

# QPC

---

## 1. 模块

---

### QEP 分层事件处理器

---

QP/C 中，对象行为由**层次化状态机 (UML 状态图)** 定义，其关键在于**状态嵌套**。这种机制的价值在于，它通过允许**子状态仅特化 (定义差异) 于其超状态**的行为，有效避免了传统 FSM 中因重复定义共性行为而导致的**状态-转换爆炸**，从而实现了行为的**高度共享与复用**。

- QHsm class
- QHsm\_ctor()
- QHSM\_INIT()
- QHSM\_DISPATCH()
- QHsm\_isIn()
- QHsm\_state()
- QHsm\_top()
- QMsm class
- QMsm\_ctor()
- QMsm\_isInState()
- QMsm\_stateObj()

### QF 活动对象框架

---

QF 是一个可移植的、事件驱动的、实时框架，用于执行活动对象（并发状态机），专为实时嵌入式 (RTE) 系统而设计。

### Active Objects

- QActive class
- QActive\_ctor()
- QACTIVE\_START()
- QACTIVE\_POST()
- QACTIVE\_POST\_X()
- QACTIVE\_POST\_LIFO()
- QActive\_defer()
- QActive\_recall()
- QActive\_flushDeferred()
- QActive\_stop()
- QMAactive class
- QMAactive\_ctor()

### Publish-Subscribe

- QSubscrList (Subscriber List struct)
- QF\_psInit()
- QF\_PUBLISH()
- QActive\_subscribe()
- QActive\_unsubscribe()

- QActive\_unsubscribeAll()

## Dynamic Events

- QEvt class
- QF\_poolInit()
- Q\_NEW()
- Q\_NEW\_X()
- Q\_NEW\_REF()
- Q\_DELETE\_REF()
- QF\_gc()

## Time Events

- QTimeEvt class
- QF\_TICK\_X()
- QTimeEvt\_ctorX()
- QTimeEvt\_armX()
- QTimeEvt\_disarm()
- QTimeEvt\_rearm()
- QTimeEvt\_ctrl()
- QTicker active object

## Event Queues (raw thread-safe)

- QEQueue class
- QEQueue\_init()
- QEQueue\_post()
- QEQueue\_postLIFO()
- QEQueue\_get()
- QEQueue\_getNFree()
- QEQueue\_getNMin()
- QEQueue\_isEmpty()
- QEQueueCtr()

## Memory Pools

- QMPool class
- QMPool\_init()
- QMPool\_get()
- QMPool\_put()

## QV 协作内核

---

QV 是一个简单的协作内核（以前称为“Vanilla”内核）。该内核一次执行一个活动对象，并在处理每个事件之前执行基于优先级的调度。

## Arm Cortex M

QV 内核是一种协作式 (Cooperative) 内核，其工作原理本质上类似于传统的前台-后台系统 (即“超级循环”):

### 1. 执行模式与上下文

- **活跃对象执行:** 所有的**活跃对象 (Active Objects)** 都在主循环 (背景) 中执行。
- **中断返回:** 中断 (前台) 处理完成后，总是**返回到被抢占的地方继续执行主循环。**
- **处理器模式:**
  - 主循环 (Main Loop) / 应用程序代码 在特权线程模式 (Privileged Thread mode) 下执行。
  - 异常 (包括所有中断) 总是由 特权处理模式 (Privileged Handler mode) 处理。
- **堆栈使用:** QV 内核**只使用主堆栈指针 (Main Stack Pointer)**，不使用也不初始化进程堆栈指针 (Process Stack Pointer)。

### 2. 中断管理与临界区

- **避免竞态条件:** 为了避免主循环和中断之间发生**竞态条件 (race conditions)**，QV 内核会**短暂地禁用中断。**
- **进入中断:** ARM Cortex-M 在进入中断上下文时，**不会自动禁用中断** (即不会设置 `PRIMASK` 或 `BASEPRI`)。
- **ISR 建议:**
  - 一般情况下，**不应该在中断服务程序 (ISR) 内部禁用中断。**
  - 特别是，调用 **QP 服务** (如 `QF_PUBLISH()`、`QF_TICK_X()`、`QACTIVE_POST()` 等) 时，**必须保持中断开启状态**，以避免临界区嵌套问题。
- **中断优先级:** 如果不希望某个中断被其他中断抢占，可以通过配置 **NVIC**，为其设置一个**更高的优先级 (即更小的数值)**。

### 3. 初始化

- **初始化功能:** `QF_init()` 函数会调用 `qv_init()`，将 MCU 中所有可用的 IRQ 的中断优先级设置为一个安全值 `QF_BASEPRI` (主要针对 ARMv7 架构)。

## **qep\_port.h**

包含了 `stdint.h`、`stdbool.h`

## **qf\_port.h**

该文件指定了中断禁用策略 (QF 临界区) 以及 QF 的配置常量

## **qv\_port.h**

- 该文件提供了宏 `QV_CPU_SLEEP()`，该宏指定如何在协作式 QV 内核中安全地进入 CPU 睡眠模式
- 为了避免中断唤醒活动对象和进入睡眠状态之间出现竞争条件，协作式 QV 内核在**禁用中断的情况下**调用 `QV_CPU_SLEEP()` 回调。

## qv\_port.c

该文件定义了函数 QV\_init(), 该函数对于 ARMv7-M 架构, 将所有 IRQ 的中断优先级设置为安全值 QF\_BASEPRI。

## ISR

为活跃对象生成事件 (即调用 `QACTIVE_POST()` 或 `QF_PUBLISH()` 等 QP 服务)

ARM EABI (嵌入式应用程序二进制接口) 要求堆栈 8 字节对齐, 而某些编译器仅保证 4 字节对齐。因此, 一些编译器 (例如 GNU-ARM) 提供了一种将 ISR 函数指定为中断的方法。例如, GNU-ARM 编译器提供了 `attribute((interrupt))` 指定, 可以保证 8 字节堆栈对齐。

```
1 // QF 的时间事件管理
2 void SysTick_Handler(void) __attribute__((__interrupt__));
3 void SysTick_Handler(void) {
4     // ~ ~ ~
5     QF_TICK_X(0U, &_sysTick_Handler); /* process all armed time events */
6 }
```

## FPU

## QV idle

当没有事件可用时, 非抢占式 QV 内核会调用平台特定的回调函数 QV\_onIdle(), 您可以使用该函数来节省 CPU 资源, 或执行任何其他“空闲”处理 (例如 Quantum Spy 软件跟踪输出)。

必须在中断被禁用时被调用(避免与可能投递事件的中断发生竞态条件)

必须在内部重新启用中断(CPU 进入低功耗模式后, 需要中断机制来唤醒)

```
1 void QV_onIdle(void)
2 {
3 #if defined NDEBUG
4     /* Put the CPU and peripherals to the low-power mode */
5     QV_CPU_SLEEP(); /* atomically go to sleep and enable interrupts */
6 #else
7     QF_INT_ENABLE(); /* just enable interrupts */
8 #endif
9 }
```

## API

- `QV_INIT()`
- `QF_run()`
- `QV_onIdle()`
- `QV_CPU_SLEEP()`

## QK 抢占式非阻塞内核

QK 是一个小型抢占式、基于优先级、非阻塞内核，专为执行活动对象而设计。QK 运行活动对象的方式与优先级中断控制器（例如 ARM Cortex-M 中的 NVIC）使用单个堆栈运行中断的方式相同。活动对象以运行至完成 (RTC) 的方式处理其事件，并从调用堆栈中移除自身，这与嵌套中断在完成后从堆栈中移除自身的方式相同。同时，高优先级活动对象可以抢占低优先级活动对象，就像优先级中断控制器下中断可以相互抢占一样。QK 满足速率单调调度（也称为速率单调分析 RMA）的所有要求，可用于硬实时系统。

## QXK 抢占式阻塞内核

## QS 软件追踪组件

## 2. 移植

QP/C 发行版包含许多 QP/C 移植版本，这些版本分为三类：

1. 原生移植版本：使 QP/C 能够“原生”运行在裸机处理器上，使用内置内核（QV、QK 或 QXK）。
2. 第三方 RTOS 移植版本：使 QP/C 能够在第三方实时操作系统（RTOS）上运行。
3. 第三方操作系统移植版本：使 QP/C 能够在第三方操作系统（OS）（例如 Windows 或 Linux）上运行。

## Arm Cortex-M Port

与任何实时内核一样，QP 实时框架需要禁用中断才能访问代码的关键部分，并在访问完成后重新启用中断。

## 中断

QP 框架在 ARM Cortex-M 处理器上采用了**选择性禁用中断**的策略，将中断分为两大类：

- “**内核感知**”中断 (Kernel-Aware)：被允许调用 QP 服务（例如，发布或投递事件）。
- “**内核无感知**”中断 (Kernel-Unaware)：不被允许调用任何 QP 服务。它们只能通过触发一个“**内核感知**”中断来进行间接通信（由该“**内核感知**”中断来投递或发布事件）。

### 1. 针对 Cortex-M3/M4/M7 (ARMv7-M) 架构

- QP 不会完全禁用所有中断，即使在临界区 (Critical Sections) 内。它使用 `BASEPRI` 寄存器来选择性地禁用中断。
- “**内核无感知**”中断 (Kernel-Unaware)：永不被禁用
- “**内核感知**”中断 (Kernel-Aware)：在 QP 临界区内会被禁用

### 2. 针对 Cortex-M0/M0+ (ARMv6-M) 架构

- 实现方式：由于这些架构没有实现 `BASEPRI` 寄存器，QP 必须使用 `PRIMASK` 寄存器来全局禁用中断
- 结果：在此架构下，所有中断都是“**内核感知**”的

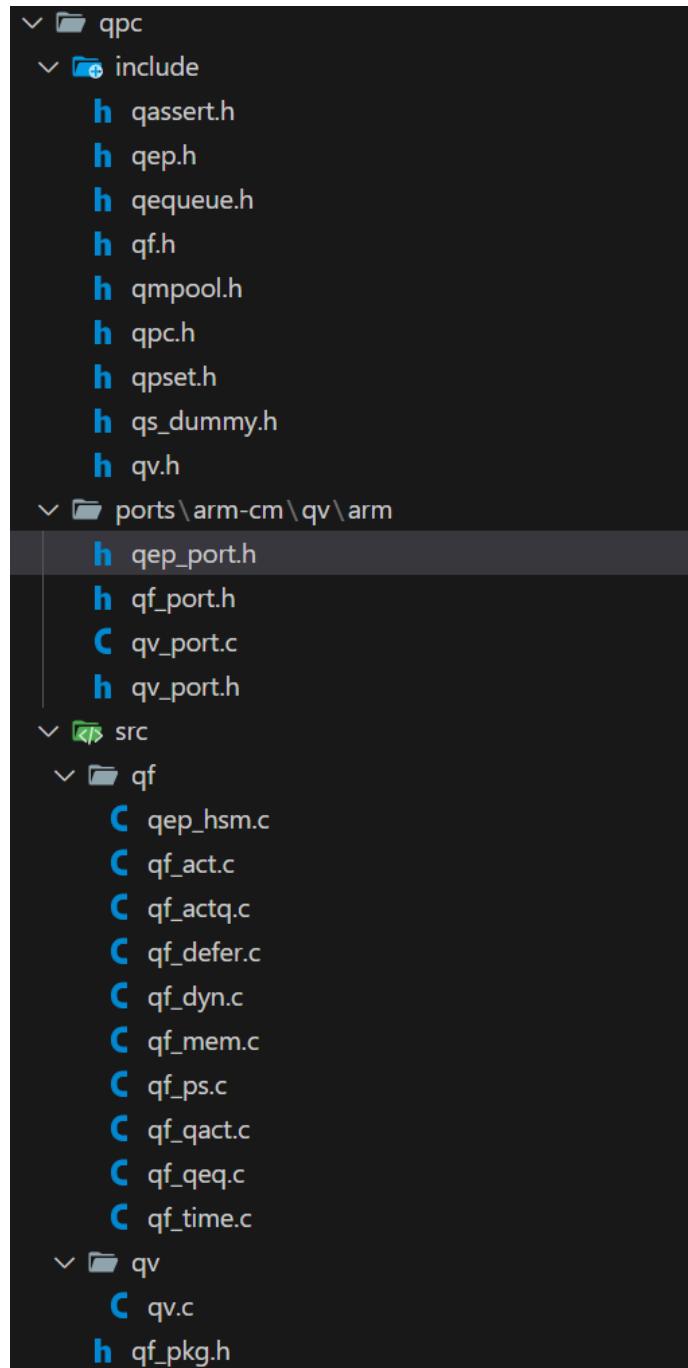
## 注意事项和建议

- **QP 5.9.x 及以上**：`QF_init()` 会将所有 IRQ 的优先级设置为“**内核感知**”值 `QF_BASEPRI`。
- **最佳实践**：强烈建议应用程序在 `QF_onStartup()` 中显式设置所有使用中断的优先级。
- **第三方库风险**：需警惕 STM32Cube 等第三方库可能会意外更改中断优先级和分组，建议在运行 QP 应用前将优先级改回适当的值。
- **设置函数**：应使用 CMSIS 提供的 `NVIC_SetPriority()` 函数来设置每个中断的优先级。请注意，`NVIC_SetPriority()` 传入的值与最终存储在 NVIC 寄存器中的值 (CMSIS priorities vs.

NVIC values) 是不同的。

## 3. 集成

Arm Cortex M 裸机集成qpc (qv) 所需文件



### 注意

1. 任何地方用到QP的仅需包含qpc.h
2. 不需要修改qpc源代码文件

### include

## qassert.h

- 断言失败进入 Q\_onAssert
- QACTIVE\_START 内部使用了 Q\_ASSERT，所以必须调用 Q\_DEFINE\_THIS\_FILE 或 Q\_DEFINE\_THIS\_MODULE

```
1 | #define Q_ASSERT(test_) ((test_) ? (void)0 : Q_onAssert(&Q_this_module_[0],  
__LINE__))
```

## qep.h

```
1 | typedef struct {  
2 |     QSignal sig;           /*!< 事件实例的信号 */  
3 |     uint8_t poolId_;       /*!< 所属事件池 ID (静态事件为 0) */  
4 |     uint8_t volatile refCtr_; /*!< 引用计数器 */  
5 | } QEvt;  
6 |  
7 | // 修改目标状态为target  
8 | #define Q_TRAN(target_) \  
9 |     ((Q_HSM_UPCAST(me))->temp.fun = Q_STATE_CAST(target_),  
10 |      (Qstate)Q_RET_TRAN)  
11 | // 修改目标状态为super  
12 | #define Q_SUPER(super_) \  
13 |     ((Q_HSM_UPCAST(me))->temp.fun = Q_STATE_CAST(super_),  
14 |      (Qstate)Q_RET_SUPER)
```

- poolId\_ 为0表示静态事件，此时 refctr\_ 不用于引用计数

## queue.h

```
1 |
```

## qf.h

```
1 | #include "qpset.h"
```

- QActive

```
1 | // 活动对象基类 (基于 ::QHsm 实现)  
2 | struct QActive {  
3 |     QHsm super;  
4 |     QEQueue eQueue;  
5 |     uint8_t prio;  
6 | };  
7 |  
8 | // QActive 类虚表  
9 | struct QActivevtable {  
10 |     // [virtual] 启动活动对象  
11 |     void (*start)(QActive *const me, uint_fast8_t prio,  
12 |                     QEvt const **const qSto, uint_fast16_t const qLen,
```

```

13         void *const stksto, uint_fast16_t const stksize,
14         void const *const par);
15     // [virtual] FIFO 异步发送事件给活动对象
16     bool (*post)(QActive *const me, QEvt const *const e,
17                   uint_fast16_t const margin);
18     // [virtual] LIFO 异步发送事件给活动对象
19     void (*postLIFO)(QActive *const me, QEvt const *const e);
20 };
21
22
23 // QM 建模工具使用
24 typedef struct {
25     QActive super;
26 } QMActive;
27 typedef QActiveVtable QMActiveVtable;
28 void QMActive_ctor(QMActive *const me, QStateHandler initial);
29
30
31 // QTicker 是一个高效的活动对象，专门用于以指定 tick 频率
32 typedef struct {
33     QActive super; /*!< inherits ::QActive */
34 } QTicker;
35 void QTicker_ctor(QTicker *const me, uint_fast8_t tickRate);

```

## public 函数

```

1 // QActive public 函数
2 #define QACTIVE_START(...)
3 #define QACTIVE_POST(...)          // 不会断言失败(FIFO)
4 #define QACTIVE_POST_X(...)        // 会断言失败(FIFO)
5 #define QACTIVE_POST_LIFO(...)

```

## protected 函数

```

1 // QActive protected 函数
2 void QActive_ctor(QActive *const me, QStateHandler initial);
3 void QActive_stop(QActive *const me);
4
5 // 订阅信号 @p sig, 以便传递给活动对象 @p me
6 void QActive_subscribe(QActive const *const me, enum_t const sig);
7
8 // 取消订阅信号 @p sig, 使其不再传递给活动对象 @p me
9 void QActive_unsubscribe(QActive const *const me, enum_t const sig);
10
11 // 取消订阅所有信号, 使其不再传递给活动对象 @p me
12 void QActive_unsubscribeAll(QActive const *const me);
13
14 // 将事件 @p e 延迟存储到指定的事件队列 @p eq 中
15 bool QActive_defer(QActive const *const me, QEQueue *const eq, QEvt const
16 *const e);
17
18 // 从指定的事件队列 @p eq 中取回一个之前延迟的事件
19 bool QActive_recall(QActive *const me, QEQueue *const eq);

```

```

20 // 清空指定的延迟队列 @p eq
21 uint_fast16_t QActive_flushDeferred(QActive const *const me, QEQueue *const
eq);
22
23 // 通用的附加属性设置 (useful in QP ports)
24 void QActive_setAttr(QActive *const me, uint32_t attr1, void const *attr2);

```

- QTimeEvt 事件事件类

```

1 typedef struct QTimeEvt {
2     QEvt super;                      // inherits ::QEvt
3     struct QTimeEvt *volatile next;   // 指向链表中下一个时间事件
4     void *volatile act;              // 接收时间事件的活动对象
5     QTimeEvtCtr volatile ctr;        // 计数器
6     QTimeEvtCtr interval;           // 重载值
7 } QTimeEvt;

```

## public 函数

```

1 // 构造函数, 初始化时间事件
2 void QTimeEvt_ctorX(QTimeEvt *const me, QActive *const act,
3                      enum_t const sig, uint_fast8_t tickRate);
4
5 // 启动一个时间事件(单次或周期性),并直接投递事件
6 void QTimeEvt_armX(QTimeEvt *const me,
7                      QTimeEvtCtr const nTicks, QTimeEvtCtr const interval);
8
9 // 重新启动一个时间事件
10 bool QTimeEvt_rearm(QTimeEvt *const me, QTimeEvtCtr const nTicks);
11
12 // 取消启动一个时间事件
13 bool QTimeEvt_disarm(QTimeEvt *const me);
14
15 // 检查时间事件是否"被取消"
16 bool QTimeEvt_wasDisarmed(QTimeEvt *const me);
17
18 // 获取时间事件当前的递减计数值
19 QTimeEvtCtr QTimeEvt_currCtr(QTimeEvt const *const me);

```

- QF facilities

```

1 // 订阅列表
2 typedef QPSet QSubscrList;
3
4 /* public functions */
5 void QF_init(void);
6
7 /*! 发布-订阅机制初始化 */
8 void QF_psInit(QSubscrList *const subscrsto, enum_t const maxSignal);
9
10 /*! 事件池初始化, 用于事件的动态分配 */
11 void QF_poolInit(void *const poolsto, uint_fast32_t const poolsize,
12                  uint_fast16_t const evtSize);
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
305
306
307
308
308
309
309
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372

```

```
13 /*! 获取任意已注册事件池的块大小 */
14 uint_fast16_t QF_poolGetMaxBlockSize(void);
15
16 /*! 将控制权交给 QF 以运行应用程序 */
17 int_t QF_run(void);
18
19 /*! 应用层调用该函数以停止 QF 应用程序，并将控制权交还给 OS/内核 */
20 void QF_stop(void);
21
22
23 // QF 回调
24 void QF_onStartup(void);
25 void QF_onCleanup(void);
26
27
28 // 事件发布
29 void QF_publish_(QEvt const *const e);
30 #define QF_PUBLISH(e_, dummy_) (QF_publish_(e_))
31
32
33 // 在时钟节拍中处理事件事件
34 void QF_tickX_(uint_fast8_t const tickRate);
35 #define QF_TICK_X(tickRate_, dummy) (QF_tickX_(tickRate_))
36 // 如果在指定的时钟速率下没有已启动的时间事件，则返回 'true'
37 bool QF_noTimeEvtsActiveX(uint_fast8_t const tickRate);
38
39
40 /*! 注册一个活动对象，使其由框架管理 */
41 void QF_add_(QActive *const a);
42
43 /*! 将活动对象从框架中移除 */
44 void QF_remove_(QActive *const a);
45
46 /*! 获取指定事件池的最小剩余空闲条目数 */
47 uint_fast16_t QF_getPoolMin(uint_fast8_t const poolId);
48
49 /*! 获取指定事件队列的最小剩余空闲条目数 */
50 uint_fast16_t QF_getQueueMin(uint_fast8_t const prio);
51
52
53 /*! 内部 QF 实现：创建新的动态事件 */
54 QEvt *QF_newX_(uint_fast16_t const evtsize, uint_fast16_t const margin,
55 enum_t const sig);
56 // 分配一个动态事件（断言版本）
57 #define Q_NEW(evtT_, sig_)
58 // 分配一个动态事件（非断言版本）
59 #define Q_NEW_X(e_, evtT_, margin_, sig_)
60
61 /*! 内部 QF 实现：创建新的事件引用 */
62 QEvt const *QF_newRef_(QEvt const *const e, void const *const evtRef);
63 /*! 内部 QF 实现：删除事件引用 */
64 void QF_deleteRef_(void const *const evtRef);
65 // 创建当前事件 `e` 的新引用
66 #define Q_NEW_REF(evtRef_, evtT_)
```

```
67 | #define Q_DELETE_REF(evtRef_)
```

## qmpool.h

```
1 |
```

## qpc.h

```
1 | #include "qf_port.h"      /* QF/C port from the port directory */
2 | #include "qassert.h"      /* QP embedded systems-friendly assertions */
```

## qpset.h

```
1 |
```

## qv.h

协作式内核

```
1 | #include "equeue.h"    /* QV kernel uses the native QP event queue */
2 | #include "qmpool.h"    /* QV kernel uses the native QP memory pool */
3 | #include "qpset.h"     /* QV kernel uses the native QP priority set */
4 |
5 | #define QF_EQUEUE_TYPE      QEQueue
6 | // QF_run() 中调用
7 | void QV_onIdle(void);
8 |
9 | /* QV 内核特有的调度器加锁机制 (但在 QV 中不需要) */
10 | #define QF_SCHED_STAT_
11 | #define QF_SCHED_LOCK_(dummy) ((void)0)
12 | #define QF_SCHED_UNLOCK_(dummy) ((void)0)
13 |
14 | /* QF 原生事件队列操作 */
15 | #define QACTIVE_EQUEUE_WAIT_(me_) \
16 |     Q_ASSERT_ID(0, (me_)->eQueue.frontEvt != (QEvt *)0)
17 |
18 | #define QACTIVE_EQUEUE_SIGNAL_(me_) \
19 |     QPSet_insert(&QV_readySet_, (uint_fast8_t)(me_)->prio)
20 |
21 |
22 |
23 | /* QF 原生事件池操作 */
24 | #define QF_EPOOL_TYPE_ QMPool
25 |
26 | // QF_EPOOL_INIT_ 事件池初始化
27 | #define QF_EPOOL_INIT_(p_, poolsto_, poolsize_, evtsize_) \
28 |     (QMPool_init(&(p_), (poolsto_), (poolsize_), (evtsize_)))
29 |
30 | // QF_EPOOL_EVENT_SIZE_ 事件池大小
31 | #define QF_EPOOL_EVENT_SIZE_(p_) ((uint_fast16_t)(p_).blocksize)
```

```
32 // QF_EPOOL_GET_
33 #define QF_EPOOL_GET_(p_, e_, m_, qs_id_) \
34     ((e_) = (QEvt *)QMPool_get(&(p_), (m_), (qs_id_)))
35
36 // QF_EPOOL_PUT_
37 #define QF_EPOOL_PUT_(p_, e_, qs_id_) \
38     (QMPool_put(&(p_), (e_), (qs_id_)))
```

## qpc.h

```
1 #include "qf_port.h"      /* QF/C port from the port directory */
2 #include "qassert.h"       /* QP embedded systems-friendly assertions */
```

## ports

移植目录

### qep\_port.h

```
1 #include <stdint.h> /* Exact-width types. WG14/N843 C99 Standard */
2 #include <stdbool.h> /* Boolean type.      WG14/N843 C99 Standard */
3 #include "qep.h"       /* QEP platform-independent public interface */
```

### qf\_port.h

```
1 #include "qep_port.h" /* QEP port */
2 #include "qv_port.h"  /* QV cooperative kernel port */
3 #include "qf.h"        /* QF platform-independent public interface */
```

宏 **QF\_AWARE\_ISR\_CMSIS\_PRI** 在应用程序中用于作为枚举"QF-aware"中断优先级的偏移量.

"QF aware"的中断:

- 优先级大于等于 **QF\_AWARE\_ISR\_CMSIS\_PRI**
- 可以调用 QF 服务

"QF unaware"的中断:

- 优先级小于 **QF\_AWARE\_ISR\_CMSIS\_PRI**
- 不能调用任何 QF 服务

### qv\_port.h

```
1 #include "qv.h" /* QV platform-independent public interface */
```

## src

## qep\_hsm.c

```
1 // 保留事件
2 static QEvt const QEP_reservedEvt_[] = {
3     { (QSignal)QEP_EMPTY_SIG_, 0U, 0U },
4     { (QSignal)Q_ENTRY_SIG,    0U, 0U },
5     { (QSignal)Q_EXIT_SIG,    0U, 0U },
6     { (QSignal)Q_INIT_SIG,    0U, 0U }
7 };
8
9 /** 在一个状态转换函数中执行 保留事件动作
10 *  QEP_TRIG_(me->temp.fun, QEP_EMPTY_SIG_)
11 *  QEP_TRIG_(t, Q_INIT_SIG)
12 */
13 #define QEP_TRIG_(state_, sig_) ((*(state_))(me, &QEP_reservedEvt_[(sig_)]))
14
15 // 状态处理函数执行退出动作Q_EXIT_SIG
16 #define QEP_EXIT_(state_, qs_id_) QEP_TRIG_(state_, Q_EXIT_SIG)
17
18 // 状态处理函数执行进入动作Q_ENTRY_SIG
19 #define QEP_ENTER_(state_, qs_id_) QEP_TRIG_(state_, Q_ENTRY_SIG)
```

```
1 void QHsm_ctor(QHsm * const me, QStateHandler initial) {
2     /* QHsm virtual table */
3     static struct QHsmVtable const vtable = {
4         &QHsm_init_,
5         &QHsm_dispatch_
6     };
7     me->vptr      = &vtable;
8     me->state.fun = Q_STATE_CAST(&QHsm_top);
9     me->temp.fun  = initial;
10 }
11
12 // 最顶层初始转换 QHsm_top → MyState_Initial → MyState_Active
13 void QHsm_init_(QHsm * const me, void const * const e)
14 {
15     QStateHandler t = me->state.fun;
16     QState r;
17
18     /** @pre the virtual pointer must be initialized, the top-most initial
19      * transition must be initialized, and the initial transition must not
20      * be taken yet.
21      */
22     Q_REQUIRE_ID(200, (me->vptr != (struct QHsmVtable *)0)
23                 && (me->temp.fun != Q_STATE_CAST(0))
24                 && (t == Q_STATE_CAST(&QHsm_top)));
25
26     /* execute the top-most initial tran. */
27     r = (*me->temp.fun)(me, Q_EVT_CAST(QEvt));
28
29     /* the top-most initial transition must be taken */
30     Q_ASSERT_ID(210, r == (QState)Q_RET_TRAN);
31
32     /* drill down into the state hierarchy with initial transitions... */
```

```

33     do {
34         QStateHandler path[QHSM_MAX_NEST_DEPTH_]; /* tran entry path array
35     */
36
37     int_fast8_t ip = 0; /* tran entry path index */
38
39     path[0] = me->temp.fun;
40     (void)QEP_TRIG_(me->temp.fun, QEP_EMPTY_SIG_);
41     while (me->temp.fun != t) {
42         ++ip;
43         Q_ASSERT_ID(220, ip < (int_fast8_t)Q_DIM(path));
44         path[ip] = me->temp.fun;
45         (void)QEP_TRIG_(me->temp.fun, QEP_EMPTY_SIG_);
46     }
47     me->temp.fun = path[0];
48
49     /* retrace the entry path in reverse (desired) order... */
50     do {
51         QEP_ENTER_(path[ip], qs_id); /* enter path[ip] */
52         --ip;
53     } while (ip >= 0);
54
55     t = path[0]; /* current state becomes the new source */
56
57     r = QEP_TRIG_(t, Q_INIT_SIG); /* execute initial transition */
58
59     } while (r == (QState)Q_RET_TRAN);
60
61     me->state.fun = t; /* change the current active state */
62     me->temp.fun = t; /* mark the configuration as stable */
63 }
64
65 QState QHsm_top(void const * const me, QEvt const * const e) {
66     (void)me;
67     (void)e;
68     return (QState)Q_RET_IGNORED; /* the top state ignores all events */
69 }
70
71 void QHsm_dispatch_(QHsm * const me, QEvt const * const e) {
72     QStateHandler t = me->state.fun;
73     QStateHandler s;
74     QState r;
75
76     /** @pre the current state must be initialized and
77      * the state configuration must be stable
78      */
79     Q_REQUIRE_ID(400, (t != Q_STATE_CAST(0))
80                 && (t == me->temp.fun));
81
82     /* process the event hierarchically... */
83     do {
84         s = me->temp.fun;
85         r = (*s)(me, e); /* invoke state handler s */
86
87         if (r == (QState)Q_RET_UNHANDLED) { /* unhandled due to a guard? */
88             r = QEP_TRIG_(s, QEP_EMPTY_SIG_); /* find superstate of s */

```

```

87 }
88 } while (r == (QState)Q_RET_SUPER);
89
90 /* transition taken? */
91 if (r >= (QState)Q_RET_TRAN) {
92     QStateHandler path[QHSM_MAX_NEST_DEPTH_];
93     int_fast8_t ip;
94
95     path[0] = me->temp.fun; /* save the target of the transition */
96     path[1] = t;
97     path[2] = s;
98
99     /* exit current state to transition source s... */
100    for (; t != s; t = me->temp.fun) {
101        if (QEP_TRIG_(t, Q_EXIT_SIG) == (QState)Q_RET_HANDLED) {
102            (void)QEP_TRIG_(t, QEP_EMPTY_SIG_); /* find superstate of t
103        */
104    }
105
106    ip = QHsm_tran_(me, path);
107
108    /* retrace the entry path in reverse (desired) order... */
109    for (; ip >= 0; --ip) {
110        QEP_ENTER_(path[ip], qs_id); /* enter path[ip] */
111    }
112
113    t = path[0]; /* stick the target into register */
114    me->temp.fun = t; /* update the next state */
115
116    /* drill into the target hierarchy... */
117    while (QEP_TRIG_(t, Q_INIT_SIG) == (QState)Q_RET_TRAN) {
118        ip = 0;
119        path[0] = me->temp.fun;
120
121        (void)QEP_TRIG_(me->temp.fun, QEP_EMPTY_SIG_); /*find superstate
122    */
123
124        while (me->temp.fun != t) {
125            ++ip;
126            path[ip] = me->temp.fun;
127            (void)QEP_TRIG_(me->temp.fun, QEP_EMPTY_SIG_); /* find super
128        */
129    }
130    me->temp.fun = path[0];
131
132    /* entry path must not overflow */
133    Q_ASSERT_ID(410, ip < QHSM_MAX_NEST_DEPTH_);
134
135    /* retrace the entry path in reverse (correct) order... */
136    do {
137        QEP_ENTER_(path[ip], qs_id); /* enter path[ip] */
138        --ip;
139    } while (ip >= 0);

```

```

139         t = path[0]; /* current state becomes the new source */
140     }
141 }
142
143 me->state.fun = t; /* change the current active state */
144 me->temp.fun = t; /* mark the configuration as stable */
145 }
146
147 // LCA 最近公共祖先
148 static int_fast8_t QHsm_tran_(QHsm * const me, QStateHandler
path[QHSM_MAX_NEST_DEPTH_]) {
149     int_fast8_t ip = -1; /* transition entry path index */
150     int_fast8_t iq; /* helper transition entry path index */
151     QStateHandler t = path[0];
152     QStateHandler const s = path[2];
153     QState r;
154
155     /* (a) check source==target (transition to self)... */
156     if (s == t) {
157         QEP_EXIT_(s, qs_id); /* exit the source */
158         ip = 0; /* enter the target */
159     }
160     else {
161         (void)QEP_TRIG_(t, QEP_EMPTY_SIG_); /* find superstate of target */
162
163         t = me->temp.fun;
164
165         /* (b) check source==target->super... */
166         if (s == t) {
167             ip = 0; /* enter the target */
168         }
169         else {
170             (void)QEP_TRIG_(s, QEP_EMPTY_SIG_); /* find superstate of src */
171         }
172
173         /* (c) check source->super==target->super... */
174         if (me->temp.fun == t) {
175             QEP_EXIT_(s, qs_id); /* exit the source */
176             ip = 0; /* enter the target */
177         }
178         else {
179             /* (d) check source->super==target... */
180             if (me->temp.fun == path[0]) {
181                 QEP_EXIT_(s, qs_id); /* exit the source */
182             }
183             else {
184                 /* (e) check rest of source==target->super->super...
185                  * and store the entry path along the way
186                  */
187                 iq = 0; /* indicate that LCA not found */
188                 ip = 1; /* enter target and its superstate */
189                 path[1] = t; /* save the superstate of target */
190                 t = me->temp.fun; /* save source->super */
191
192                 /* find target->super->super... */

```

```

192     r = QEP_TRIG_(path[1], QEP_EMPTY_SIG_);
193     while (r == (QState)Q_RET_SUPER) {
194         ++ip;
195         path[ip] = me->temp.fun; /* store the entry path */
196         if (me->temp.fun == s) { /* is it the source? */
197             iq = 1; /* indicate that LCA found */
198
199             /* entry path must not overflow */
200             Q_ASSERT_ID(510,
201                         ip < QHSM_MAX_NEST_DEPTH_);
202             --ip; /* do not enter the source */
203             r = (QState)Q_RET_HANDLED; /* terminate loop */
204         }
205         /* it is not the source, keep going up */
206     else {
207         r = QEP_TRIG_(me->temp.fun, QEP_EMPTY_SIG_);
208     }
209 }
210
211 /* the LCA not found yet? */
212 if (iq == 0) {
213
214     /* entry path must not overflow */
215     Q_ASSERT_ID(520, ip < QHSM_MAX_NEST_DEPTH_);
216
217     QEP_EXIT_(s, qs_id); /* exit the source */
218
219     /* (f) check the rest of source->super
220      *          == target->super->super...
221      */
222     iq = ip;
223     r = (QState)Q_RET_IGNORED; /* LCA NOT found */
224     do {
225         if (t == path[iq]) { /* is this the LCA? */
226             r = (QState)Q_RET_HANDLED; /* LCA found */
227             ip = iq - 1; /* do not enter LCA */
228             iq = -1; /* cause termination of the loop */
229         }
230     else {
231         --iq; /* try lower superstate of target */
232     }
233 } while (iq >= 0);
234
235 /* LCA not found? */
236 if (r != (QState)Q_RET_HANDLED) {
237     /* (g) check each source->super->...
238      * for each target->super...
239      */
240     r = (QState)Q_RET_IGNORED; /* keep looping */
241     do {
242         /* exit t unhandled? */
243         if (QEP_TRIG_(t, Q_EXIT_SIG)
244             == (QState)Q_RET_HANDLED)
245         {

```

```

247             (void)QEP_TRIG_(t, QEP_EMPTY_SIG_);
248         }
249         t = me->temp.fun; /* set to super of t */
250         iq = ip;
251         do {
252             /* is this LCA? */
253             if (t == path[iq]) {
254                 /* do not enter LCA */
255                 ip = (int_fast8_t)(iq - 1);
256                 iq = -1; /* break out of inner loop
257             */
258             /* break out of outer loop */
259             r = (QState)Q_RET_HANDLED;
260         }
261         else {
262             --iq;
263         }
264     } while (iq >= 0);
265     } while (r != (QState)Q_RET_HANDLED);
266 }
267 }
268 }
269 }
270 }
271 return ip;
272 }
```

状态配置稳定: me->temp.fun == me->state.fun

## 转换规则

### qv.c

**QF\_Init** 内部调用了 **QV\_INIT**

### qf\_actq.c

活动对象队列

### qf\_qact.c

活动对象

### qf\_qep.c

事件队列

## qf\_time.c

```
1 // 启动定时器
2 void QTimeEvt_armX(QTimeEvt * const me,
3                      QTimeEvtCtr const nTicks, QTimeEvtCtr const interval);
```

时间事件QTimEvt 是静态事件

## NOTE

1. 状态处理函数中初始化 `qstate status = Q_HANDLED();`, 以防信号处理后未初始化status
2. 订阅 (`QActive_subscribe()`) 的信号 只能通过 `QF_PUBLISH()` 投递 (发布) 。  
而 `QACTIVE_POST()` 是直接发送给某一个活动对象的, 不走订阅系统。
3. 静态事件和动态事件(事件池)
  - 静态事件

```
1 static QEvt evt = {SIG_LED_TOGGLE, 0, 0};
2 QACTIVE_POST(AO_LED, &evt, 0);
```

- 动态事件

```
1 QF_PoolInit(poolsto, poolsize, sizeof(MyEvt));
2 MyEvt *pe = QF_NEW(MyEvt, SIG_MY_EVENT);
3 QACTIVE_POST(AO, &pe->super, 0); // 或 QF_PUBLISH
```

1. **QTimeEvt\_armX 400错误**: 定时器**重复启动**导致断言失败 `Q_REQUIRE_ID(400, t->ctr == 0u);`
4. 使用QF\_NO\_MARGIN的函数会有断言失败机制
5. 使用 `QACTIVE_START` 同文件必须定义 `Q_DEFINE_THIS_FILE`