

```
*****  
25289 Tue Nov 20 20:17:43 2018  
new/usr/src/uts/common/io/cxgbe/t4nex/adapter.h
```

```
9994 cxgbe t4nex: Handle get_f1_payload() alloc failures
```

```
9995 cxgbe t4_devo_attach() should initialize ->sfl
```

```
*****  
unchanged_portion_omitted
```

```
259 #define FL_RUNNING_LOW(f1)      (f1->cap - f1->needed <= f1->lowat)  
260 #define FL_NOT_RUNNING_LOW(f1)   (f1->cap - f1->needed >= 2 * f1->lowat)  
  
262 struct sge_f1 {  
263     unsigned int flags;  
264     kmutex_t lock;  
265     ddi_dma_handle_t dhdl;  
266     ddi_acc_handle_t ahdl;  
  
268     __be64 *desc;           /* KVA of descriptor ring, ptr to addresses */  
269     uint64_t ba;            /* bus address of descriptor ring */  
270     struct fl_sdesc *sdesc; /* KVA of software descriptor ring */  
271     uint32_t cap;           /* max # of buffers, for convenience */  
272     uint16_t qsize;          /* size (# of entries) of the queue */  
273     uint16_t ctxtx_id;       /* SGE context id for the freelist */  
274     uint32_t cidx;           /* consumer idx (buffer idx, NOT hw desc idx) */  
275     uint32_t pidx;           /* producer idx (buffer idx, NOT hw desc idx) */  
276     uint32_t needed;         /* # of buffers needed to fill up f1. */  
277     uint32_t lowat;          /* # of buffers <= this means f1 needs help */  
278     uint32_t pending;        /* # of bufs allocated since last doorbell */  
279     uint32_t offset;          /* current packet within the larger buffer */  
280     uint16_t copy_threshold; /* anything this size or less is copied up */  
  
282     uint64_t copied_up;      /* # of frames copied into mblk and handed up */  
283     uint64_t passed_up;      /* # of frames wrapped in mblk and handed up */  
284     uint64_t allocb_fail;    /* # of mblk allocation failures */  
  
286     TAILQ_ENTRY(sge_f1) link; /* All starving freelists */  
287 };  
unchanged_portion_omitted
```

```
*****
75026 Tue Nov 20 20:17:43 2018
new/usr/src/uts/common/io/cxgbe/t4nex/t4_nexus.c
9994 cxgbe t4nex: Handle get_f1_payload() alloc failures
9995 cxgbe t4_devo_attach() should initialize ->sfl
*****
_____ unchanged_portion_omitted_



276 static int
277 t4_devo_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
278 {
279     struct adapter *sc = NULL;
280     struct sge *s;
281     int i, instance, rc = DDI_SUCCESS, rqidx, tqidx, q;
282     int irq = 0, nxg, n100g, n40g, n25g, n10g, nlg;
283 #ifdef TCP_OFFLOAD_ENABLE
284     int ofld_rqidx, ofld_tqidx;
285 #endif
286     char name[16];
287     struct driver_properties *prp;
288     struct intrs_and_queues iaq;
289     ddi_device_acc_attr_t da = {
290         .devacc_attr_version = DDI_DEVICE_ATTR_V0,
291         .devacc_attr_endian_flags = DDI_STRUCTURE_LE_ACC,
292         .devacc_attr_dataorder = DDI_UNORDERED_OK_ACC
293     };
294     ddi_device_acc_attr_t dal = {
295         .devacc_attr_version = DDI_DEVICE_ATTR_V0,
296         .devacc_attr_endian_flags = DDI_STRUCTURE_LE_ACC,
297         .devacc_attr_dataorder = DDI_MERGING_OK_ACC
298     };
299
300     if (cmd != DDI_ATTACH)
301         return (DDI_FAILURE);
302
303     /*
304      * Allocate space for soft state.
305      */
306     instance = ddi_get_instance(dip);
307     rc = ddi_soft_state_zalloc(t4_list, instance);
308     if (rc != DDI_SUCCESS) {
309         cxgb_printf(dip, CE_WARN,
310                     "failed to allocate soft state: %d", rc);
311         return (DDI_FAILURE);
312     }
313
314     sc = ddi_get_soft_state(t4_list, instance);
315     sc-> dip = dip;
316     sc-> dev = makedevice(ddi_driver_major(dip), instance);
317     mutex_init(&sc-> lock, NULL, MUTEX_DRIVER, NULL);
318     cv_init(&sc-> cv, NULL, CV_DRIVER, NULL);
319     mutex_init(&sc-> sfl_lock, NULL, MUTEX_DRIVER, NULL);
320     TAILQ_INIT(&sc-> sfl);
321
322     mutex_enter(&t4_adapter_list_lock);
323     SLIST_INSERT_HEAD(&t4_adapter_list, sc, link);
324     mutex_exit(&t4_adapter_list_lock);
325
326     sc-> pf = getpf(sc);
327     if (sc-> pf > 8) {
328         rc = EINVAL;
329         cxgb_printf(dip, CE_WARN,
330                     "failed to determine PCI PF# of device");
331         goto done;
332     }
333     sc-> mbox = sc-> pf;
```

```
335     /* Initialize the driver properties */
336     prp = &sc-> props;
337     (void) init_driver_props(sc, prp);
338
339     /*
340      * Enable access to the PCI config space.
341      */
342     rc = pci_config_setup(dip, &sc-> pci_regh);
343     if (rc != DDI_SUCCESS) {
344         cxgb_printf(dip, CE_WARN,
345                     "failed to enable PCI config space access: %d", rc);
346         goto done;
347     }
348
349     /* TODO: Set max read request to 4K */
350
351     /*
352      * Enable MMIO access.
353      */
354     rc = ddi_regs_map_setup(dip, 1, &sc-> regp, 0, 0, &da, &sc-> regh);
355     if (rc != DDI_SUCCESS) {
356         cxgb_printf(dip, CE_WARN,
357                     "failed to map device registers: %d", rc);
358         goto done;
359     }
360
361     (void) memset(sc-> chan_map, 0xff, sizeof (sc-> chan_map));
362
363     /*
364      * Initialize cpl handler.
365      */
366     for (i = 0; i < ARRAY_SIZE(sc-> cpl_handler); i++) {
367         sc-> cpl_handler[i] = cpl_not_handled;
368     }
369
370     for (i = 0; i < ARRAY_SIZE(sc-> fw_msg_handler); i++) {
371         sc-> fw_msg_handler[i] = fw_msg_not_handled;
372     }
373
374     for (i = 0; i < NCHAN; i++) {
375         (void) snprintf(name, sizeof (name), "%s-%d",
376                         "reclaim", i);
377         sc-> tq[i] = ddi_taskq_create(sc-> dip,
378                                       name, 1, TASKQ_DEFAULTPRI, 0);
379
380         if (sc-> tq[i] == NULL) {
381             cxgb_printf(dip, CE_WARN,
382                         "failed to create task queues");
383             rc = DDI_FAILURE;
384             goto done;
385         }
386     }
387
388     /*
389      * Prepare the adapter for operation.
390      */
391     rc = -t4_prep_adapter(sc, false);
392     if (rc != 0) {
393         cxgb_printf(dip, CE_WARN, "failed to prepare adapter: %d", rc);
394         goto done;
395     }
396
397     /*
398      * Enable BAR1 access.
399      */
```

```

400     sc->doorbells |= DOORBELL_KDB;
401     rc = ddi_regs_map_setup(dip, 2, &sc->reglp, 0, 0, &dal, &sc->reglh);
402     if (rc != DDI_SUCCESS) {
403         cxgb_printf(dip, CE_WARN,
404                     "failed to map BAR1 device registers: %d", rc);
405         goto done;
406     } else {
407         if (is_t5(sc->params.chip)) {
408             sc->doorbells |= DOORBELL_UDB;
409             if (prp->wc) {
410                 /*
411                  * Enable write combining on BAR2. This is the
412                  * userspace doorbell BAR and is split into 128B
413                  * (UDBS_SEG_SIZE) doorbell regions, each associ-
414                  * ated with an egress queue. The first 64B has the
415                  * and the second 64B can be used to submit a tx
416                  * request with an implicit doorbell.
417                  */
418             sc->doorbells &= ~DOORBELL_UDB;
419             sc->doorbells |= (DOORBELL_WCWR |
420                               DOORBELL_UDBWC);
421             t4_write_reg(sc, A_SGE_STAT_CFG,
422                          V_STATSOURCE_T5(7) | V_STATMODE(0));
423         }
424     }
425
426     /*
427      * Do this really early. Note that minor number = instance.
428      */
429     (void) sprintf(name, sizeof(name), "%s,%d", T4_NEXUS_NAME, instance);
430     rc = ddi_create_minor_node(dip, name, S_IFCHR, instance,
431                               DDI_NT_NEXUS, 0);
432     if (rc != DDI_SUCCESS) {
433         cxgb_printf(dip, CE_WARN,
434                     "failed to create device node: %d", rc);
435         rc = DDI_SUCCESS; /* carry on */
436     }
437
438     /* Do this early. Memory window is required for loading config file. */
439     setup_memwin(sc);
440
441     /* Prepare the firmware for operation */
442     rc = prep_firmware(sc);
443     if (rc != 0)
444         goto done; /* error message displayed already */
445
446     rc = adap_pre_init_tweaks(sc);
447     if (rc != 0)
448         goto done;
449
450     rc = get_params_pre_init(sc);
451     if (rc != 0)
452         goto done; /* error message displayed already */
453
454     t4_sge_init(sc);
455
456     if (sc->flags & MASTER_PF) {
457         /* get basic stuff going */
458         rc = -t4_fw_initialize(sc, sc->mbox);
459         if (rc != 0) {
460             cxgb_printf(sc->dip, CE_WARN,
461                         "early init failed: %d.\n", rc);
462             goto done;
463         }
464     }
465 }
```

```

467     rc = get_params_post_init(sc);
468     if (rc != 0)
469         goto done; /* error message displayed already */
470
471     rc = set_params_post_init(sc);
472     if (rc != 0)
473         goto done; /* error message displayed already */
474
475     /*
476      * TODO: This is the place to call t4_set_filter_mode()
477      */
478
479     /* tweak some settings */
480     t4_write_reg(sc, A_TP_SHIFT_CNT, V_SYNSHIFTMAX(6) | V_RXTSIFTMAXR1(4) |
481                  V_RXTSIFTMAXR2(15) | V_PERSHIFTBACKOFFMAX(8) | V_PERSHIFTMAX(8) |
482                  V_KEEPALIVEMAXR1(4) | V_KEEPALIVEMAXR2(9));
483     t4_write_reg(sc, A_ULP_RX_TDDP_PSZ, V_HPZ0(PAGE_SHIFT - 12));
484
485     /*
486      * Work-around for bug 2619
487      * Set DisableVlan field in TP_RSS_CONFIG_VRT register so that the
488      * VLAN tag extraction is disabled.
489      */
490     t4_set_reg_field(sc, A_TP_RSS_CONFIG_VRT, F_DISABLEVLAN, F_DISABLEVLAN);
491
492     /* Store filter mode */
493     t4_read_indirect(sc, A_TP_PIO_ADDR, A_TP_PIO_DATA, &sc->filter_mode, 1,
494                      A_TP_VLAN_PRI_MAP);
495
496     /*
497      * First pass over all the ports - allocate VIs and initialize some
498      * basic parameters like mac address, port type, etc. We also figure
499      * out whether a port is 10G or 1G and use that information when
500      * calculating how many interrupts to attempt to allocate.
501      */
502     n100g = n40g = n25g = n10g = nlg = 0;
503     for_each_port(sc, i) {
504         struct port_info *pi;
505
506         pi = kmalloc(sizeof(*pi), KM_SLEEP);
507         sc->port[i] = pi;
508
509         /* These must be set before t4_port_init */
510         pi->adapter = sc;
511         /* LINTED: E_ASSIGN_NARROW_CONV */
512         pi->port_id = i;
513     }
514
515     /* Allocate the vi and initialize parameters like mac addr */
516     rc = -t4_port_init(sc, sc->mbox, sc->pf, 0);
517     if (rc) {
518         cxgb_printf(dip, CE_WARN,
519                     "unable to initialize port: %d", rc);
520         goto done;
521     }
522
523     for_each_port(sc, i) {
524         struct port_info *pi = sc->port[i];
525
526         mutex_init(&pi->lock, NULL, MUTEX_DRIVER, NULL);
527         pi->mtu = ETHERMTU;
528
529         if (is_100G_port(pi)) {
530             nl00g++;
531             pi->tmr_idx = prp->tmr_idx_10g;
532     }
```

```

532         pi->pktc_idx = prp->pktc_idx_10g;
533     } else if (is_40G_port(pi)) {
534         n40g++;
535         pi->tmr_idx = prp->tmr_idx_10g;
536         pi->pktc_idx = prp->pktc_idx_10g;
537     } else if (is_25G_port(pi)) {
538         n25g++;
539         pi->tmr_idx = prp->tmr_idx_10g;
540         pi->pktc_idx = prp->pktc_idx_10g;
541     } else if (is_10G_port(pi)) {
542         n10g++;
543         pi->tmr_idx = prp->tmr_idx_10g;
544         pi->pktc_idx = prp->pktc_idx_10g;
545     } else {
546         nlg++;
547         pi->tmr_idx = prp->tmr_idx_lg;
548         pi->pktc_idx = prp->pktc_idx_lg;
549     }

550     pi->xact_addr_filt = -1;
551     t4_mc_init(pi);
552
553     setbit(&sc->registered_device_map, i);
554 }
555
556 nxg = n10g + n25g + n40g + n100g;
557 (void) remove_extra_props(sc, nxg, nlg);
558
559 if (sc->registered_device_map == 0) {
560     cxgb_printf(dip, CE_WARN, "no usable ports");
561     rc = DDI_FAILURE;
562     goto done;
563 }
564
565 rc = cfg_itype_and_nqueues(sc, nxg, nlg, &iaq);
566 if (rc != 0)
567     goto done; /* error message displayed already */
568
569 sc->intr_type = iaq.intr_type;
570 sc->intr_count = iaq.nirq;
571
572 if (sc->props.multi_rings && (sc->intr_type != DDI_INTR_TYPE_MSIX)) {
573     sc->props.multi_rings = 0;
574     cxgb_printf(dip, CE_WARN,
575                 "Multiple rings disabled as interrupt type is not MSI-X");
576 }
577
578 if (sc->props.multi_rings && iaq.intr_fwd) {
579     sc->props.multi_rings = 0;
580     cxgb_printf(dip, CE_WARN,
581                 "Multiple rings disabled as interrupts are forwarded");
582 }
583
584 if (!sc->props.multi_rings) {
585     iaq.ntxq10g = 1;
586     iaq.ntxqlg = 1;
587 }
588
589 s = &sc->sge;
590 s->nrxq = nxg * iaq.nrxq10g + nlg * iaq.nrxqlg;
591 s->ntxq = nxg * iaq.ntxq10g + nlg * iaq.ntxqlg;
592 s->neq = s->ntxq + s->nrxq; /* the fl in an rxq is an eq */
593 #ifdef TCP_OFFLOAD_ENABLE
594     /* control queues, 1 per port + 1 mgmtq */
595     s->neq += sc->params.nports + 1;
596 #endif
597 s->niq = s->nrxq + 1; /* 1 extra for firmware event queue */

```

```

598     if (iaq.intr_fwd != 0)
599         sc->flags |= INTR_FWD;
600 #ifdef TCP_OFFLOAD_ENABLE
601     if (is_offload(sc) != 0) {
602
603         s->noflrxq = nxg * iaq.noflrxq10g + nlg * iaq.noflrxqlg;
604         s->nofltxq = nxg * iaq.nofltxq10g + nlg * iaq.nofltxqlg;
605         s->neq += s->nofltxq + s->noflrxq;
606         s->niq += s->noflrxq;
607
608         s->ofld_rxq = kmem_zalloc(s->noflrxq *
609                                     sizeof (struct sge_ofld_rxq), KM_SLEEP);
610         s->ofld_txq = kmem_zalloc(s->nofltxq *
611                                     sizeof (struct sge_wrq), KM_SLEEP);
612         s->ctrlq = kmem_zalloc(sc->params.nports *
613                               sizeof (struct sge_wrq), KM_SLEEP);
614
615     }
616 #endif
617     s->rnxq = kmem_zalloc(s->nrxq * sizeof (struct sge_rxq), KM_SLEEP);
618     s->ntxq = kmem_zalloc(s->ntxq * sizeof (struct sge_txq), KM_SLEEP);
619     s->iqmap = kmem_zalloc(s->niq * sizeof (struct sge_iq *), KM_SLEEP);
620     s->eqmap = kmem_zalloc(s->neq * sizeof (struct sge_eq *), KM_SLEEP);
621
622     sc->intr_handle = kmem_zalloc(sc->intr_count *
623                                    sizeof (ddi_intr_handle_t), KM_SLEEP);
624
625     /*
626      * Second pass over the ports. This time we know the number of rx and
627      * tx queues that each port should get.
628      */
629     rqidx = tqidx = 0;
630 #ifdef TCP_OFFLOAD_ENABLE
631     ofld_rqidx = ofld_tqidx = 0;
632 #endif
633     for_each_port(sc, i) {
634         struct port_info *pi = sc->port[i];
635
636         if (pi == NULL)
637             continue;
638
639         t4_mc_cb_init(pi);
640         /* LINTED: E_ASSIGN_NARROW_CONV */
641         pi->first_rxq = rqidx;
642         /* LINTED: E_ASSIGN_NARROW_CONV */
643         pi->nrxq = (is_10XG_port(pi)) ? iaq.nrxq10g
644                                         : iaq.nrxqlg;
645         /* LINTED: E_ASSIGN_NARROW_CONV */
646         pi->first_txq = tqidx;
647         /* LINTED: E_ASSIGN_NARROW_CONV */
648         pi->ntxq = (is_10XG_port(pi)) ? iaq.ntxq10g
649                                         : iaq.ntxqlg;
650
651         rqidx += pi->nrxq;
652         tqidx += pi->ntxq;
653
654 #ifdef TCP_OFFLOAD_ENABLE
655     if (is_offload(sc) != 0) {
656         /* LINTED: E_ASSIGN_NARROW_CONV */
657         pi->first_ofld_rxq = ofld_rqidx;
658         pi->noflrxq = max(1, pi->nrxq / 4);
659
660         /* LINTED: E_ASSIGN_NARROW_CONV */
661         pi->first_ofld_txq = ofld_tqidx;
662         pi->nofltxq = max(1, pi->ntxq / 2);
663     }
664 #endif

```

```

664         ofld_rqidx += pi->noflrxq;
665         ofld_tqidx += pi->nofltxq;
666     }
667 #endif
668     /*
669      * Enable hw checksumming and LSO for all ports by default.
670      * They can be disabled using ndd (hw_csum and hw_lso).
671      */
672     pi->features |= (CXGBE_HW_CSUM | CXGBE_HW_LSO);
673 }
674 }

675 #ifdef TCP_OFFLOAD_ENABLE
676     sc->l2t = t4_init_l2t(sc);
677 #endif

678 */

679     /*
680      * Setup Interrupts.
681      */
682     i = 0;
683     rc = ddi_intr_alloc(dip, sc->intr_handle, sc->intr_type, 0,
684                         sc->intr_count, &i, DDI_INTR_ALLOC_STRICT);
685     if (rc != DDI_SUCCESS) {
686         cxgb_printf(dip, CE_WARN,
687                     "failed to allocate %d interrupt(s) of type %d: %d, %d",
688                     sc->intr_count, sc->intr_type, rc, i);
689         goto done;
690     }
691     ASSERT(sc->intr_count == i); /* allocation was STRICT */
692     (void) ddi_intr_get_cap(sc->intr_handle[0], &sc->intr_cap);
693     (void) ddi_intr_get_pri(sc->intr_handle[0], &sc->intr_pri);
694     if (sc->intr_count == 1) {
695         ASSERT(sc->flags & INTR_FWD);
696         (void) ddi_intr_add_handler(sc->intr_handle[0], t4_intr_all, sc,
697                                     &ss->fwq);
698     } else {
699         /* Multiple interrupts. The first one is always error intr */
700         (void) ddi_intr_add_handler(sc->intr_handle[0], t4_intr_err, sc,
701                                     NULL);
702         irq++;
703
704         /* The second one is always the firmware event queue */
705         (void) ddi_intr_add_handler(sc->intr_handle[1], t4_intr, sc,
706                                     &ss->fwq);
707         irq++;
708
709         /*
710          * Note that if INTR_FWD is set then either the NIC rx
711          * queues or (exclusive or) the TOE rx queueus will be taking
712          * direct interrupts.
713          *
714          * There is no need to check for is_offload(sc) as nofldrxq
715          * will be 0 if offload is disabled.
716          */
717         for_each_port(sc, i) {
718             struct port_info *pi = sc->port[i];
719             struct sge_rxq *rxq;
720
721 #ifdef TCP_OFFLOAD_ENABLE
722             struct sge_ofld_rxq *ofld_rxq;
723
724             /*
725              * Skip over the NIC queues if they aren't taking direct
726              * interrupts.
727              */
728             if ((sc->flags & INTR_FWD) &&
729                 pi->noflrxq > pi->nrxq)

```

```

730         goto ofld_queues;
731     }
732     /*
733      * for (q = 0; q < pi->nrxq; q++, rxq++) {
734      *   (void) ddi_intr_add_handler(
735      *     sc->intr_handle[irq], t4_intr, sc,
736      *     &rxq->iq);
737      *   irq++;
738     }
739
740 #ifdef TCP_OFFLOAD_ENABLE
741     /*
742      * Skip over the offload queues if they aren't taking
743      * direct interrupts.
744      */
745     if ((sc->flags & INTR_FWD))
746         continue;
747     ofld_queues:
748     ofld_rxq = &s->ofld_rxq[pi->first_ofld_rxq];
749     for (q = 0; q < pi->noflrxq; q++, ofld_rxq++) {
750         (void) ddi_intr_add_handler(
751             sc->intr_handle[irq], t4_intr, sc,
752             &ofld_rxq->iq);
753         irq++;
754     }
755 #endif
756 }
757
758 }
759     sc->flags |= INTR_ALLOCATED;
760
761 ASSERT(rc == DDI_SUCCESS);
762 ddi_report_dev(dip);
763
764 /*
765  * Hardware/Firmware/etc. Version/Revision IDs.
766  */
767 t4_dump_version_info(sc);
768
769 if (n100g) {
770     cxgb_printf(dip, CE_NOTE,
771                 "%dx100G (%d rxq, %d txq total) %d %s.",
772                 n100g, rqidx, tqidx, sc->intr_count,
773                 sc->intr_type == DDI_INTR_TYPE_MSIX ? "MSI-X interrupts" :
774                 sc->intr_type == DDI_INTR_TYPE_MSI ? "MSI interrupts" :
775                 "fixed interrupt");
776 } else if (n40g) {
777     cxgb_printf(dip, CE_NOTE,
778                 "%dx40G (%d rxq, %d txq total) %d %s.",
779                 n40g, rqidx, tqidx, sc->intr_count,
780                 sc->intr_type == DDI_INTR_TYPE_MSIX ? "MSI-X interrupts" :
781                 sc->intr_type == DDI_INTR_TYPE_MSI ? "MSI interrupts" :
782                 "fixed interrupt");
783 } else if (n25g) {
784     cxgb_printf(dip, CE_NOTE,
785                 "%dx25G (%d rxq, %d txq total) %d %s.",
786                 n25g, rqidx, tqidx, sc->intr_count,
787                 sc->intr_type == DDI_INTR_TYPE_MSIX ? "MSI-X interrupts" :
788                 sc->intr_type == DDI_INTR_TYPE_MSI ? "MSI interrupts" :
789                 "fixed interrupt");
790 } else if (n10g && nl1g) {
791     cxgb_printf(dip, CE_NOTE,
792                 "%dx10G %dx1G (%d rxq, %d txq total) %d %s.",
793                 n10g, nl1g, rqidx, tqidx, sc->intr_count,
794                 sc->intr_type == DDI_INTR_TYPE_MSIX ? "MSI-X interrupts" :
795                 sc->intr_type == DDI_INTR_TYPE_MSI ? "MSI interrupts" :

```

```
796         "fixed interrupt");
797     } else {
798         cxgb_printf(dip, CE_NOTE,
799                     "%dx%sg (%d rxq, %d txq per port) %d %s.",
800                     n10g ? n10g : nlg,
801                     n10g ? "10" : "1",
802                     n10g ? iaq.nrxql0g : iaq.nrxqlg,
803                     n10g ? iaq.ntxql0g : iaq.ntxqlg,
804                     sc->intr_count,
805                     sc->intr_type == DDI_INTR_TYPE_MSIX ? "MSI-X interrupts" :
806                     sc->intr_type == DDI_INTR_TYPE_MSI ? "MSI interrupts" :
807                     "fixed interrupt");
808     }
810     sc->ksp = setup_kstats(sc);
811     sc->ksp_stat = setup_wc_kstats(sc);
812     sc->params.drv_memwin = MEMWIN_NIC;
814 done:
815     if (rc != DDI_SUCCESS) {
816         (void) t4_devo_detach(dip, DDI_DETACH);
818         /* rc may have errno style errors or DDI errors */
819         rc = DDI_FAILURE;
820     }
822 }
823 }
```

unchanged_portion_omitted_

```
*****
96753 Tue Nov 20 20:17:43 2018
new/usr/src/uts/common/io/cxgbe/t4nex/t4_sge.c
9994 cxgbe t4nex: Handle get_fl_payload() alloc failures
9995 cxgbe t4_devo_attach() should initialize ->sfl
*****
unchanged_portion_omitted

728 /*
729 * t4_ring_rx - Process responses from an SGE response queue.
730 *
731 * This function processes responses from an SGE response queue up to the supplied
732 * Responses include received packets as well as control messages from FW
733 * or HW.
734 * It returns a chain of mblk's containing the received data, to be
735 * passed up to mac_ring_rx().
736 */
737 mblk_t *
738 t4_ring_rx(struct sge_rxq *rxq, int budget)
739 {
740     struct sge_iq *iq = &rxq->iq;
741     struct sge_fl *fl = &rxq->fl; /* Use iff IQ_HAS_FL */
742     struct adapter *sc = iq->adapter;
743     struct rsp_ctrl *ctrl;
744     const struct rss_header *rss;
745     int ndescs = 0, fl_bufs_used = 0;
746     int rsp_type;
747     uint32_t lq;
748     mblk_t *mblk_head = NULL, **mblk_tail, *m;
749     struct cpl_rx_pkt *cpl;
750     uint32_t received_bytes = 0, pkt_len = 0;
751     bool csum_ok;
752     uint16_t err_vec;
753
754     mblk_tail = &mblk_head;
755
756     while (is_new_response(iq, &ctrl)) {
757         membar_consumer();
758
759         m = NULL;
760         rsp_type = G_RSPD_TYPE(ctrl->u.type_gen);
761         lq = be32_to_cpu(ctrl->pldbuflen_qid);
762         rss = (const void *)iq->cdesc;
763
764         switch (rsp_type) {
765             case X_RSPD_TYPE_FLBUFF:
766                 ASSERT(iq->flags & IQ_HAS_FL);
767
768                 if (CPL_RX_PKT == rss->opcode) {
769                     cpl = (void *)(rss + 1);
770                     pkt_len = be16_to_cpu(cpl->len);
771
772                     if (iq->polling && ((received_bytes + pkt_len) >
773                         goto done;
774
775                     m = get_fl_payload(sc, fl, lq, &fl_bufs_used);
776                     if (m == NULL)
777                         goto done;
778                     if (m == NULL) {
779                         panic("%s: line %d.", __func__,
780                               __LINE__);
781                 }
782
783         iq->intr_next = iq->intr_params;
```

```
782     m->b_rptr += sc->sge.pktshift;
783     if (sc->params.tp.rx_pkt_encap)
784         /* It is enabled only in T6 config file */
785         err_vec = G_T6_COMPRESS_RXERR_VEC(ntohs(cpl));
786     else
787         err_vec = ntohs(cpl->err_vec);
788
789     csum_ok = cpl->csum_calc && !err_vec;
790
791     /* TODO: what about cpl->ip_frag? */
792     if (csum_ok && !cpl->ip_frag) {
793         mac_hcksum_set(m, 0, 0, 0, 0xffff,
794                         HCK_FULLCKSUM_OK | HCK_FULLCKSUM |
795                         HCK_IPV4_HDRCKSUM_OK);
796         rxq->rxcsum++;
797     }
798     rxq->rpxpkts++;
799     rxq->rxbytes += pkt_len;
800     received_bytes += pkt_len;
801
802     *mblk_tail = m;
803     mblk_tail = &m->b_next;
804
805     break;
806 }
807
808 m = get_fl_payload(sc, fl, lq, &fl_bufs_used);
809 if (m == NULL)
810     goto done;
811 if (m == NULL) {
812     panic("%s: line %d.", __func__,
813           __LINE__);
814 }
815 /* FALLTHROUGH */
816
817 case X_RSPD_TYPE_CPL:
818     ASSERT(rss->opcode < NUM_CPL_CMDS);
819     sc->cpl_handler[rss->opcode](iq, rss, m);
820     break;
821
822 default:
823     break;
824 }
825
826 iq_next(iq);
827 ++ndescs;
828 if (!iq->polling && (ndescs == budget))
829     break;
830
831 done:
832
833     t4_write_reg(sc, MYPF_REG(A_SGE_PF_GTS),
834                  V_CIDXINC(ndescs) | V_INGRESSQID(iq->cntxt_id) |
835                  V_SEINTARM(V_QINTR_TIMER_IDX(X_TIMERREG_UPDATE_CIDX)));
836
837     if ((fl_bufs_used > 0) || (iq->flags & IQ_HAS_FL)) {
838         int starved;
839         FL_LOCK(fl);
840         fl->needed += fl_bufs_used;
841         starved = refill_fl(sc, fl, fl->cap / 8);
842         FL_UNLOCK(fl);
843         if (starved)
844             add_fl_to_sfl(sc, fl);
845     }
846
847 return (mblk_head);
```

```

845 /*
846 * Deals with anything and everything on the given ingress queue.
847 */
848 static int
849 service_iq(struct sge_iq *iq, int budget)
850 {
851     struct sge_iq *q;
852     struct sge_rxq *rxq = iq_to_rxq(iq); /* Use iff iq is part of rxq */
853     struct sge_fl *fl = &rxq->fl; /* Use iff IQ_HAS_FL */
854     struct adapter *sc = iq->adapter;
855     struct rsp_ctrl *ctrl;
856     const struct rss_header *rss;
857     int ndescs = 0, limit, fl_bufs_used = 0;
858     int rsp_type;
859     uint32_t lq;
860     int starved;
861     mblk_t *m;
862     STAILQ_HEAD(, sge_iq) iql = STAILQ_HEAD_INITIALIZER(iql);
863
864     limit = budget ? budget : iq->qsize / 8;
865
866     /*
867      * We always come back and check the descriptor ring for new indirect
868      * interrupts and other responses after running a single handler.
869      */
870     for (;;) {
871         while (is_new_response(iq, &ctrl)) {
872
873             membar_consumer();
874
875             m = NULL;
876             rsp_type = G_RSPD_TYPE(ctrl->u.type_gen);
877             lq = be32_to_cpu(ctrl->pldbuflen_qid);
878             rss = (const void *)iq->cdesc;
879
880             switch (rsp_type) {
881                 case X_RSPD_TYPE_FLBUF:
882
883                     ASSERT(iq->flags & IQ_HAS_FL);
884
885                     m = get_fl_payload(sc, fl, lq, &fl_bufs_used);
886                     if (m == NULL) {
887                         /*
888                          * Rearm the iq with a
889                          * longer-than-default timer
890                          */
891                         t4_write_reg(sc, MYPF_REG(A_SGE_PF_GTS),
892                                     V_INGRESSQID((u32)iq->cn
893                                     V_SEINTARM(V_QINTR_TIMER));
894
895                     if (fl_bufs_used > 0) {
896                         ASSERT(iq->flags & IQ_HAS_FL);
897                         FL_LOCK(fl);
898                         fl->needed += fl_bufs_used;
899                         starved = refill_fl(sc, fl, fl-+
900                                         FL_UNLOCK(fl));
901                         if (starved)
902                             add_fl_to_sfl(sc, fl);
903                         panic("%s: line %d.", __func__,
904                               __LINE__);
905                     }
906                     return (0);
907
908                 /* FALLTHRU */
909                 case X_RSPD_TYPE_CPL:
910             }

```

```

909     ASSERT(rss->opcode < NUM_CPL_CMDS);
910     sc->cpl_handler[rss->opcode](iq, rss, m);
911     break;
913
914 case X_RSPD_TYPE_INTR:
915
916     /*
917      * Interrupts should be forwarded only to queues
918      * that are not forwarding their interrupts.
919      * This means service_iq can recurse but only 1
920      * level deep.
921      */
922     ASSERT(budget == 0);
923
924     q = sc->sge.iqmap[lq - sc->sge.iq_start];
925     if (atomic_cas_uint(&q->state, IQS_IDLE,
926                         IQS_BUSY) == IQS_IDLE) {
927         if (service_iq(q, q->qsize / 8) == 0) {
928             (void) atomic_cas_uint(
929                             &q->state, IQS_BUSY,
930                             IQS_IDLE);
931         } else {
932             STAILQ_INSERT_TAIL(&iql, q,
933                                 link);
934         }
935     }
936     break;
937
938 default:
939     break;
940 }
941
942 iq_next(iq);
943 if (++ndescs == limit) {
944     t4_write_reg(sc, MYPF_REG(A_SGE_PF_GTS),
945                  V_CIDXINC(ndescs) |
946                  V_INGRESSQID(iq->cntxt_id) |
947                  V_SEINTARM(V_QINTR_TIMER_IDX(
948                               X_TIMERREG_UPDATE_CIDX)));
949     ndescs = 0;
950
951     if (fl_bufs_used > 0) {
952         ASSERT(iq->flags & IQ_HAS_FL);
953         FL_LOCK(f1);
954         fl->needed += fl_bufs_used;
955         (void) refill_fl(sc, fl, fl->cap / 8);
956         FL_UNLOCK(f1);
957         fl_bufs_used = 0;
958     }
959
960     if (budget != 0)
961         return (EINPROGRESS);
962 }
963
964 if (STAILQ_EMPTY(&iql) != 0)
965     break;
966
967 /*
968  * Process the head only, and send it to the back of the list if
969  * it's still not done.
970  */
971 q = STAILQ_FIRST(&iql);
972 STAILQ_REMOVE_HEAD(&iql, link);
973 if (service_iq(q, q->qsize / 8) == 0)

```

```

974         (void) atomic_cas_uint(&q->state, IQS_BUSY, IQS_IDLE);
975     else
976         STAILQ_INSERT_TAIL(&iql, q, link);
977 }
978
979 t4_write_reg(sc, MYPF_REG(A_SGE_PF_GTS), V_CIDXINC(ndescs) |
980 V_INGRESSQID((u32)iq->cntxt_id) | V_SEINTARM(iq->intr_next));
981
982 if (iq->flags & IQ_HAS_FL) {
983     int starved;
984
985     FL_LOCK(f1);
986     fl->needed += fl_bufs_used;
987     starved = refill_fl(sc, f1, fl->cap / 4);
988     FL_UNLOCK(f1);
989     if (starved != 0)
990         add_f1_to_sfl(sc, f1);
991 }
992
993 } unchanged portion omitted
1262 static inline void
1263 init_f1(struct sge_f1 *f1, uint16_t qsize)
1264 {
1265     f1->qsize = qsize;
1266     f1->allocb_fail = 0;
1267 } unchanged portion omitted
1268
1269 /* Note that f1->cidx and f1->offset are left unchanged in case of failure.
1270 */
1271
1272 static mblk_t *
1273 get_f1_payload(struct adapter *sc, struct sge_f1 *f1,
1274                 uint32_t len_newbuf, int *fl_bufs_used)
1275 {
1276     struct mblk_pair frame = {0};
1277     struct rxbuf *rxb;
1278     mblk_t *m = NULL;
1279     uint_t nbuf = 0, len, copy, n;
1280     uint32_t cidx, offset, rcidx, roffset;
1281     uint32_t cidx, offset;
1282
1283     /*
1284      * The SGE won't pack a new frame into the current buffer if the entire
1285      * payload doesn't fit in the remaining space. Move on to the next buf
1286      * in that case.
1287      */
1288     rcidx = f1->cidx;
1289     roffset = f1->offset;
1290     if (f1->offset > 0 && len_newbuf & F_RSPD_NEWSUF) {
1291         f1->offset = 0;
1292         if (++f1->cidx == f1->cap)
1293             f1->cidx = 0;
1294         nbuf++;
1295     }
1296     cidx = f1->cidx;
1297     offset = f1->offset;
1298
1299     len = G_RSPD_LEN(len_newbuf); /* pktshift + payload length */
1300     copy = (len <= f1->copy_threshold);
1301     if (copy != 0) {
1302         frame.head = m = allocb(len, BPRI_HI);
1303
1304         fl->cidx = rcidx;
1305         fl->offset = roffset;
1306
1307         fl->needed += fl_bufs_used;
1308         starved = refill_f1(sc, f1, fl->cap / 4);
1309         if (starved != 0)
1310             add_f1_to_sfl(sc, f1);
1311
1312         if (copy != 0)
1313             add_f1_to_sfl(sc, f1);
1314
1315         if (copy != 0)
1316             add_f1_to_sfl(sc, f1);
1317
1318         if (copy != 0)
1319             add_f1_to_sfl(sc, f1);
1320
1321         if (copy != 0)
1322             add_f1_to_sfl(sc, f1);
1323
1324         if (copy != 0)
1325             add_f1_to_sfl(sc, f1);
1326
1327         if (copy != 0)
1328             add_f1_to_sfl(sc, f1);
1329
1330         if (copy != 0)
1331             add_f1_to_sfl(sc, f1);
1332
1333         if (copy != 0)
1334             add_f1_to_sfl(sc, f1);
1335
1336         if (copy != 0)
1337             add_f1_to_sfl(sc, f1);
1338
1339         if (copy != 0)
1340             add_f1_to_sfl(sc, f1);
1341
1342         if (copy != 0)
1343             add_f1_to_sfl(sc, f1);
1344
1345         if (copy != 0)
1346             add_f1_to_sfl(sc, f1);
1347
1348         if (copy != 0)
1349             add_f1_to_sfl(sc, f1);
1350
1351         if (copy != 0)
1352             add_f1_to_sfl(sc, f1);
1353
1354         if (copy != 0)
1355             add_f1_to_sfl(sc, f1);
1356
1357         if (copy != 0)
1358             add_f1_to_sfl(sc, f1);
1359
1360         if (copy != 0)
1361             add_f1_to_sfl(sc, f1);
1362
1363         if (copy != 0)
1364             add_f1_to_sfl(sc, f1);
1365
1366         if (copy != 0)
1367             add_f1_to_sfl(sc, f1);
1368
1369         if (copy != 0)
1370             add_f1_to_sfl(sc, f1);
1371
1372         if (copy != 0)
1373             add_f1_to_sfl(sc, f1);
1374
1375         if (copy != 0)
1376             add_f1_to_sfl(sc, f1);
1377
1378         if (copy != 0)
1379             add_f1_to_sfl(sc, f1);
1380
1381         if (copy != 0)
1382             add_f1_to_sfl(sc, f1);
1383
1384         if (copy != 0)
1385             add_f1_to_sfl(sc, f1);
1386
1387         if (copy != 0)
1388             add_f1_to_sfl(sc, f1);
1389
1390         if (copy != 0)
1391             add_f1_to_sfl(sc, f1);
1392
1393         if (copy != 0)
1394             add_f1_to_sfl(sc, f1);
1395
1396         if (copy != 0)
1397             add_f1_to_sfl(sc, f1);
1398
1399         if (copy != 0)
1400             add_f1_to_sfl(sc, f1);
1401
1402         if (copy != 0)
1403             add_f1_to_sfl(sc, f1);
1404
1405         if (copy != 0)
1406             add_f1_to_sfl(sc, f1);
1407
1408         if (copy != 0)
1409             add_f1_to_sfl(sc, f1);
1410
1411         if (copy != 0)
1412             add_f1_to_sfl(sc, f1);
1413
1414         if (copy != 0)
1415             add_f1_to_sfl(sc, f1);
1416
1417         if (copy != 0)
1418             add_f1_to_sfl(sc, f1);
1419
1420         if (copy != 0)
1421             add_f1_to_sfl(sc, f1);
1422
1423         if (copy != 0)
1424             add_f1_to_sfl(sc, f1);
1425
1426         if (copy != 0)
1427             add_f1_to_sfl(sc, f1);
1428
1429         if (copy != 0)
1430             add_f1_to_sfl(sc, f1);
1431
1432         if (copy != 0)
1433             add_f1_to_sfl(sc, f1);
1434
1435         if (copy != 0)
1436             add_f1_to_sfl(sc, f1);
1437
1438         if (copy != 0)
1439             add_f1_to_sfl(sc, f1);
1440
1441         if (copy != 0)
1442             add_f1_to_sfl(sc, f1);
1443
1444         if (copy != 0)
1445             add_f1_to_sfl(sc, f1);
1446
1447         if (copy != 0)
1448             add_f1_to_sfl(sc, f1);
1449
1450         if (copy != 0)
1451             add_f1_to_sfl(sc, f1);
1452
1453         if (copy != 0)
1454             add_f1_to_sfl(sc, f1);
1455
1456         if (copy != 0)
1457             add_f1_to_sfl(sc, f1);
1458
1459         if (copy != 0)
1460             add_f1_to_sfl(sc, f1);
1461
1462         if (copy != 0)
1463             add_f1_to_sfl(sc, f1);
1464
1465         if (copy != 0)
1466             add_f1_to_sfl(sc, f1);
1467
1468         if (copy != 0)
1469             add_f1_to_sfl(sc, f1);
1470
1471         if (copy != 0)
1472             add_f1_to_sfl(sc, f1);
1473
1474         if (copy != 0)
1475             add_f1_to_sfl(sc, f1);
1476
1477         if (copy != 0)
1478             add_f1_to_sfl(sc, f1);
1479
1480         if (copy != 0)
1481             add_f1_to_sfl(sc, f1);
1482
1483         if (copy != 0)
1484             add_f1_to_sfl(sc, f1);
1485
1486         if (copy != 0)
1487             add_f1_to_sfl(sc, f1);
1488
1489         if (copy != 0)
1490             add_f1_to_sfl(sc, f1);
1491
1492         if (copy != 0)
1493             add_f1_to_sfl(sc, f1);
1494
1495         if (copy != 0)
1496             add_f1_to_sfl(sc, f1);
1497
1498         if (copy != 0)
1499             add_f1_to_sfl(sc, f1);
1500
1501         if (copy != 0)
1502             add_f1_to_sfl(sc, f1);
1503
1504         if (copy != 0)
1505             add_f1_to_sfl(sc, f1);
1506
1507         if (copy != 0)
1508             add_f1_to_sfl(sc, f1);
1509
1510         if (copy != 0)
1511             add_f1_to_sfl(sc, f1);
1512
1513         if (copy != 0)
1514             add_f1_to_sfl(sc, f1);
1515
1516         if (copy != 0)
1517             add_f1_to_sfl(sc, f1);
1518
1519         if (copy != 0)
1520             add_f1_to_sfl(sc, f1);
1521
1522         if (copy != 0)
1523             add_f1_to_sfl(sc, f1);
1524
1525         if (copy != 0)
1526             add_f1_to_sfl(sc, f1);
1527
1528         if (copy != 0)
1529             add_f1_to_sfl(sc, f1);
1530
1531         if (copy != 0)
1532             add_f1_to_sfl(sc, f1);
1533
1534         if (copy != 0)
1535             add_f1_to_sfl(sc, f1);
1536
1537         if (copy != 0)
1538             add_f1_to_sfl(sc, f1);
1539
1540         if (copy != 0)
1541             add_f1_to_sfl(sc, f1);
1542
1543         if (copy != 0)
1544             add_f1_to_sfl(sc, f1);
1545
1546         if (copy != 0)
1547             add_f1_to_sfl(sc, f1);
1548
1549         if (copy != 0)
1550             add_f1_to_sfl(sc, f1);
1551
1552         if (copy != 0)
1553             add_f1_to_sfl(sc, f1);
1554
1555         if (copy != 0)
1556             add_f1_to_sfl(sc, f1);
1557
1558         if (copy != 0)
1559             add_f1_to_sfl(sc, f1);
1560
1561         if (copy != 0)
1562             add_f1_to_sfl(sc, f1);
1563
1564         if (copy != 0)
1565             add_f1_to_sfl(sc, f1);
1566
1567         if (copy != 0)
1568             add_f1_to_sfl(sc, f1);
1569
1570         if (copy != 0)
1571             add_f1_to_sfl(sc, f1);
1572
1573         if (copy != 0)
1574             add_f1_to_sfl(sc, f1);
1575
1576         if (copy != 0)
1577             add_f1_to_sfl(sc, f1);
1578
1579         if (copy != 0)
1580             add_f1_to_sfl(sc, f1);
1581
1582         if (copy != 0)
1583             add_f1_to_sfl(sc, f1);
1584
1585         if (copy != 0)
1586             add_f1_to_sfl(sc, f1);
1587
1588         if (copy != 0)
1589             add_f1_to_sfl(sc, f1);
1590
1591         if (copy != 0)
1592             add_f1_to_sfl(sc, f1);
1593
1594         if (copy != 0)
1595             add_f1_to_sfl(sc, f1);
1596
1597         if (copy != 0)
1598             add_f1_to_sfl(sc, f1);
1599
1600         if (copy != 0)
1601             add_f1_to_sfl(sc, f1);
1602
1603         if (copy != 0)
1604             add_f1_to_sfl(sc, f1);
1605
1606         if (copy != 0)
1607             add_f1_to_sfl(sc, f1);
1608
1609         if (copy != 0)
1610             add_f1_to_sfl(sc, f1);
1611
1612         if (copy != 0)
1613             add_f1_to_sfl(sc, f1);
1614
1615         if (copy != 0)
1616             add_f1_to_sfl(sc, f1);
1617
1618         if (copy != 0)
1619             add_f1_to_sfl(sc, f1);
1620
1621         if (copy != 0)
1622             add_f1_to_sfl(sc, f1);
1623
1624         if (copy != 0)
1625             add_f1_to_sfl(sc, f1);
1626
1627         if (copy != 0)
1628             add_f1_to_sfl(sc, f1);
1629
1630         if (copy != 0)
1631             add_f1_to_sfl(sc, f1);
1632
1633         if (copy != 0)
1634             add_f1_to_sfl(sc, f1);
1635
1636         if (copy != 0)
1637             add_f1_to_sfl(sc, f1);
1638
1639         if (copy != 0)
1640             add_f1_to_sfl(sc, f1);
1641
1642         if (copy != 0)
1643             add_f1_to_sfl(sc, f1);
1644
1645         if (copy != 0)
1646             add_f1_to_sfl(sc, f1);
1647
1648         if (copy != 0)
1649             add_f1_to_sfl(sc, f1);
1650
1651         if (copy != 0)
1652             add_f1_to_sfl(sc, f1);
1653
1654         if (copy != 0)
1655             add_f1_to_sfl(sc, f1);
1656
1657         if (copy != 0)
1658             add_f1_to_sfl(sc, f1);
1659
1660         if (copy != 0)
1661             add_f1_to_sfl(sc, f1);
1662
1663         if (copy != 0)
1664             add_f1_to_sfl(sc, f1);
1665
1666         if (copy != 0)
1667             add_f1_to_sfl(sc, f1);
1668
1669         if (copy != 0)
1670             add_f1_to_sfl(sc, f1);
1671
1672         if (copy != 0)
1673             add_f1_to_sfl(sc, f1);
1674
1675         if (copy != 0)
1676             add_f1_to_sfl(sc, f1);
1677
1678         if (copy != 0)
1679             add_f1_to_sfl(sc, f1);
1680
1681         if (copy != 0)
1682             add_f1_to_sfl(sc, f1);
1683
1684         if (copy != 0)
1685             add_f1_to_sfl(sc, f1);
1686
1687         if (copy != 0)
1688             add_f1_to_sfl(sc, f1);
1689
1690         if (copy != 0)
1691             add_f1_to_sfl(sc, f1);
1692
1693         if (copy != 0)
1694             add_f1_to_sfl(sc, f1);
1695
1696         if (copy != 0)
1697             add_f1_to_sfl(sc, f1);
1698
1699         if (copy != 0)
1700             add_f1_to_sfl(sc, f1);
1701
1702         if (copy != 0)
1703             add_f1_to_sfl(sc, f1);
1704
1705         if (copy != 0)
1706             add_f1_to_sfl(sc, f1);
1707
1708         if (copy != 0)
1709             add_f1_to_sfl(sc, f1);
1710
1711         if (copy != 0)
1712             add_f1_to_sfl(sc, f1);
1713
1714         if (copy != 0)
1715             add_f1_to_sfl(sc, f1);
1716
1717         if (copy != 0)
1718             add_f1_to_sfl(sc, f1);
1719
1720         if (copy != 0)
1721             add_f1_to_sfl(sc, f1);
1722
1723         if (copy != 0)
1724             add_f1_to_sfl(sc, f1);
1725
1726         if (copy != 0)
1727             add_f1_to_sfl(sc, f1);
1728
1729         if (copy != 0)
1730             add_f1_to_sfl(sc, f1);
1731
1732         if (copy != 0)
1733             add_f1_to_sfl(sc, f1);
1734
1735         if (copy != 0)
1736             add_f1_to_sfl(sc, f1);
1737
1738         if (copy != 0)
1739             add_f1_to_sfl(sc, f1);
1740
1741         if (copy != 0)
1742             add_f1_to_sfl(sc, f1);
1743
1744         if (copy != 0)
1745             add_f1_to_sfl(sc, f1);
1746
1747         if (copy != 0)
1748             add_f1_to_sfl(sc, f1);
1749
1750         if (copy != 0)
1751             add_f1_to_sfl(sc, f1);
1752
1753         if (copy != 0)
1754             add_f1_to_sfl(sc, f1);
1755
1756         if (copy != 0)
1757             add_f1_to_sfl(sc, f1);
1758
1759         if (copy != 0)
1760             add_f1_to_sfl(sc, f1);
1761
1762         if (copy != 0)
1763             add_f1_to_sfl(sc, f1);
1764
1765         if (copy != 0)
1766             add_f1_to_sfl(sc, f1);
1767
1768         if (copy != 0)
1769             add_f1_to_sfl(sc, f1);
1770
1771         if (copy != 0)
1772             add_f1_to_sfl(sc, f1);
1773
1774         if (copy != 0)
1775             add_f1_to_sfl(sc, f1);
1776
1777         if (copy != 0)
1778             add_f1_to_sfl(sc, f1);
1779
1780         if (copy != 0)
1781             add_f1_to_sfl(sc, f1);
1782
1783         if (copy != 0)
1784             add_f1_to_sfl(sc, f1);
1785
1786         if (copy != 0)
1787             add_f1_to_sfl(sc, f1);
1788
1789         if (copy != 0)
1790             add_f1_to_sfl(sc, f1);
1791
1792         if (copy != 0)
1793             add_f1_to_sfl(sc, f1);
1794
1795         if (copy != 0)
1796             add_f1_to_sfl(sc, f1);
1797
1798         if (copy != 0)
1799             add_f1_to_sfl(sc, f1);
1800
1801         if (copy != 0)
1802             add_f1_to_sfl(sc, f1);
1803
1804         if (copy != 0)
1805             add_f1_to_sfl(sc, f1);
1806
1807         if (copy != 0)
1808             add_f1_to_sfl(sc, f1);
1809
1810         if (copy != 0)
1811             add_f1_to_sfl(sc, f1);
1812
1813         if (copy != 0)
1814             add_f1_to_sfl(sc, f1);
1815
1816         if (copy != 0)
1817             add_f1_to_sfl(sc, f1);
1818
1819         if (copy != 0)
1820             add_f1_to_sfl(sc, f1);
1821
1822         if (copy != 0)
1823             add_f1_to_sfl(sc, f1);
1824
1825         if (copy != 0)
1826             add_f1_to_sfl(sc, f1);
1827
1828         if (copy != 0)
1829             add_f1_to_sfl(sc, f1);
1830
1831         if (copy != 0)
1832             add_f1_to_sfl(sc, f1);
1833
1834         if (copy != 0)
1835             add_f1_to_sfl(sc, f1);
1836
1837         if (copy != 0)
1838             add_f1_to_sfl(sc, f1);
1839
1840         if (copy != 0)
1841             add_f1_to_sfl(sc, f1);
1842
1843         if (copy != 0)
1844             add_f1_to_sfl(sc, f1);
1845
1846         if (copy != 0)
1847             add_f1_to_sfl(sc, f1);
1848
1849         if (copy != 0)
1850             add_f1_to_sfl(sc, f1);
1851
1852         if (copy != 0)
1853             add_f1_to_sfl(sc, f1);
1854
1855         if (copy != 0)
1856             add_f1_to_sfl(sc, f1);
1857
1858         if (copy != 0)
1859             add_f1_to_sfl(sc, f1);
1860
1861         if (copy != 0)
1862             add_f1_to_sfl(sc, f1);
1863
1864         if (copy != 0)
1865             add_f1_to_sfl(sc, f1);
1866
1867         if (copy != 0)
1868             add_f1_to_sfl(sc, f1);
1869
1870         if (copy != 0)
1871             add_f1_to_sfl(sc, f1);
1872
1873         if (copy != 0)
1874             add_f1_to_sfl(sc, f1);
1875
1876         if (copy != 0)
1877             add_f1_to_sfl(sc, f1);
1878
1879         if (copy != 0)
1880             add_f1_to_sfl(sc, f1);
1881
1882         if (copy != 0)
1883             add_f1_to_sfl(sc, f1);
1884
1885         if (copy != 0)
1886             add_f1_to_sfl(sc, f1);
1887
1888         if (copy != 0)
1889             add_f1_to_sfl(sc, f1);
1890
1891         if (copy != 0)
1892             add_f1_to_sfl(sc, f1);
1893
1894         if (copy != 0)
1895             add_f1_to_sfl(sc, f1);
1896
1897         if (copy != 0)
1898             add_f1_to_sfl(sc, f1);
1899
1900         if (copy != 0)
1901             add_f1_to_sfl(sc, f1);
1902
1903         if (copy != 0)
1904             add_f1_to_sfl(sc, f1);
1905
1906         if (copy != 0)
1907             add_f1_to_sfl(sc, f1);
1908
1909         if (copy != 0)
1910             add_f1_to_sfl(sc, f1);
1911
1912         if (copy != 0)
1913             add_f1_to_sfl(sc, f1);
1914
1915         if (copy != 0)
1916             add_f1_to_sfl(sc, f1);
1917
1918         if (copy != 0)
1919             add_f1_to_sfl(sc, f1);
1920
1921         if (copy != 0)
1922             add_f1_to_sfl(sc, f1);
1923
1924         if (copy != 0)
1925             add_f1_to_sfl(sc, f1);
1926
1927         if (copy != 0)
1928             add_f1_to_sfl(sc, f1);
1929
1930         if (copy != 0)
1931             add_f1_to_sfl(sc, f1);
1932
1933         if (copy != 0)
1934             add_f1_to_sfl(sc, f1);
1935
1936         if (copy != 0)
1937             add_f1_to_sfl(sc, f1);
1938
1939         if (copy != 0)
1940             add_f1_to_sfl(sc, f1);
1941
1942         if (copy != 0)
1943             add_f1_to_sfl(sc, f1);
1944
1945         if (copy != 0)
1946             add_f1_to_sfl(sc, f1);
1947
1948         if (copy != 0)
1949             add_f1_to_sfl(sc, f1);
1950
1951         if (copy != 0)
1952             add_f1_to_sfl(sc, f1);
1953
1954         if (copy != 0)
1955             add_f1_to_sfl(sc, f1);
1956
1957         if (copy != 0)
1958             add_f1_to_sfl(sc, f1);
1959
1960         if (copy != 0)
1961             add_f1_to_sfl(sc, f1);
1962
1963         if (copy != 0)
1964             add_f1_to_sfl(sc, f1);
1965
1966         if (copy != 0)
1967             add_f1_to_sfl(sc, f1);
1968
1969         if (copy != 0)
1970             add_f1_to_sfl(sc, f1);
1971
1972         if (copy != 0)
1973             add_f1_to_sfl(sc, f1);
1974
1975         if (copy != 0)
1976             add_f1_to_sfl(sc, f1);
1977
1978         if (copy != 0)
1979             add_f1_to_sfl(sc, f1);
1980
1981         if (copy != 0)
1982             add_f1_to_sfl(sc, f1);
1983
1984         if (copy != 0)
1985             add_f1_to_sfl(sc, f1);
1986
1987         if (copy != 0)
1988             add_f1_to_sfl(sc, f1);
1989
1990         if (copy != 0)
1991             add_f1_to_sfl(sc, f1);
1992
1993         if (copy != 0)
1994             add_f1_to_sfl(sc, f1);
1995
1996         if (copy != 0)
1997             add_f1_to_sfl(sc, f1);
1998
1999         if (copy != 0)
2000             add_f1_to_sfl(sc, f1);
2001
2002         if (copy != 0)
2003             add_f1_to_sfl(sc, f1);
2004
2005         if (copy != 0)
2006             add_f1_to_sfl(sc, f1);
2007
2008         if (copy != 0)
2009             add_f1_to_sfl(sc, f1);
2010
2011         if (copy != 0)
2012             add_f1_to_sfl(sc, f1);
2013
2014         if (copy != 0)
2015             add_f1_to_sfl(sc, f1);
2016
2017         if (copy != 0)
2018             add_f1_to_sfl(sc, f1);
2019
2020         if (copy != 0)
2021             add_f1_to_sfl(sc, f1);
2022
2023         if (copy != 0)
2024             add_f1_to_sfl(sc, f1);
2025
2026         if (copy != 0)
2027             add_f1_to_sfl(sc, f1);
2028
2029         if (copy != 0)
2030             add_f1_to_sfl(sc, f1);
2031
2032         if (copy != 0)
2033             add_f1_to_sfl(sc, f1);
2034
2035         if (copy != 0)
2036             add_f1_to_sfl(sc, f1);
2037
2038         if (copy != 0)
2039             add_f1_to_sfl(sc, f1);
2040
2041         if (copy != 0)
2042             add_f1_to_sfl(sc, f1);
2043
2044         if (copy != 0)
2045             add_f1_to_sfl(sc, f1);
2046
2047         if (copy != 0)
2048             add_f1_to_sfl(sc, f1);
2049
2050         if (copy != 0)
2051             add_f1_to_sfl(sc, f1);
2052
2053         if (copy != 0)
2054             add_f1_to_sfl(sc, f1);
2055
2056         if (copy != 0)
2057             add_f1_to_sfl(sc, f1);
2058
2059         if (copy != 0)
2060             add_f1_to_sfl(sc, f1);
2061
2062         if (copy != 0)
2063             add_f1_to_sfl(sc, f1);
2064
2065         if (copy != 0)
2066             add_f1_to_sfl(sc, f1);
2067
2068         if (copy != 0)
2069             add_f1_to_sfl(sc, f1);
2070
2071         if (copy != 0)
2072             add_f1_to_sfl(sc, f1);
2073
2074         if (copy != 0)
2075             add_f1_to_sfl(sc, f1);
2076
2077         if (copy != 0)
2078             add_f1_to_sfl(sc, f1);
2079
2080         if (copy != 0)
2081             add_f1_to_sfl(sc, f1);
2082
2083         if (copy != 0)
2084             add_f1_to_sfl(sc, f1);
2085
2086         if (copy != 0)
2087             add_f1_to_sfl(sc, f1);
2088
2089         if (copy != 0)
2090             add_f1_to_sfl(sc, f1);
2091
2092         if (copy != 0)
2093             add_f1_to_sfl(sc, f1);
2094
2095         if (copy != 0)
2096             add_f1_to_sfl(sc, f1);
2097
2098         if (copy != 0)
2099             add_f1_to_sfl(sc, f1);
2100
2101         if (copy != 0)
2102             add_f1_to_sfl(sc, f1);
2103
2104         if (copy != 0)
2105             add_f1_to_sfl(sc, f1);
2106
2107         if (copy != 0)
2108             add_f1_to_sfl(sc, f1);
2109
2110         if (copy != 0)
2111             add_f1_to_sfl(sc, f1);
2112
2113         if (copy != 0)
2114             add_f1_to_sfl(sc, f1);
2115
2116         if (copy != 0)
2117             add_f1_to_sfl(sc, f1);
2118
2119         if (copy != 0)
2120             add_f1_to_sfl(sc, f1);
2121
2122         if (copy != 0)
2123             add_f1_to_sfl(sc, f1);
2124
2125         if (copy != 0)
2126             add_f1_to_sfl(sc, f1);
2127
2128         if (copy != 0)
2129             add_f1_to_sfl(sc, f1);
2130
2131         if (copy != 0)
2132             add_f1_to_sfl(sc, f1);
2133
2134         if (copy != 0)
2135             add_f1_to_sfl(sc, f1);
2136
2137         if (copy != 0)
2138             add_f1_to_sfl(sc, f1);
2139
2140         if (copy != 0)
2141             add_f1_to_sfl(sc, f1);
2142
2143         if (copy != 0)
2144             add_f1_to_sfl(sc, f1);
2145
2146         if (copy != 0)
2147             add_f1_to_sfl(sc, f1);
2148
2149         if (copy != 0)
2150             add_f1_to_sfl(sc, f1);
2151
2152         if (copy != 0)
2153             add_f1_to_sfl(sc, f1);
2154
2155         if (copy != 0)
2156             add_f1_to_sfl(sc, f1);
2157
2158         if (copy != 0)
2159             add_f1_to_sfl(sc, f1);
2160
2161         if (copy != 0)
2162             add_f1_to_sfl(sc, f1);
2163
2164         if (copy != 0)
2165             add_f1_to_sfl(sc, f1);
2166
2167         if (copy != 0)
2168             add_f1_to_sfl(sc, f1);
2169
2170         if (copy != 0)
2171             add_f1_to_sfl(sc, f1);
2172
2173         if (copy != 0)
2174             add_f1_to_sfl(sc, f1);
2175
2176         if (copy != 0)
2177             add_f1_to_sfl(sc, f1);
2178
2179         if (copy != 0)
2180             add_f1_to_sfl(sc, f1);
2181
2182         if (copy != 0)
2183             add_f1_to_sfl(sc, f1);
2184
2185         if (copy != 0)
2186             add_f1_to_sfl(sc, f1);
2187
2188         if (copy != 0)
2189             add_f1_to_sfl(sc, f1);
2190
2191         if (copy != 0)
2192             add_f1_to_sfl(sc, f1);
2193
2194         if (copy != 0)
2195             add_f1_to_sfl(sc, f1);
2196
2197         if (copy != 0)
2198             add_f1_to_sfl(sc, f1);
2199
2200         if (copy != 0)
2201             add_f1_to_sfl(sc, f1);
2202
2203         if (copy != 0)
2204             add_f1_to_sfl(sc, f1);
2205
2206         if (copy != 0)
2207             add_f1_to_sfl(sc, f1);
2208
2209         if (copy != 0)
2210             add_f1_to_sfl(sc, f1);
2211
2212         if (copy != 0)
2213             add_f1_to_sfl(sc, f1);
2214
2215         if (copy != 0)
2216             add_f1_to_sfl(sc, f1);
2217
2218         if (copy != 0)
2219             add_f1_to_sfl(sc, f1);
2220
2221         if (copy != 0)
2222             add_f1_to_sfl(sc, f1);
2223
2224         if (copy != 0)
2225             add_f1_to_sfl(sc, f1);
2226
2227         if (copy != 0)
2228             add_f1_to_sfl(sc, f1);
2229
2230         if (copy != 0)
2231             add_f1_to_sfl(sc, f1);
2232
2233         if (copy != 0)
2234             add_f1_to_sfl(sc, f1);
2235
2236         if (copy != 0)
2237             add_f1_to_sfl(sc, f1);
2238
2239         if (copy != 0)
2240             add_f1_to_sfl(sc, f1);
2241
2242         if (copy != 0)
2243             add_f1_to_sfl(sc, f1);
2244
2245         if (copy != 0)
2246             add_f1_to_sfl(sc, f1);
2247
2248         if (copy != 0)
2249             add_f1_to_sfl(sc, f
```