```
/*
    libxbee - a C library to aid the use of Digi's Series 1 XBee modules
              running in API mode (AP=2).

    Copyright (C) 2009  Attie Grande (attie@attie.co.uk)

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "globals.h"
#include "api.h"

/* ready flag.
   needs to be set to -1 so that the listen thread can begin.
   then 1 so that functions can be used (after setup of course...) */
volatile int xbee_ready = 0;

/* ######################################################################## */
/* ### Memory Handling ################################################## */
/* ######################################################################## */

/* malloc wrapper function */
static void *Xmalloc(size_t size) {
  void *t;
  t = malloc(size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("xbee:malloc()");
    exit(1);
  }
  return t;
}

/* calloc wrapper function */
static void *Xcalloc(size_t size) {
  void *t;
  t = calloc(1, size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("xbee:calloc()");
    exit(1);
  }
  return t;
}

/* realloc wrapper function */
static void *Xrealloc(void *ptr, size_t size) {
  void *t;
  t = realloc(ptr,size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("xbee:realloc()");
    exit(1);
  }
  return t;
}

/* free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
static void Xfree2(void **ptr) {
  free(*ptr);
  *ptr = NULL;
}

/* ######################################################################## */
/* ### Helper Functions ################################################## */
/* ######################################################################## */

/* ########################################################################
   returns 1 if the packet has data for the digital input else 0 */
int xbee_hasdigital(xbee_pkt *pkt, int sample, int input) {
  int mask = 0x0001;
  if (input < 0 || input > 7) return 0;
  if (sample >= pkt->samples) return 0;

```

```
  mask <<= input;
  return !!(pkt->IOdata[sample].IOmask & mask);
}

/* ################################################################
   returns 1 if the digital input is high else 0 (or 0 if no digital data present) */
int xbee_getdigital(xbee_pkt *pkt, int sample, int input) {
  int mask = 0x0001;
  if (!xbee_hasdigital(pkt,sample,input)) return 0;

  mask <<= input;
  return !!(pkt->IOdata[sample].IOdigital & mask);
}

/* ################################################################
   returns 1 if the packet has data for the analog input else 0 */
int xbee_hasanalog(xbee_pkt *pkt, int sample, int input) {
  int mask = 0x0200;
  if (input < 0 || input > 5) return 0;
  if (sample >= pkt->samples) return 0;

  mask <<= input;
  return !!(pkt->IOdata[sample].IOmask & mask);
}

/* ################################################################
   returns analog input as a voltage if vRef is non-zero, else raw value (or 0 if no analog data present) */
double xbee_getanalog(xbee_pkt *pkt, int sample, int input, double Vref) {
  if (!xbee_hasanalog(pkt,sample,input)) return 0;

  if (Vref) return (Vref / 1023) * pkt->IOdata[sample].IOanalog[input];
  return pkt->IOdata[sample].IOanalog[input];
}

/* ################################################################ */
/* ### XBee Functions ############################################# */
/* ################################################################ */

/* ################################################################
   xbee_setup
   opens xbee serial port & creates xbee listen thread
   the xbee must be configured for API mode 2
   THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
int xbee_setup(char *path, int baudrate) {
  return xbee_setuplog(path,baudrate,0);
}
int xbee_setuplog(char *path, int baudrate, int logfd) {
  t_info info;
  struct flock fl;
  struct termios tc;
  speed_t chosenbaud;

#ifdef DEBUG
  xbee.logfd = ((logfd)?logfd:stdout);
#else
  xbee.logfd = logfd;
#endif
  if (xbee.logfd) {
    xbee.log = fdopen(xbee.logfd,"w");
    if (!xbee.log) {
      /* errno == 9 is bad file descriptor (probrably not provided) */
      if (errno != 9) perror("Failed opening logfile");
      xbee.logfd = 0;
    } else {
      /* set to line buffer - ensure lines are written to file when complete */
      setvbuf(xbee.log,NULL,_IOLBF,BUFSIZ);
    }
  }

  if (xbee.log) fprintf(xbee.log,"libxbee: Starting (r%s)\n",svn_version());

  /* select the baud rate */
  switch (baudrate) {
    case 1200:  chosenbaud = B1200;   break;
    case 2400:  chosenbaud = B2400;   break;
    case 4800:  chosenbaud = B4800;   break;
    case 9600:  chosenbaud = B9600;   break;
    case 19200: chosenbaud = B19200;  break;
    case 38400: chosenbaud = B38400;  break;
    case 57600: chosenbaud = B57600;  break;
    case 115200:chosenbaud = B115200; break;
    default:
      fprintf(stderr,"XBee: Unknown or incompatiable baud rate specified... (%d)\n",baudrate);
      return -1;
  };
```

```
171:
172:   /* setup the connection mutex */
173:   xbee.conlist = NULL;
174:   if (pthread_mutex_init(&xbee.conmutex,NULL)) {
175:     perror("xbee_setup():pthread_mutex_init(conmutex)");
176:     return -1;
177:   }
178:
179:   /* setup the packet mutex */
180:   xbee.pktlist = NULL;
181:   if (pthread_mutex_init(&xbee.pktmutex,NULL)) {
182:     perror("xbee_setup():pthread_mutex_init(pktmutex)");
183:     return -1;
184:   }
185:
186:   /* setup the send mutex */
187:   if (pthread_mutex_init(&xbee.sendmutex,NULL)) {
188:     perror("xbee_setup():pthread_mutex_init(sendmutex)");
189:     return -1;
190:   }
191:
192:   /* take a copy of the XBee device path */
193:   if ((xbee.path = Xmalloc(sizeof(char) * (strlen(path) + 1))) == NULL) {
194:     perror("xbee_setup():Xmalloc(path)");
195:     return -1;
196:   }
197:   strcpy(xbee.path,path);
198:
199:   /* open the serial port as a file descriptor */
200:   if ((xbee.ttyfd = open(path,O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1) {
201:     perror("xbee_setup():open()");
202:     Xfree(xbee.path);
203:     xbee.ttyfd = -1;
204:     xbee.tty = NULL;
205:     return -1;
206:   }
207:
208:   /* lock the file */
209:   fl.l_type = F_WRLCK | F_RDLCK;
210:   fl.l_whence = SEEK_SET;
211:   fl.l_start = 0;
212:   fl.l_len = 0;
213:   fl.l_pid = getpid();
214:   if (fcntl(xbee.ttyfd, F_SETLK, &fl) == -1) {
215:     perror("xbee_setup():fcntl()");
216:     Xfree(xbee.path);
217:     close(xbee.ttyfd);
218:     xbee.ttyfd = -1;
219:     xbee.tty = NULL;
220:     return -1;
221:   }
222:
223:
224:   /* open the serial port as a FILE* */
225:   if ((xbee.tty = fdopen(xbee.ttyfd,"r+")) == NULL) {
226:     perror("xbee_setup():fdopen()");
227:     Xfree(xbee.path);
228:     close(xbee.ttyfd);
229:     xbee.ttyfd = -1;
230:     xbee.tty = NULL;
231:     return -1;
232:   }
233:
234:   /* flush the serial port */
235:   fflush(xbee.tty);
236:
237:   /* setup the baud rate and other io attributes */
238:   tcgetattr(xbee.ttyfd, &tc);
239:   /* input flags */
240:   tc.c_iflag &= ~IGNBRK;           /* enable ignoring break */
241:   tc.c_iflag &= ~(IGNPAR | PARMRK);/* disable parity checks */
242:   tc.c_iflag &= ~INPCK;            /* disable parity checking */
243:   tc.c_iflag &= ~ISTRIP;           /* disable stripping 8th bit */
244:   tc.c_iflag &= ~(INLCR | ICRNL);  /* disable translating NL <-> CR */
245:   tc.c_iflag &= ~IGNCR;            /* disable ignoring CR */
246:   tc.c_iflag &= ~(IXON | IXOFF);   /* disable XON/XOFF flow control */
247:   /* output flags */
248:   tc.c_oflag &= ~OPOST;            /* disable output processing */
249:   tc.c_oflag &= ~(ONLCR | OCRNL);  /* disable translating NL <-> CR */
250:   tc.c_oflag &= ~OFILL;            /* disable fill characters */
251:   /* control flags */
252:   tc.c_cflag |= CREAD;             /* enable reciever */
253:   tc.c_cflag &= ~PARENB;           /* disable parity */
254:   tc.c_cflag &= ~CSTOPB;           /* disable 2 stop bits */
255:   tc.c_cflag &= ~CSIZE;            /* remove size flag... */
```

```
256:   tc.c_cflag |= CS8;              /* ...enable 8 bit characters */
257:   tc.c_cflag |= HUPCL;            /* enable lower control lines on close - hang up */
258:   /* local flags */
259:   tc.c_lflag &= ~ISIG;            /* disable generating signals */
260:   tc.c_lflag &= ~ICANON;          /* disable canonical mode - line by line */
261:   tc.c_lflag &= ~ECHO;            /* disable echoing characters */
262:   tc.c_lflag &= ~ECHONL;          /* ??? */
263:   tc.c_lflag &= ~NOFLSH;          /* disable flushing on SIGINT */
264:   tc.c_lflag &= ~IEXTEN;          /* disable input processing */
265:   /* control characters */
266:   memset(tc.c_cc,0,sizeof(tc.c_cc));
267:   /* i/o rates */
268:   cfsetspeed(&tc, chosenbaud);     /* set i/o baud rate */
269:   tcsetattr(xbee.ttyfd, TCSANOW, &tc);
270:   tcflow(xbee.ttyfd, TCOON|TCION); /* enable input & output transmission */
271:
272:   /* allow the listen thread to start */
273:   xbee_ready = -1;
274:
275:   /* can start xbee_listen thread now */
276:   if (pthread_create(&xbee.listent,NULL,(void *(*)(void *))xbee_listen_wrapper,(void *)&info) != 0) {
277:     perror("xbee_setup():pthread_create()");
278:     Xfree(xbee.path);
279:     fclose(xbee.tty);
280:     close(xbee.ttyfd);
281:     xbee.ttyfd = -1;
282:     xbee.tty = NULL;
283:     return -1;
284:   }
285:
286:   usleep(100);
287:   while (xbee_ready != -2) {
288:     usleep(100);
289:     if (xbee.logfd) {
290:       fprintf(xbee.log,"XBee: Waiting for xbee_listen() to be ready...\n");
291:     }
292:   }
293:
294:   /* allow other functions to be used! */
295:   xbee_ready = 1;
296:
297:   return 0;
298: }
299:
300: /* ################################################################
301:    xbee_con
302:    produces a connection to the specified device and frameID
303:    if a connection had already been made, then this connection will be returned */
304: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
305:   xbee_con *con, *ocon;
306:   unsigned char tAddr[8];
307:   va_list ap;
308:   int t;
309:   int i;
310:
311:   ISREADY;
312:
313:   if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
314:   else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
315:
316:   va_start(ap,type);
317:   /* if: 64 bit address expected (2 ints) */
318:   if ((type == xbee_64bitRemoteAT) ||
319:       (type == xbee_64bitData) ||
320:       (type == xbee_64bitIO)) {
321:     t = va_arg(ap, int);
322:     tAddr[0] = (t >> 24) & 0xFF;
323:     tAddr[1] = (t >> 16) & 0xFF;
324:     tAddr[2] = (t >>  8) & 0xFF;
325:     tAddr[3] = (t      ) & 0xFF;
326:     t = va_arg(ap, int);
327:     tAddr[4] = (t >> 24) & 0xFF;
328:     tAddr[5] = (t >> 16) & 0xFF;
329:     tAddr[6] = (t >>  8) & 0xFF;
330:     tAddr[7] = (t      ) & 0xFF;
331:
332:   /* if: 16 bit address expected (1 int) */
333:   } else if ((type == xbee_16bitRemoteAT) ||
334:              (type == xbee_16bitData) ||
335:              (type == xbee_16bitIO)) {
336:     t = va_arg(ap, int);
337:     tAddr[0] = (t >>  8) & 0xFF;
338:     tAddr[1] = (t      ) & 0xFF;
339:     tAddr[2] = 0;
340:     tAddr[3] = 0;
```

```
341:     tAddr[4] = 0;
342:     tAddr[5] = 0;
343:     tAddr[6] = 0;
344:     tAddr[7] = 0;
345:
346:   /* otherwise clear the address */
347:   } else {
348:     memset(tAddr,0,8);
349:   }
350:   va_end(ap);
351:
352:   /* lock the connection mutex */
353:   pthread_mutex_lock(&xbee.conmutex);
354:
355:   /* are there any connections? */
356:   if (xbee.conlist) {
357:     con = xbee.conlist;
358:     while (con) {
359:       /* if: after a modemStatus, and the types match! */
360:       if ((type == xbee_modemStatus) &&
361:           (con->type == type)) {
362:         pthread_mutex_unlock(&xbee.conmutex);
363:         return con;
364:
365:       /* if: after a txStatus and frameIDs match! */
366:       } else if ((type == xbee_txStatus) &&
367:                  (con->type == type) &&
368:                  (frameID == con->frameID)) {
369:         pthread_mutex_unlock(&xbee.conmutex);
370:         return con;
371:
372:       /* if: after a localAT, and the frameIDs match! */
373:       } else if ((type == xbee_localAT) &&
374:                  (con->type == type) &&
375:                  (frameID == con->frameID)) {
376:         pthread_mutex_unlock(&xbee.conmutex);
377:         return con;
378:
379:       /* if: connection types match, the frameIDs match, and the addresses match! */
380:       } else if ((type == con->type) &&
381:                  (frameID == con->frameID) &&
382:                  (!memcmp(tAddr,con->tAddr,8))) {
383:         pthread_mutex_unlock(&xbee.conmutex);
384:         return con;
385:       }
386:
387:       /* if there are more, move along, dont want to loose that last item! */
388:       if (con->next == NULL) break;
389:       con = con->next;
390:     }
391:
392:     /* keep hold of the last connection... we will need to link it up later */
393:     ocon = con;
394:   }
395:
396:   /* create a new connection and set its attributes */
397:   con = Xcalloc(sizeof(xbee_con));
398:   con->type = type;
399:   /* is it a 64bit connection? */
400:   if ((type == xbee_64bitRemoteAT) ||
401:       (type == xbee_64bitData) ||
402:       (type == xbee_64bitIO)) {
403:     con->tAddr64 = TRUE;
404:   }
405:   con->atQueue = 0; /* queue AT commands? */
406:   con->txDisableACK = 0; /* disable ACKs? */
407:   con->txBroadcast = 0; /* broadcast? */
408:   con->frameID = frameID;
409:   memcpy(con->tAddr,tAddr,8); /* copy in the remote address */
410:
411:   if (xbee.logfd) {
412:     switch(type) {
413:     case xbee_localAT:
414:       fprintf(xbee.log,"XBee: New local AT connection!\n");
415:       break;
416:     case xbee_16bitRemoteAT:
417:     case xbee_64bitRemoteAT:
418:       fprintf(xbee.log,"XBee: New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
419:       for (i=0;i<(con->tAddr64?8:2);i++) {
420:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
421:       }
422:       fprintf(xbee.log,")\n");
423:       break;
424:     case xbee_16bitData:
425:     case xbee_64bitData:
```

```
426:       fprintf(xbee.log,"XBee: New %d-bit data connection! (to: ",(con->tAddr64?64:16));
427:       for (i=0;i<(con->tAddr64?8:2);i++) {
428:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
429:       }
430:       fprintf(xbee.log,")\n");
431:       break;
432:     case xbee_16bitIO:
433:     case xbee_64bitIO:
434:       fprintf(xbee.log,"XBee: New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
435:       for (i=0;i<(con->tAddr64?8:2);i++) {
436:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
437:       }
438:       fprintf(xbee.log,")\n");
439:       break;
440:     case xbee_txStatus:
441:       fprintf(xbee.log,"XBee: New Tx status connection!\n");
442:       break;
443:     case xbee_modemStatus:
444:       fprintf(xbee.log,"XBee: New modem status connection!\n");
445:       break;
446:     case xbee_unknown:
447:     default:
448:       fprintf(xbee.log,"XBee: New unknown connection!\n");
449:     }
450:   }
451:
452:   /* make it the last in the list */
453:   con->next = NULL;
454:   /* add it to the list */
455:   if (xbee.conlist) {
456:     ocon->next = con;
457:   } else {
458:     xbee.conlist = con;
459:   }
460:
461:   /* unlock the mutex */
462:   pthread_mutex_unlock(&xbee.conmutex);
463:   return con;
464: }
465:
466: /* ################################################################
467:    xbee_conflush
468:    removes any packets that have been collected for the specified
469:    connection */
470: void xbee_flushcon(xbee_con *con) {
471:   xbee_pkt *r, *p;
472:
473:   /* lock the packet mutex */
474:   pthread_mutex_lock(&xbee.pktmutex);
475:
476:   /* if: there are packets */
477:   if ((p = xbee.pktlist) != NULL) {
478:     r = NULL;
479:     /* get all packets for this connection */
480:     do {
481:       /* does the packet match the connection? */
482:       if (xbee_matchpktcon(p,con)) {
483:         /* if it was the first packet */
484:         if (!r) {
485:           /* move the chain along */
486:           xbee.pktlist = p->next;
487:         } else {
488:           /* otherwise relink the list */
489:           r->next = p->next;
490:         }
491:
492:         /* free this packet! */
493:         Xfree(p);
494:       }
495:       /* move on */
496:       r = p;
497:       p = p->next;
498:     } while (p);
499:   }
500:
501:   /* unlock the packet mutex */
502:   pthread_mutex_unlock(&xbee.pktmutex);
503: }
504:
505: /* ################################################################
506:    xbee_endcon
507:    close the unwanted connection
508:    free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
509: void xbee_endcon2(xbee_con **con) {
510:   xbee_con *t, *u;
```

```
511:
512:   /* lock the connection mutex */
513:   pthread_mutex_lock(&xbee.conmutex);
514:
515:   u = t = xbee.conlist;
516:   while (t && t != *con) {
517:     u = t;
518:     t = t->next;
519:   }
520:   if (!u) {
521:     /* invalid connection given... */
522:     if (xbee.logfd) {
523:       fprintf(xbee.log,"XBee: Attempted to close invalid connection...\n");
524:     }
525:     /* unlock the connection mutex */
526:     pthread_mutex_unlock(&xbee.conmutex);
527:     return;
528:   }
529:   /* extract this connection from the list */
530:   u->next = u->next->next;
531:
532:   /* unlock the connection mutex */
533:   pthread_mutex_unlock(&xbee.conmutex);
534:
535:   /* remove all packets for this connection */
536:   xbee_flushcon(*con);
537:
538:   /* free the connection! */
539:   Xfree(*con);
540: }
541:
542: /* ##################################################################
543:    xbee_senddata
544:    send the specified data to the provided connection */
545: int xbee_senddata(xbee_con *con, char *format, ...) {
546:   int ret;
547:   va_list ap;
548:
549:   ISREADY;
550:
551:   /* xbee_vsenddata() wants a va_list... */
552:   va_start(ap, format);
553:   /* hand it over :) */
554:   ret = xbee_vsenddata(con,format,ap);
555:   va_end(ap);
556:   return ret;
557: }
558:
559: int xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
560:   unsigned char data[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
561:   int length;
562:
563:   ISREADY;
564:
565:   /* make up the data and keep the length, its possible there are nulls in there */
566:   length = vsnprintf((char *)data,128,format,ap);
567:
568:   /* hand it over :) */
569:   return xbee_nsenddata(con,(char *)data,length);
570: }
571:
572: int xbee_nsenddata(xbee_con *con, char *data, int length) {
573:   t_data *pkt;
574:   int i;
575:   unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
576:
577:   ISREADY;
578:
579:   if (!con) return -1;
580:   if (con->type == xbee_unknown) return -1;
581:   if (length > 127) return -1;
582:
583:   if (xbee.logfd) {
584:     fprintf(xbee.log,"XBee: --== TX Packet ============--\n");
585:     fprintf(xbee.log,"XBee: Length: %d\n",length);
586:     for (i=0;i<length;i++) {
587:       fprintf(xbee.log,"XBee: %3d | 0x%02X ",i,data[i]);
588:       if ((data[i] > 32) && (data[i] < 127)) {
589:         fprintf(xbee.log,"'%c'\n",data[i]);
590:       } else{
591:         fprintf(xbee.log," _\n");
592:       }
593:     }
594:   }
595:
```

```
596:   /* ######################################### */
597:   /* if: local AT */
598:   if (con->type == xbee_localAT) {
599:     /* AT commands are 2 chars long (plus optional parameter) */
600:     if (length < 2) return -1;
601:
602:     /* use the command? */
603:     buf[0] = ((!con->atQueue)?0x08:0x09);
604:     buf[1] = con->frameID;
605:
606:     /* copy in the data */
607:     for (i=0;i<length;i++) {
608:       buf[i+2] = data[i];
609:     }
610:
611:     /* setup the packet */
612:     pkt = xbee_make_pkt(buf,i+2);
613:     /* send it on */
614:     xbee_send_pkt(pkt);
615:
616:     return 0;
617:
618:   /* ######################################### */
619:   /* if: remote AT */
620:   } else if ((con->type == xbee_16bitRemoteAT) ||
621:              (con->type == xbee_64bitRemoteAT)) {
622:     if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
623:     buf[0] = 0x17;
624:     buf[1] = con->frameID;
625:
626:     /* copy in the relevant address */
627:     if (con->tAddr64) {
628:       memcpy(&buf[2],con->tAddr,8);
629:       buf[10] = 0xFF;
630:       buf[11] = 0xFE;
631:     } else {
632:       memset(&buf[2],0,8);
633:       memcpy(&buf[10],con->tAddr,2);
634:     }
635:     /* queue the command? */
636:     buf[12] = ((!con->atQueue)?0x02:0x00);
637:
638:     /* copy in the data */
639:     for (i=0;i<length;i++) {
640:       buf[i+13] = data[i];
641:     }
642:
643:     /* setup the packet */
644:     pkt = xbee_make_pkt(buf,i+13);
645:     /* send it on */
646:     xbee_send_pkt(pkt);
647:
648:     return 0;
649:
650:   /* ######################################### */
651:   /* if: 16 or 64bit Data */
652:   } else if ((con->type == xbee_16bitData) ||
653:              (con->type == xbee_64bitData)) {
654:     int offset;
655:
656:     /* if: 16bit Data */
657:     if (con->type == xbee_16bitData) {
658:       buf[0] = 0x01;
659:       offset = 5;
660:       /* copy in the address */
661:       memcpy(&buf[2],con->tAddr,2);
662:
663:     /* if: 64bit Data */
664:     } else { /* 64bit Data */
665:       buf[0] = 0x00;
666:       offset = 11;
667:       /* copy in the address */
668:       memcpy(&buf[2],con->tAddr,8);
669:     }
670:
671:     /* copy frameID */
672:     buf[1] = con->frameID;
673:
674:     /* disable ack? broadcast? */
675:     buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
676:
677:     /* copy in the data */
678:     for (i=0;i<length;i++) {
679:       buf[i+offset] = data[i];
680:     }
```

```
681:
682:     /* setup the packet */
683:     pkt = xbee_make_pkt(buf,i+offset);
684:     /* send it on */
685:     xbee_send_pkt(pkt);
686:
687:     return 0;
688:
689:   /* ######################################### */
690:   /* if: I/O */
691:   } else if ((con->type == xbee_64bitIO) ||
692:              (con->type == xbee_16bitIO)) {
693:     /* not currently implemented... is it even allowed? */
694:     if (xbee.logfd) {
695:       fprintf(xbee.log,"******* TODO ********\n");
696:     }
697:   }
698:
699:   return -2;
700: }
701:
702: /* ###################################################################
703:    xbee_getpacket
704:    retrieves the next packet destined for the given connection
705:    once the packet has been retrieved, it is removed for the list! */
706: xbee_pkt *xbee_getpacketwait(xbee_con *con) {
707:   xbee_pkt *p;
708:   int i;
709:
710:   /* 50ms * 20 = 1 second */
711:   for (i = 0; i < 20; i++) {
712:     p = xbee_getpacket(con);
713:     if (p) break;
714:     usleep(50000); /* 50ms */
715:   }
716:
717:   return p;
718: }
719: xbee_pkt *xbee_getpacket(xbee_con *con) {
720:   xbee_pkt *l, *p, *q;
721:   int c;
722:   if (xbee.logfd) {
723:     fprintf(xbee.log,"XBee: --== Get Packet ==========--\n");
724:   }
725:
726:   /* lock the packet mutex */
727:   pthread_mutex_lock(&xbee.pktmutex);
728:
729:   /* if: there are no packets */
730:   if ((p = xbee.pktlist) == NULL) {
731:     pthread_mutex_unlock(&xbee.pktmutex);
732:     if (xbee.logfd) {
733:       fprintf(xbee.log,"XBee: No packets avaliable...\n");
734:     }
735:     return NULL;
736:   }
737:
738:   l = NULL;
739:   q = NULL;
740:   /* get the first avaliable packet for this connection */
741:   do {
742:     /* does the packet match the connection? */
743:     if (xbee_matchpktcon(p,con)) {
744:       q = p;
745:       break;
746:     }
747:     /* move on */
748:     l = p;
749:     p = p->next;
750:   } while (p);
751:
752:   /* if: no packet was found */
753:   if (!q) {
754:     pthread_mutex_unlock(&xbee.pktmutex);
755:     if (xbee.logfd) {
756:       fprintf(xbee.log,"XBee: No packets avaliable (for connection)...\n");
757:     }
758:     return NULL;
759:   }
760:
761:   /* if it was not the first packet */
762:   if (l) {
763:     /* otherwise relink the list */
764:     l->next = p->next;
765:   } else {
```

```
766:     /* move the chain along */
767:     xbee.pktlist = p->next;
768:   }
769:
770:   /* unlink this packet from the chain! */
771:   q->next = NULL;
772:
773:   if (xbee.logfd) {
774:     fprintf(xbee.log,"XBee: Got a packet\n");
775:     for (p = xbee.pktlist,c = 0;p;c++,p = p->next);
776:     fprintf(xbee.log,"XBee: Packets left: %d\n",c);
777:   }
778:
779:   /* unlock the packet mutex */
780:   pthread_mutex_unlock(&xbee.pktmutex);
781:
782:   /* and return the packet (must be freed by caller!) */
783:   return q;
784: }
785:
786: /* ################################################################
787:    xbee_matchpktcon - INTERNAL
788:    checks if the packet matches the connection */
789: static int xbee_matchpktcon(xbee_pkt *pkt, xbee_con *con) {
790:   /* if: the connection type matches the packet type OR
791:      the connection is 16/64bit remote AT, and the packet is a remote AT response */
792:   if ((pkt->type == con->type) || /* -- */
793:       ((pkt->type == xbee_remoteAT) && /* -- */
794:        ((con->type == xbee_16bitRemoteAT) ||
795:         (con->type == xbee_64bitRemoteAT)))) {
796:
797:     /* if: the packet is modem status OR
798:        the packet is tx status or AT data and the frame IDs match OR
799:        the addresses match */
800:     if (pkt->type == xbee_modemStatus) return 1;
801:
802:     if ((pkt->type == xbee_txStatus) ||
803:         (pkt->type == xbee_localAT) ||
804:         (pkt->type == xbee_remoteAT)) {
805:       if (pkt->frameID == con->frameID) {
806:         return 1;
807:       }
808:     } else if (pkt->sAddr64 && !memcmp(pkt->Addr64,con->tAddr,8)) {
809:       return 1;
810:     } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16,con->tAddr,2)) {
811:       return 1;
812:     }
813:   }
814:   return 0;
815: }
816:
817: /* ################################################################
818:    xbee_parse_io - INTERNAL
819:    parses the data given into the packet io information */
820: static int xbee_parse_io(xbee_pkt *p, unsigned char *d, int maskOffset, int sampleOffset, int sample) {
821:   xbee_sample *s = &(p->IOdata[sample]);
822:
823:   /* copy in the I/O data mask */
824:   s->IOmask = (((d[maskOffset]<<8) | d[maskOffset + 1]) & 0x7FFF);
825:
826:   /* copy in the digital I/O data */
827:   s->IOdigital = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x01FF);
828:
829:   /* advance over the digital data, if its there */
830:   sampleOffset += ((s->IOmask & 0x01FF)?2:0);
831:
832:   /* copy in the analog I/O data */
833:   if (s->IOmask & 0x0200) {
834:     s->IOanalog[0] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
835:     sampleOffset+=2;
836:   }
837:   if (s->IOmask & 0x0400) {
838:     s->IOanalog[1] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
839:     sampleOffset+=2;
840:   }
841:   if (s->IOmask & 0x0800) {
842:     s->IOanalog[2] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
843:     sampleOffset+=2;
844:   }
845:   if (s->IOmask & 0x1000) {
846:     s->IOanalog[3] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
847:     sampleOffset+=2;
848:   }
849:   if (s->IOmask & 0x2000) {
850:     s->IOanalog[4] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
```

```
851:     sampleOffset+=2;
852:   }
853:   if (s->IOmask & 0x4000) {
854:     s->IOanalog[5] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
855:     sampleOffset+=2;
856:   }
857:
858:   if (xbee.logfd) {
859:     if (s->IOmask & 0x0001)
860:       fprintf(xbee.log,"XBee: Digital 0: %c\n",((s->IOdigital & 0x0001)?'1':'0'));
861:     if (s->IOmask & 0x0002)
862:       fprintf(xbee.log,"XBee: Digital 1: %c\n",((s->IOdigital & 0x0002)?'1':'0'));
863:     if (s->IOmask & 0x0004)
864:       fprintf(xbee.log,"XBee: Digital 2: %c\n",((s->IOdigital & 0x0004)?'1':'0'));
865:     if (s->IOmask & 0x0008)
866:       fprintf(xbee.log,"XBee: Digital 3: %c\n",((s->IOdigital & 0x0008)?'1':'0'));
867:     if (s->IOmask & 0x0010)
868:       fprintf(xbee.log,"XBee: Digital 4: %c\n",((s->IOdigital & 0x0010)?'1':'0'));
869:     if (s->IOmask & 0x0020)
870:       fprintf(xbee.log,"XBee: Digital 5: %c\n",((s->IOdigital & 0x0020)?'1':'0'));
871:     if (s->IOmask & 0x0040)
872:       fprintf(xbee.log,"XBee: Digital 6: %c\n",((s->IOdigital & 0x0040)?'1':'0'));
873:     if (s->IOmask & 0x0080)
874:       fprintf(xbee.log,"XBee: Digital 7: %c\n",((s->IOdigital & 0x0080)?'1':'0'));
875:     if (s->IOmask & 0x0100)
876:       fprintf(xbee.log,"XBee: Digital 8: %c\n",((s->IOdigital & 0x0100)?'1':'0'));
877:     if (s->IOmask & 0x0200)
878:       fprintf(xbee.log,"XBee: Analog  0: %d (~%.2fv)\n",s->IOanalog[0],(3.3/1023)*s->IOanalog[0]);
879:     if (s->IOmask & 0x0400)
880:       fprintf(xbee.log,"XBee: Analog  1: %d (~%.2fv)\n",s->IOanalog[1],(3.3/1023)*s->IOanalog[1]);
881:     if (s->IOmask & 0x0800)
882:       fprintf(xbee.log,"XBee: Analog  2: %d (~%.2fv)\n",s->IOanalog[2],(3.3/1023)*s->IOanalog[2]);
883:     if (s->IOmask & 0x1000)
884:       fprintf(xbee.log,"XBee: Analog  3: %d (~%.2fv)\n",s->IOanalog[3],(3.3/1023)*s->IOanalog[3]);
885:     if (s->IOmask & 0x2000)
886:       fprintf(xbee.log,"XBee: Analog  4: %d (~%.2fv)\n",s->IOanalog[4],(3.3/1023)*s->IOanalog[4]);
887:     if (s->IOmask & 0x4000)
888:       fprintf(xbee.log,"XBee: Analog  5: %d (~%.2fv)\n",s->IOanalog[5],(3.3/1023)*s->IOanalog[5]);
889:   }
890:
891:   return sampleOffset;
892: }
893:
894: /* ####################################################################
895:    xbee_listen_wrapper - INTERNAL
896:    the xbee_listen wrapper. Prints an error when xbee_listen ends */
897: static void xbee_listen_wrapper(t_info *info) {
898:   int ret;
899:
900:   /* just falls out if the proper 'go-ahead' isn't given */
901:   if (xbee_ready != -1) return;
902:   /* now allow the parent to continue */
903:   xbee_ready = -2;
904:
905:   info->i = -1;
906:   for (;;) {
907:     ret = xbee_listen(info);
908:     if (xbee.logfd) {
909:       fprintf(xbee.log,"XBee: xbee_listen() returned [%d]... Restarting in 250ms!\n",ret);
910:     }
911:     usleep(25000);
912:   }
913: }
914:
915: /* xbee_listen - INTERNAL
916:    the xbee xbee_listen thread
917:    reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
918: static int xbee_listen(t_info *info) {
919:   unsigned char c, t, d[1024];
920:   unsigned int l, i, chksum, o;
921:   int j;
922:   xbee_pkt *p, *q, *po;
923:   xbee_con *con;
924:   int hasCon;
925:
926:   /* just falls out if the proper 'go-ahead' isn't given */
927:   if (info->i != -1) return -1;
928:
929:   /* do this forever :) */
930:   while(1) {
931:     /* wait for a valid start byte */
932:     if (xbee_getRawByte() != 0x7E) continue;
933:
934:     if (xbee.logfd) {
935:       fprintf(xbee.log,"XBee: --== RX Packet ===========--\nXBee: Got a packet!...\n");
```

```
936:     }
937:
938:     /* get the length */
939:     l = xbee_getByte() << 8;
940:     l += xbee_getByte();
941:
942:     /* check it is a valid length... */
943:     if (!l) {
944:       if (xbee.logfd) {
945:         fprintf(xbee.log,"XBee: Recived zero length packet!\n");
946:       }
947:       continue;
948:     }
949:     if (l > 100) {
950:       if (xbee.logfd) {
951:         fprintf(xbee.log,"XBee: Recived oversized packet! Length: %d\n",l - 1);
952:       }
953:     }
954:     if (l > sizeof(d) - 1) {
955:       if (xbee.logfd) {
956:         fprintf(xbee.log,"XBee: Recived packet larger than buffer! Discarding... Length: %d\n",l - 1);
957:       }
958:       continue;
959:     }
960:
961:     if (xbee.logfd) {
962:       fprintf(xbee.log,"XBee: Length: %d\n",l - 1);
963:     }
964:
965:     /* get the packet type */
966:     t = xbee_getByte();
967:
968:     /* start the checksum */
969:     chksum = t;
970:
971:     /* suck in all the data */
972:     for (i = 0; l > 1 && i < 128; l--, i++) {
973:       /* get an unescaped byte */
974:       c = xbee_getByte();
975:       d[i] = c;
976:       chksum += c;
977:       if (xbee.logfd) {
978:         fprintf(xbee.log,"XBee: %3d | 0x%02X | ",i,c);
979:         if ((c > 32) && (c < 127)) fprintf(xbee.log,"'%c'\n",c); else fprintf(xbee.log," _\n");
980:       }
981:     }
982:     i--; /* it went up too many times!... */
983:
984:     /* add the checksum */
985:     chksum += xbee_getByte();
986:
987:     /* check if the whole packet was recieved, or something else occured... unlikely... */
988:     if (l>1) {
989:       if (xbee.logfd) {
990:         fprintf(xbee.log,"XBee: Didn't get whole packet... :(\n");
991:       }
992:       continue;
993:     }
994:
995:     /* check the checksum */
996:     if ((chksum & 0xFF) != 0xFF) {
997:       if (xbee.logfd) {
998:         fprintf(xbee.log,"XBee: Invalid Checksum: 0x%02X\n",chksum);
999:       }
1000:      continue;
1001:    }
1002:
1003:    /* make a new packet */
1004:    po = p = Xcalloc(sizeof(xbee_pkt));
1005:    q = NULL;
1006:    p->datalen = 0;
1007:
1008:    /* ######################################### */
1009:    /* if: modem status */
1010:    if (t == 0x8A) {
1011:      if (xbee.logfd) {
1012:        fprintf(xbee.log,"XBee: Packet type: Modem Status (0x8A)\n");
1013:        fprintf(xbee.log,"XBee: ");
1014:        switch (d[0]) {
1015:        case 0x00: fprintf(xbee.log,"Hardware reset"); break;
1016:        case 0x01: fprintf(xbee.log,"Watchdog timer reset"); break;
1017:        case 0x02: fprintf(xbee.log,"Associated"); break;
1018:        case 0x03: fprintf(xbee.log,"Disassociated"); break;
1019:        case 0x04: fprintf(xbee.log,"Synchronization lost"); break;
1020:        case 0x05: fprintf(xbee.log,"Coordinator realignment"); break;
```

```
1021:         case 0x06: fprintf(xbee.log,"Coordinator started"); break;
1022:         }
1023:         fprintf(xbee.log,"... (0x%02X)\n",d[0]);
1024:       }
1025:       p->type = xbee_modemStatus;
1026:
1027:       p->sAddr64 = FALSE;
1028:       p->dataPkt = FALSE;
1029:       p->txStatusPkt = FALSE;
1030:       p->modemStatusPkt = TRUE;
1031:       p->remoteATPkt = FALSE;
1032:       p->IOPkt = FALSE;
1033:
1034:       /* modem status can only ever give 1 'data' byte */
1035:       p->datalen = 1;
1036:       p->data[0] = d[0];
1037:
1038:     /* ######################################### */
1039:     /* if: local AT response */
1040:     } else if (t == 0x88) {
1041:       if (xbee.logfd) {
1042:         fprintf(xbee.log,"XBee: Packet type: Local AT Response (0x88)\n");
1043:         fprintf(xbee.log,"XBee: FrameID: 0x%02X\n",d[0]);
1044:         fprintf(xbee.log,"XBee: AT Command: %c%c\n",d[1],d[2]);
1045:         fprintf(xbee.log,"XBee: Status: ");
1046:         if (d[3] == 0) fprintf(xbee.log,"OK");
1047:         else if (d[3] == 1) fprintf(xbee.log,"Error");
1048:         else if (d[3] == 2) fprintf(xbee.log,"Invalid Command");
1049:         else if (d[3] == 3) fprintf(xbee.log,"Invalid Parameter");
1050:         fprintf(xbee.log," (0x%02X)\n",d[3]);
1051:       }
1052:       p->type = xbee_localAT;
1053:
1054:       p->sAddr64 = FALSE;
1055:       p->dataPkt = FALSE;
1056:       p->txStatusPkt = FALSE;
1057:       p->modemStatusPkt = FALSE;
1058:       p->remoteATPkt = FALSE;
1059:       p->IOPkt = FALSE;
1060:
1061:       p->frameID = d[0];
1062:       p->atCmd[0] = d[1];
1063:       p->atCmd[1] = d[2];
1064:
1065:       p->status = d[3];
1066:
1067:       /* copy in the data */
1068:       p->datalen = i-3;
1069:       for (;i>3;i--) p->data[i-4] = d[i];
1070:
1071:     /* ######################################### */
1072:     /* if: remote AT response */
1073:     } else if (t == 0x97) {
1074:       if (xbee.logfd) {
1075:         fprintf(xbee.log,"XBee: Packet type: Remote AT Response (0x97)\n");
1076:         fprintf(xbee.log,"XBee: FrameID: 0x%02X\n",d[0]);
1077:         fprintf(xbee.log,"XBee: 64-bit Address: ");
1078:         for (j=0;j<8;j++) {
1079:           fprintf(xbee.log,(j?":%02X":"%02X"),d[1+j]);
1080:         }
1081:         fprintf(xbee.log,"\n");
1082:         fprintf(xbee.log,"XBee: 16-bit Address: ");
1083:         for (j=0;j<2;j++) {
1084:           fprintf(xbee.log,(j?":%02X":"%02X"),d[9+j]);
1085:         }
1086:         fprintf(xbee.log,"\n");
1087:         fprintf(xbee.log,"XBee: AT Command: %c%c\n",d[11],d[12]);
1088:         fprintf(xbee.log,"XBee: Status: ");
1089:         if (d[13] == 0) fprintf(xbee.log,"OK");
1090:         else if (d[13] == 1) fprintf(xbee.log,"Error");
1091:         else if (d[13] == 2) fprintf(xbee.log,"Invalid Command");
1092:         else if (d[13] == 3) fprintf(xbee.log,"Invalid Parameter");
1093:         else if (d[13] == 4) fprintf(xbee.log,"No Response");
1094:         fprintf(xbee.log," (0x%02X)\n",d[13]);
1095:       }
1096:       p->type = xbee_remoteAT;
1097:
1098:       p->sAddr64 = FALSE;
1099:       p->dataPkt = FALSE;
1100:       p->txStatusPkt = FALSE;
1101:       p->modemStatusPkt = FALSE;
1102:       p->remoteATPkt = TRUE;
1103:       p->IOPkt = FALSE;
1104:
1105:       p->frameID = d[0];
```

```
1106:
1107:       p->Addr64[0] = d[1];
1108:       p->Addr64[1] = d[2];
1109:       p->Addr64[2] = d[3];
1110:       p->Addr64[3] = d[4];
1111:       p->Addr64[4] = d[5];
1112:       p->Addr64[5] = d[6];
1113:       p->Addr64[6] = d[7];
1114:       p->Addr64[7] = d[8];
1115:
1116:       p->Addr16[0] = d[9];
1117:       p->Addr16[1] = d[10];
1118:
1119:       p->atCmd[0] = d[11];
1120:       p->atCmd[1] = d[12];
1121:
1122:       p->status = d[13];
1123:
1124:       p->samples = 1;
1125:
1126:       if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
1127:         /* parse the io data */
1128:         if (xbee.logfd) fprintf(xbee.log,"XBee: --- Sample -----------------\n");
1129:         xbee_parse_io(p, d, 15, 17, 0);
1130:         if (xbee.logfd) fprintf(xbee.log,"XBee: ---------------------------\n");
1131:       } else {
1132:         /* copy in the data */
1133:         p->datalen = i-13;
1134:         for (;i>13;i--) p->data[i-14] = d[i];
1135:       }
1136:
1137:     /* ############################################ */
1138:     /* if: TX status */
1139:     } else if (t == 0x89) {
1140:       if (xbee.logfd) {
1141:         fprintf(xbee.log,"XBee: Packet type: TX Status Report (0x89)\n");
1142:         fprintf(xbee.log,"XBee: FrameID: 0x%02X\n",d[0]);
1143:         fprintf(xbee.log,"XBee: Status: ");
1144:         if (d[1] == 0) fprintf(xbee.log,"Success");
1145:         else if (d[1] == 1) fprintf(xbee.log,"No ACK");
1146:         else if (d[1] == 2) fprintf(xbee.log,"CCA Failure");
1147:         else if (d[1] == 3) fprintf(xbee.log,"Purged");
1148:         fprintf(xbee.log," (0x%02X)\n",d[13]);
1149:       }
1150:       p->type = xbee_txStatus;
1151:
1152:       p->sAddr64 = FALSE;
1153:       p->dataPkt = FALSE;
1154:       p->txStatusPkt = TRUE;
1155:       p->modemStatusPkt = FALSE;
1156:       p->remoteATPkt = FALSE;
1157:       p->IOPkt = FALSE;
1158:
1159:       p->frameID = d[0];
1160:
1161:       p->status = d[1];
1162:
1163:       /* never returns data */
1164:       p->datalen = 0;
1165:
1166:     /* ############################################ */
1167:     /* if: 16 / 64bit data recieve */
1168:     } else if ((t == 0x80) ||
1169:                (t == 0x81)) {
1170:       int offset;
1171:       if (t == 0x80) { /* 64bit */
1172:         offset = 8;
1173:       } else { /* 16bit */
1174:
1175:         offset = 2;
1176:       }
1177:       if (xbee.logfd) {
1178:         fprintf(xbee.log,"XBee: Packet type: %d-bit RX Data (0x%02X)\n",((t == 0x80)?64:16),t);
1179:         fprintf(xbee.log,"XBee: %d-bit Address: ",((t == 0x80)?64:16));
1180:         for (j=0;j<offset;j++) {
1181:           fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1182:         }
1183:         fprintf(xbee.log,"\n");
1184:         fprintf(xbee.log,"XBee: RSSI: -%ddB\n",d[offset]);
1185:         if (d[offset + 1] & 0x02) fprintf(xbee.log,"XBee: Options: Address Broadcast\n");
1186:         if (d[offset + 1] & 0x03) fprintf(xbee.log,"XBee: Options: PAN Broadcast\n");
1187:       }
1188:       p->dataPkt = TRUE;
1189:       p->txStatusPkt = FALSE;
1190:       p->modemStatusPkt = FALSE;
```

```
1191:       p->remoteATPkt = FALSE;
1192:       p->IOPkt = FALSE;
1193:
1194:       if (t == 0x80) { /* 64bit */
1195:         p->type = xbee_64bitData;
1196:
1197:         p->sAddr64 = TRUE;
1198:
1199:         p->Addr64[0] = d[0];
1200:         p->Addr64[1] = d[1];
1201:         p->Addr64[2] = d[2];
1202:         p->Addr64[3] = d[3];
1203:         p->Addr64[4] = d[4];
1204:         p->Addr64[5] = d[5];
1205:         p->Addr64[6] = d[6];
1206:         p->Addr64[7] = d[7];
1207:       } else { /* 16bit */
1208:         p->type = xbee_16bitData;
1209:
1210:         p->sAddr64 = FALSE;
1211:
1212:         p->Addr16[0] = d[0];
1213:         p->Addr16[1] = d[1];
1214:       }
1215:
1216:       /* save the RSSI / signal strength
1217:          this can be used with printf as:
1218:          printf("-%ddB\n",p->RSSI); */
1219:       p->RSSI = d[offset];
1220:
1221:       p->status = d[offset + 1];
1222:
1223:       /* copy in the data */
1224:       p->datalen = i-(offset + 1);
1225:       for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
1226:
1227:     /* ######################################### */
1228:     /* if: 16 / 64bit I/O recieve */
1229:     } else if ((t == 0x82) ||
1230:                (t == 0x83)) {
1231:       int offset;
1232:       if (t == 0x82) { /* 64bit */
1233:         p->type = xbee_64bitIO;
1234:
1235:         p->sAddr64 = TRUE;
1236:
1237:         p->Addr64[0] = d[0];
1238:         p->Addr64[1] = d[1];
1239:         p->Addr64[2] = d[2];
1240:         p->Addr64[3] = d[3];
1241:         p->Addr64[4] = d[4];
1242:         p->Addr64[5] = d[5];
1243:         p->Addr64[6] = d[6];
1244:         p->Addr64[7] = d[7];
1245:
1246:         offset = 8;
1247:         p->samples = d[10];
1248:       } else { /* 16bit */
1249:         p->type = xbee_16bitIO;
1250:
1251:         p->sAddr64 = FALSE;
1252:
1253:         p->Addr16[0] = d[0];
1254:         p->Addr16[1] = d[1];
1255:
1256:         offset = 2;
1257:         p->samples = d[4];
1258:       }
1259:       if (p->samples > 1) {
1260:         p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * (p->samples - 1)));
1261:       }
1262:       if (xbee.logfd) {
1263:         fprintf(xbee.log,"XBee: Packet type: %d-bit RX I/O Data (0x%02X)\n",((t == 0x82)?64:16),t);
1264:         fprintf(xbee.log,"XBee: %d-bit Address: ",((t == 0x82)?64:16));
1265:         for (j = 0; j < offset; j++) {
1266:           fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1267:         }
1268:         fprintf(xbee.log,"\n");
1269:         fprintf(xbee.log,"XBee: RSSI: -%ddB\n",d[offset]);
1270:         if (d[9] & 0x02) fprintf(xbee.log,"XBee: Options: Address Broadcast\n");
1271:         if (d[9] & 0x02) fprintf(xbee.log,"XBee: Options: PAN Broadcast\n");
1272:         fprintf(xbee.log,"XBee: Samples: %d\n",d[offset + 2]);
1273:       }
1274:       i = offset + 5;
1275:
```

```
1276:       /* never returns data */
1277:       p->datalen = 0;
1278:
1279:       p->dataPkt = FALSE;
1280:       p->txStatusPkt = FALSE;
1281:       p->modemStatusPkt = FALSE;
1282:       p->remoteATPkt = FALSE;
1283:       p->IOPkt = TRUE;
1284:
1285:       /* save the RSSI / signal strength
1286:          this can be used with printf as:
1287:          printf("-%ddB\n",p->RSSI); */
1288:       p->RSSI = d[offset];
1289:
1290:       p->status = d[offset + 1];
1291:
1292:       /* each sample is split into its own packet here, for simplicity */
1293:       for (o = 0; o < p->samples; o++) {
1294:         if (xbee.logfd) {
1295:           fprintf(xbee.log,"XBee: --- Sample %3d ------------\n", o);
1296:         }
1297:
1298:         /* parse the io data */
1299:         i = xbee_parse_io(p, d, offset + 3, i, o);
1300:       }
1301:       if (xbee.logfd) {
1302:         fprintf(xbee.log,"XBee: --------------------------\n");
1303:       }
1304:
1305:     /* ########################################## */
1306:     /* if: Unknown */
1307:     } else {
1308:       if (xbee.logfd) {
1309:         fprintf(xbee.log,"XBee: Packet type: Unknown (0x%02X)\n",t);
1310:       }
1311:       p->type = xbee_unknown;
1312:     }
1313:     p->next = NULL;
1314:
1315:     /* lock the connection mutex */
1316:     pthread_mutex_lock(&xbee.conmutex);
1317:
1318:     con = xbee.conlist;
1319:     hasCon = 0;
1320:     while (con) {
1321:       if (xbee_matchpktcon(p,con)) {
1322:         hasCon = 1;
1323:         break;
1324:       }
1325:       con = con->next;
1326:     }
1327:
1328:     /* unlock the connection mutex */
1329:     pthread_mutex_unlock(&xbee.conmutex);
1330:
1331:     /* if the packet doesn't have a connection, don't add it! */
1332:     if (!hasCon) {
1333:       Xfree(p);
1334:       if (xbee.logfd) {
1335:         fprintf(xbee.log,"XBee: Connectionless packet... discarding!\n");
1336:       }
1337:       continue;
1338:     }
1339:
1340:     /* lock the packet mutex, so we can safely add the packet to the list */
1341:     pthread_mutex_lock(&xbee.pktmutex);
1342:     i = 1;
1343:     /* if: the list is empty */
1344:     if (!xbee.pktlist) {
1345:       /* start the list! */
1346:       xbee.pktlist = po;
1347:     } else {
1348:       /* add the packet to the end */
1349:       q = xbee.pktlist;
1350:       while (q->next) {
1351:         q = q->next;
1352:         i++;
1353:       }
1354:       q->next = po;
1355:     }
1356:
1357:     if (xbee.logfd) {
1358:       while (q && q->next) {
1359:         q = q->next;
1360:         i++;
```

```
1361:       }
1362:       fprintf(xbee.log,"XBee: --======================--\n");
1363:       fprintf(xbee.log,"XBee: Packets: %d\n",i);
1364:     }
1365:
1366:     po = p = q = NULL;
1367:
1368:     /* unlock the packet mutex */
1369:     pthread_mutex_unlock(&xbee.pktmutex);
1370:   }
1371: }
1372:
1373: /* ############################################################################
1374:    xbee_getByte - INTERNAL
1375:    waits for an escaped byte of data */
1376: static unsigned char xbee_getByte(void) {
1377:   unsigned char c;
1378:
1379:   ISREADY;
1380:
1381:   /* take a byte */
1382:   c = xbee_getRawByte();
1383:   /* if its escaped, take another and un-escape */
1384:   if (c == 0x7D) c = xbee_getRawByte() ^ 0x20;
1385:
1386:   return (c & 0xFF);
1387: }
1388:
1389: /* ############################################################################
1390:    xbee_getRawByte - INTERNAL
1391:    waits for a raw byte of data */
1392: static unsigned char xbee_getRawByte(void) {
1393:   unsigned char c;
1394:   fd_set fds;
1395:
1396:   ISREADY;
1397:
1398:   /* the loop is just incase there actually isnt a byte there to be read... */
1399:   do {
1400:     /* wait for a read to be possible */
1401:     FD_ZERO(&fds);
1402:     FD_SET(xbee.ttyfd,&fds);
1403:     if (select(xbee.ttyfd+1,&fds,NULL,NULL,NULL) == -1) {
1404:       perror("xbee:xbee_listen():xbee_getRawByte()");
1405:       exit(1);
1406:     }
1407:
1408:     /* read 1 character */
1409:     if (read(xbee.ttyfd,&c,1) == 0) {
1410:       usleep(10);
1411:       continue;
1412:     }
1413:   } while (0);
1414:
1415:   return (c & 0xFF);
1416: }
1417:
1418: /* ############################################################################
1419:    xbee_send_pkt - INTERNAL
1420:    sends a complete packet of data */
1421: static void xbee_send_pkt(t_data *pkt) {
1422:   ISREADY;
1423:
1424:
1425:   /* lock the send mutex */
1426:   pthread_mutex_lock(&xbee.sendmutex);
1427:
1428:   /* write and flush the data */
1429:   fwrite(pkt->data,pkt->length,1,xbee.tty);
1430:   fflush(xbee.tty);
1431:
1432:   /* unlock the mutex */
1433:   pthread_mutex_unlock(&xbee.sendmutex);
1434:
1435:   if (xbee.logfd) {
1436:     int i;
1437:     /* prints packet in hex byte-by-byte */
1438:     fprintf(xbee.log,"XBee: TX Packet - ");
1439:     for (i=0;i<pkt->length;i++) {
1440:       fprintf(xbee.log,"0x%02X ",pkt->data[i]);
1441:     }
1442:     fprintf(xbee.log,"\n");
1443:   }
1444:
1445:   /* free the packet */
```

```
1446:   Xfree(pkt);
1447: }
1448:
1449: /* ####################################################################
1450:    xbee_make_pkt - INTERNAL
1451:    adds delimiter field
1452:    calculates length and checksum
1453:    escapes bytes */
1454: static t_data *xbee_make_pkt(unsigned char *data, int length) {
1455:   t_data *pkt;
1456:   unsigned int l, i, o, t, x, m;
1457:   char d = 0;
1458:
1459:   ISREADY;
1460:
1461:   /* check the data given isnt too long
1462:      100 bytes maximum payload + 12 bytes header information */
1463:   if (length > 100 + 12) return NULL;
1464:
1465:   /* calculate the length of the whole packet
1466:      start, length (MSB), length (LSB), DATA, checksum */
1467:   l = 3 + length + 1;
1468:
1469:   /* prepare memory */
1470:   pkt = Xcalloc(sizeof(t_data));
1471:
1472:   /* put start byte on */
1473:   pkt->data[0] = 0x7E;
1474:
1475:   /* copy data into packet */
1476:   for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
1477:     /* if: its time for the checksum */
1478:     if (i == length) d = M8((0xFF - M8(t)));
1479:     /* if: its time for the high length byte */
1480:     else if (m == 1) d = M8(length >> 8);
1481:     /* if: its time for the low length byte */
1482:     else if (m == 2) d = M8(length);
1483:     /* if: its time for the normal data */
1484:     else if (m > 2) d = data[i];
1485:
1486:     x = 0;
1487:     /* check for any escapes needed */
1488:     if ((d == 0x11) || /* XON */
1489:         (d == 0x13) || /* XOFF */
1490:         (d == 0x7D) || /* Escape */
1491:         (d == 0x7E)) { /* Frame Delimiter */
1492:       l++;
1493:       pkt->data[o++] = 0x7D;
1494:       x = 1;
1495:     }
1496:
1497:     /* move data in */
1498:     pkt->data[o] = ((!x)?d:d^0x20);
1499:     if (m > 2) {
1500:       i++;
1501:       t += d;
1502:     }
1503:   }
1504:
1505:   /* remember the length */
1506:   pkt->length = l;
1507:
1508:   return pkt;
1509: }
```