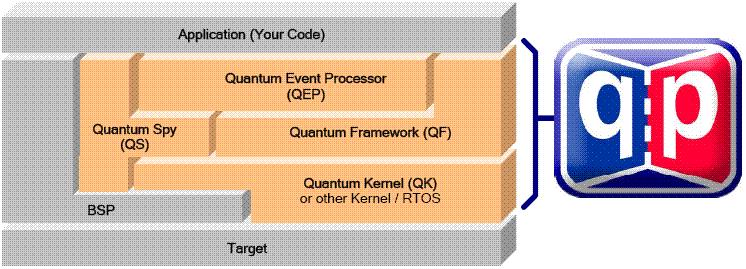
3、QActive\_POST() 发行信号

# **[QP之QF原理](https://www.cnblogs.com/hyper99/p/QP-zhiQF-yuan-li.html)**

## **1.QP简介：**

量子平台(Quantum Platform, 简称QP)是一个用于实时嵌入式系统的软件框架，QP是轻量级的、开源的、基于层次式状态机的、事件驱动的平台。

QP包括事件处理器(QEP)、轻量级的事件驱动框架(QF)、任务调度微内核(QK)和实时跟踪调试器(QS)四个部分。

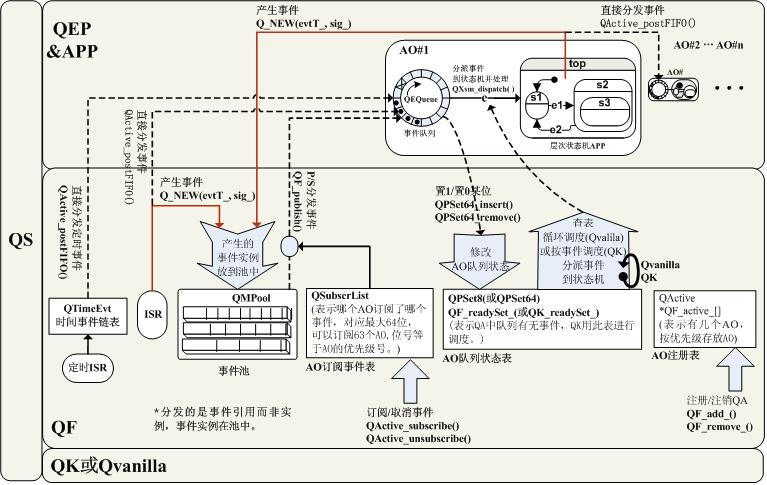
利用QP可以开发出结构清晰的嵌入式应用程序(使用C或C++语言)。  
  
**图1.QP总体结构**

## **2.QP之QF简介**

QF是QP的核心，可以把QF理解为软件总线(software bus，SBus)。在这个软件总线上连接着很多的AO和硬件中断服务程序ISR，QF负责事件的存贮、分发和回收等功能，也就是事件驱动。

QF是一个事件驱动框架。应用程序可以创建事件实例，放到事件池中，并分发事件到相应注册了这个事件的活动对象(状态机,任务)AO中。应用程序可以注册特定的事件，当有事件发生时，QF就会分发事件到AO中。应用程序是由多个AO组成，每个AO可以认为是一个任务。把硬件中断程序也当做AO看待，只是优先级比较高。

QF量子框架由五个数据结构及操作组成，其数据结构采用了uCOS-II相似的结构。

  
**图2.细化的QF总体结构**

## **3.QF分发事件**

如图 1所示，QF是个软件总线，在这个软件总线上连接着AO和ISR。QF中包含5个主要的数据结构（受uCOS II影响很深，要看一下uCOS II内核）。

有两种事件的分发方式:

**（1）直接分发事件**

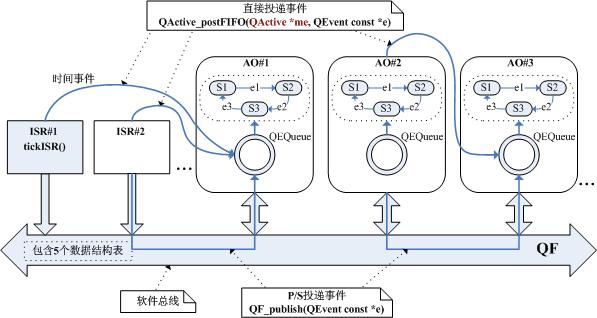
一个AO直接分发事件到另一个AO，就是直接投递事件。这种方式分发事件的AO要知道目标AO。

使用void QActive\_postFIFO(QActive \*me, QEvent const \*e) 函数分发事件。

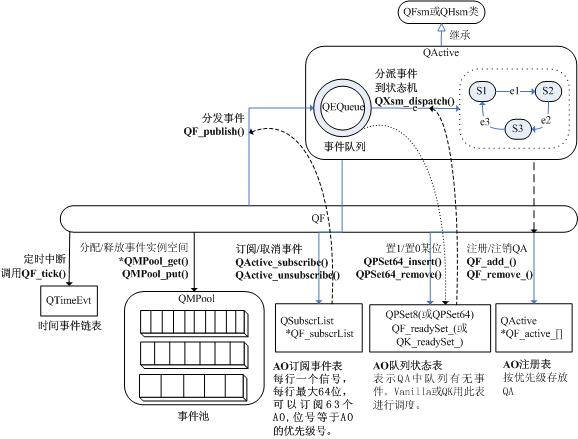
**（2）P/S分发事件**

利用QF中的AO订阅事件表，以P/S(Publish-Subscribe出版-订阅)方式分发事件，也就是P/S投递事件。 这种方式分发事件AO不用知道目标AO，QF查找AO订阅事件表就知道把事件分发到什么地方。P/S投递方式减少了AO之间的耦合度。

使用void QF\_publish(QEvent const \*e)函数分发事件。

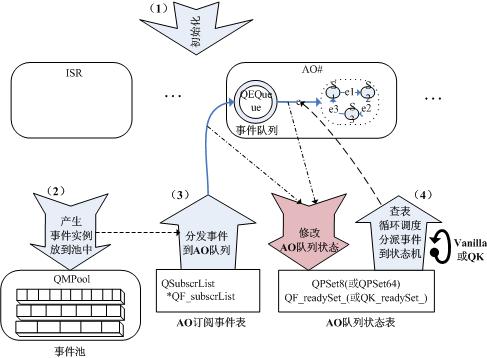
  
**图3.QF软件总线和两种投递事件方式**

框架QF包括活动对象AO，每个AO包含一个AO队列(QEQueue)和状态机。QF中还有5个主要的数据结构表，其中有3个AO相关的数据结构：AO订阅事件表，AO队列状态表，AO注册表；有1个事件池QMPool，用于保存事件实例，其它事件都是对这个事件实例的引用；有1个时间事件链表QtimeEvt，用于定时事件处理。图 2是相应数据表与QF和AO的关系。

  
**图4.QF框架及五个数据表**

## **4.QF运行**

QF运行时，包括的过程有：  
（1）QF初始化；  
（2）AO和ISR产生事件；  
（3）分发事件到AO队列；  
（4）循环调度分派事件到状态机。  
在把事件放到AO队列中时，会置位AO队列状态表相应位；从AO队列中取出事件时，会清零AO队列状态表相应位。总之，放事件到AO队列或从AO队列取事件，都会修改AO队列状态表。

  
**图5.QF运行过程及数据表关系**

**（1）QF初始化**  
在QF运行之前，首先要进行初始化，要创建AO、创建事件池、创建AO订阅事件表，创建AO队列等一系统开始工作。

  
**图6.QF初始化**

**（2）AO和ISR产生事件**  
在AO或ISR中，用Q\_NEW()宏可以创建用户事件实例，并放到事件池中。  
如用 UserEvt \*pe = Q\_NEW(UserEvt , UserSig1)  
创建了一个用户事件pe。可以分发这个用户事件pe到AO队列中。

其中，UserEvt是增加了参数的用户自定义事件，它是继承自QEvent，UserEvt定义如下，

typedef struct UserEvtTag {  
QEvent  
super; / 继承自QEvent /  
uint8\_t parameter;/增加的事件参数 /  
} UserEvt;

UserSig1是用户定义的信号，用枚举来定义，定义如下，

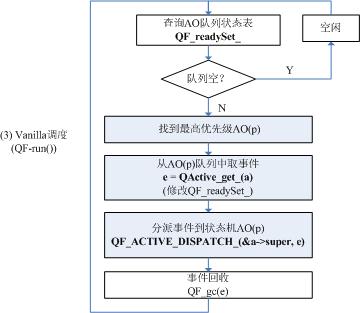
enum UserSignals {  
User1SIG = QUSER\_SIG,/ 用户定义第一个信号 /  
User2\_SIG, / 第二个信号 /  
User3\_SIG, / 第三个信号 /  
...  
  
/ 其它信号 /  
};

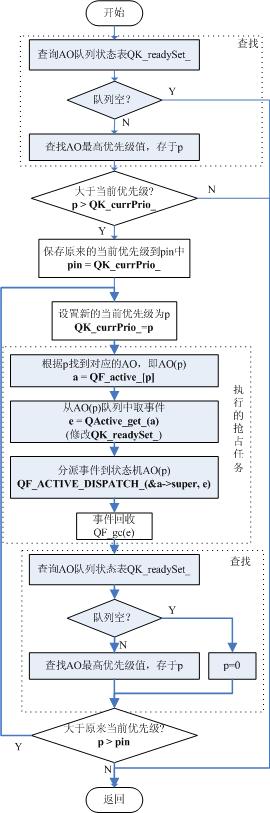
**（3）分发事件到AO队列**  
  
可以用QActive\_postFIFO(TargetAO, (QEvent \*)pe)直接分发事件或用QF\_publish((QEvent \*)pe)间接分发事件到目标AO队列中。使用直接分发事件方式时，不需要AO订阅事件表QF\_subscrList的参与，而间接分发事件时要用到AO订阅事件表。

这两种分发事件都会修改AO队列状态表QF\_readySet\_(QK中，用QK\_readySet)，置位相应位，而在Vanilla调度或QK调度会清除AO队列状态表相应位。

**（4）循环调度分派事件到状态机**  
- 有两种调度方式，一种是超级循环调度Vanilla方式，这种方式的任务是非抢占式的；  
- 另一种方式是QK调度方式，QK是一个微调度内核，这种方式任务是抢占式的。

**两种调度方式的流程：**

  
**图7.Vanilla调度流程**

  
**图8.QK调度流程**

## **5.总结**

QF是一个框架，用于事件的产生、保存、分发事件。有两种事件分发方式，并支持两种对任务的调度方式；可以理解QF为一个软件总线，其上安装了很多的AO和ISR。最多可以支持63个AO+ISR。

GIT使用

1、创建版本库

$ mkdir filename 创建文件夹

$ cd filename 进入版本库文件夹

$ pwd 显示当前目录路径

$ git init 把当前文件夹变成Git管理的库

2、在版本库文件夹添加一个文件

$ git add filename添加文件到暂存区

$ git commit -m "文件说明信息" 把暂存区的文件提交到分支master

3、查看版本库的状态

$ git status 查看版本库的状态

$ git diff filename 查看文件被修改的内容

4、版本回退（时光穿梭）

$ git log 查看提交的日志，以便确定要回退的版本

$ git reset --hard HEAD^ 一个^代表上个版本超过三个便用HEAD~number表示

回退到以前版本后再git log上看不到后来版本的ID 要回到之后版本。

如果记得版本的ID可以直接git reset ID如果不记得了可以用git relog来查看每一次命令

5、工作区和暂存区

$ git add就是把文件提交到暂存区然后用git commit把暂存区的文件提交到分支master

6、撤销修改

$ git checkout -- filename 就是把文件放到工作区或暂存区的修改撤销回到最近一次的

gitcommit或者add状态，当然也可以使用版本回退回到上个版本把这次修改统统抛弃掉。

7、删除文件

$ git rm filename 删除指定文件

QM的使用

1、创建一个qpc工程

2、add a Package 添加一个程序包

3、往Package里add a class 属性 superclass为qpc ::QActive

4、往Package里add Free Operation 属性return type为void

5、添加state\_machine到类

SM状态机（双击show Diagram）

6、在Diagram Toolbox中添加state

7、generation code