```c
/*
    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...) */
int xbee_ready = 0;

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

/* malloc wrapper function */
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 */
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 */
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) */
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 input) {
  int mask = 0x0001;
  if (input < 0 || input > 7) return 0;

  mask <<= input;
```

```
  86:
  87:   return !!(pkt->IOmask & mask);
  88: }
  89:
  90: /* ################################################################
  91:    returns 1 if the digital input is high else 0 (or 0 if no digital data present) */
  92: int xbee_getdigital(xbee_pkt *pkt, int input) {
  93:   int mask = 0x0001;
  94:   if (input < 0 || input > 7) return 0;
  95:
  96:   if (!xbee_hasdigital(pkt,input)) return 0;
  97:
  98:   mask <<= input;
  99:   return !!(pkt->IOdata & mask);
 100: }
 101:
 102: /* ################################################################
 103:    returns 1 if the packet has data for the analog input else 0 */
 104: int xbee_hasanalog(xbee_pkt *pkt, int input) {
 105:   int mask = 0x0200;
 106:   if (input < 0 || input > 5) return 0;
 107:
 108:   mask <<= input;
 109:
 110:   return !!(pkt->IOmask & mask);
 111: }
 112:
 113: /* ################################################################
 114:    returns analog input as a voltage if vRef is non-zero, else raw value (or 0 if no analog data present) */
 115: double xbee_getanalog(xbee_pkt *pkt, int input, double Vref) {
 116:   if (input < 0 || input > 5) return 0;
 117:   if (!xbee_hasanalog(pkt,input)) return 0;
 118:
 119:   if (Vref) return (Vref / 1024) * pkt->IOanalog[0];
 120:   return pkt->IOanalog[input];
 121: }
 122:
 123: /* ################################################################ */
 124: /* ### XBee Functions ############################################ */
 125: /* ################################################################ */
 126:
 127: /* ################################################################
 128:    xbee_setup
 129:    opens xbee serial port & creates xbee listen thread
 130:    the xbee must be configured for API mode 2
 131:    THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
 132: int xbee_setup(char *path, int baudrate) {
 133:   return xbee_setuplog(path,baudrate,0);
 134: }
 135: int xbee_setuplog(char *path, int baudrate, int logfd) {
 136:   t_info info;
 137:   struct flock fl;
 138:   struct termios tc;
 139:   speed_t chosenbaud;
 140:
 141: #ifdef DEBUG
 142:   xbee.logfd = ((logfd)?logfd:stdout);
 143: #else
 144:   xbee.logfd = logfd;
 145: #endif
 146:   if (xbee.logfd) {
 147:     xbee.log = fdopen(xbee.logfd,"w");
 148:     if (!xbee.log) {
 149:       /* errno == 9 is bad file descriptor (probrably not provided) */
 150:       if (errno != 9) perror("Failed opening logfile");
 151:       xbee.logfd = 0;
 152:     }
 153:   }
 154:
 155:   /* select the baud rate */
 156:   switch (baudrate) {
 157:     case 1200:  chosenbaud = B1200;   break;
 158:     case 2400:  chosenbaud = B2400;   break;
 159:     case 4800:  chosenbaud = B4800;   break;
 160:     case 9600:  chosenbaud = B9600;   break;
 161:     case 19200: chosenbaud = B19200;  break;
 162:     case 38400: chosenbaud = B38400;  break;
 163:     case 57600: chosenbaud = B57600;  break;
 164:     case 115200:chosenbaud = B115200; break;
 165:     default:
 166:       fprintf(stderr,"XBee: Unknown or incompatiable baud rate specified... (%d)\n",baudrate);
 167:       return -1;
 168:   };
 169:
 170:   /* setup the connection mutex */
```

```
171:   xbee.conlist = NULL;
172:   if (pthread_mutex_init(&xbee.conmutex,NULL)) {
173:     perror("xbee_setup():pthread_mutex_init(conmutex)");
174:     return -1;
175:   }
176:
177:   /* setup the packet mutex */
178:   xbee.pktlist = NULL;
179:   if (pthread_mutex_init(&xbee.pktmutex,NULL)) {
180:     perror("xbee_setup():pthread_mutex_init(pktmutex)");
181:     return -1;
182:   }
183:
184:   /* setup the send mutex */
185:   if (pthread_mutex_init(&xbee.sendmutex,NULL)) {
186:     perror("xbee_setup():pthread_mutex_init(sendmutex)");
187:     return -1;
188:   }
189:
190:   /* take a copy of the XBee device path */
191:   if ((xbee.path = malloc(sizeof(char) * (strlen(path) + 1))) == NULL) {
192:     perror("xbee_setup():malloc(path)");
193:     return -1;
194:   }
195:   strcpy(xbee.path,path);
196:
197:   /* open the serial port as a file descriptor */
198:   if ((xbee.ttyfd = open(path,O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1) {
199:     perror("xbee_setup():open()");
200:     Xfree(xbee.path);
201:     xbee.ttyfd = -1;
202:     xbee.tty = NULL;
203:     return -1;
204:   }
205:
206:   /* lock the file */
207:   fl.l_type = F_WRLCK | F_RDLCK;
208:   fl.l_whence = SEEK_SET;
209:   fl.l_start = 0;
210:   fl.l_len = 0;
211:   fl.l_pid = getpid();
212:   if (fcntl(xbee.ttyfd, F_SETLK, &fl) == -1) {
213:     perror("xbee_setup():fcntl()");
214:     Xfree(xbee.path);
215:     close(xbee.ttyfd);
216:     xbee.ttyfd = -1;
217:     xbee.tty = NULL;
218:     return -1;
219:   }
220:
221:
222:   /* setup the baud rate and other io attributes */
223:   tcgetattr(xbee.ttyfd, &tc);
224:   cfsetispeed(&tc, chosenbaud);    /* set input baud rate */
225:   cfsetospeed(&tc, chosenbaud);    /* set output baud rate */
226:   /* input flags */
227:   tc.c_iflag |= IGNBRK;            /* enable ignoring break */
228:   tc.c_iflag &= ~(IGNPAR | PARMRK);/* disable parity checks */
229:   tc.c_iflag &= ~INPCK;            /* disable parity checking */
230:   tc.c_iflag &= ~ISTRIP;           /* disable stripping 8th bit */
231:   tc.c_iflag &= ~(INLCR | ICRNL);  /* disable translating NL <-> CR */
232:   tc.c_iflag &= ~IGNCR;            /* disable ignoring CR */
233:   tc.c_iflag &= ~(IXON | IXOFF);   /* disable XON/XOFF flow control */
234:   /* output flags */
235:   tc.c_oflag &= ~OPOST;            /* disable output processing */
236:   tc.c_oflag &= ~(ONLCR | OCRNL);  /* disable translating NL <-> CR */
237:   tc.c_oflag &= ~OFILL;            /* disable fill characters */
238:   /* control flags */
239:   tc.c_cflag |= CREAD;             /* enable reciever */
240:   tc.c_cflag &= ~PARENB;           /* disable parity */
241:   tc.c_cflag &= ~CSTOPB;           /* disable 2 stop bits */
242:   tc.c_cflag &= ~CSIZE;            /* remove size flag... */
243:   tc.c_cflag |= CS8;               /* ...enable 8 bit characters */
244:   tc.c_cflag |= HUPCL;             /* enable lower control lines on close - hang up */
245:   /* local flags */
246:   tc.c_lflag &= ~ISIG;             /* disable generating signals */
247:   tc.c_lflag &= ~ICANON;           /* disable canonical mode - line by line */
248:   tc.c_lflag &= ~ECHO;             /* disable echoing characters */
249:   tc.c_lflag &= ~NOFLSH;           /* disable flushing on SIGINT */
250:   tc.c_lflag &= ~IEXTEN;           /* disable input processing */
251:   tcsetattr(xbee.ttyfd, TCSANOW, &tc);
252:
253:   /* open the serial port as a FILE* */
254:   if ((xbee.tty = fdopen(xbee.ttyfd,"r+")) == NULL) {
255:     perror("xbee_setup():fdopen()");
```

```
256:     Xfree(xbee.path);
257:     close(xbee.ttyfd);
258:     xbee.ttyfd = -1;
259:     xbee.tty = NULL;
260:     return -1;
261:   }
262:
263:   /* flush the serial port */
264:   fflush(xbee.tty);
265:
266:   /* allow the listen thread to start */
267:   xbee_ready = -1;
268:
269:   /* can start xbee_listen thread now */
270:   if (pthread_create(&xbee.listent,NULL,(void *(*)(void *))xbee_listen,(void *)&info) != 0) {
271:     perror("xbee_setup():pthread_create()");
272:     Xfree(xbee.path);
273:     fclose(xbee.tty);
274:     close(xbee.ttyfd);
275:     xbee.ttyfd = -1;
276:     xbee.tty = NULL;
277:     return -1;
278:   }
279:
280:   /* allow other functions to be used! */
281:   xbee_ready = 1;
282:
283:   return 0;
284: }
285:
286: /* ################################################################
287:    xbee_con
288:    produces a connection to the specified device and frameID
289:    if a connection had already been made, then this connection will be returned */
290: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
291:   xbee_con *con, *ocon;
292:   unsigned char tAddr[8];
293:   va_list ap;
294:   int t;
295:   int i;
296:
297:   ISREADY;
298:
299:   if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
300:   else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
301:
302:   va_start(ap,type);
303:   /* if: 64 bit address expected (2 ints) */
304:   if ((type == xbee_64bitRemoteAT) ||
305:       (type == xbee_64bitData) ||
306:       (type == xbee_64bitIO)) {
307:     t = va_arg(ap, int);
308:     tAddr[0] = (t >> 24) & 0xFF;
309:     tAddr[1] = (t >> 16) & 0xFF;
310:     tAddr[2] = (t >>  8) & 0xFF;
311:     tAddr[3] = (t      ) & 0xFF;
312:     t = va_arg(ap, int);
313:     tAddr[4] = (t >> 24) & 0xFF;
314:     tAddr[5] = (t >> 16) & 0xFF;
315:     tAddr[6] = (t >>  8) & 0xFF;
316:     tAddr[7] = (t      ) & 0xFF;
317:
318:   /* if: 16 bit address expected (1 int) */
319:   } else if ((type == xbee_16bitRemoteAT) ||
320:              (type == xbee_16bitData) ||
321:              (type == xbee_16bitIO)) {
322:     t = va_arg(ap, int);
323:     tAddr[0] = (t >>  8) & 0xFF;
324:     tAddr[1] = (t      ) & 0xFF;
325:     tAddr[2] = 0;
326:     tAddr[3] = 0;
327:     tAddr[4] = 0;
328:     tAddr[5] = 0;
329:     tAddr[6] = 0;
330:     tAddr[7] = 0;
331:
332:   /* otherwise clear the address */
333:   } else {
334:     memset(tAddr,0,8);
335:   }
336:   va_end(ap);
337:
338:   /* lock the connection mutex */
339:   pthread_mutex_lock(&xbee.conmutex);
340:
```

```
341:   /* are there any connections? */
342:   if (xbee.conlist) {
343:     con = xbee.conlist;
344:     while (con) {
345:       /* if: after a modemStatus, and the types match! */
346:       if ((type == xbee_modemStatus) &&
347:           (con->type == type)) {
348:         pthread_mutex_unlock(&xbee.conmutex);
349:         return con;
350:
351:       /* if: after a txStatus and frameIDs match! */
352:       } else if ((type == xbee_txStatus) &&
353:                  (con->type == type) &&
354:                  (frameID == con->frameID)) {
355:         pthread_mutex_unlock(&xbee.conmutex);
356:         return con;
357:
358:       /* if: after a localAT, and the frameIDs match! */
359:       } else if ((type == xbee_localAT) &&
360:                  (con->type == type) &&
361:                  (frameID == con->frameID)) {
362:         pthread_mutex_unlock(&xbee.conmutex);
363:         return con;
364:
365:       /* if: connection types match, the frameIDs match, and the addresses match! */
366:       } else if ((type == con->type) &&
367:                  (frameID == con->frameID) &&
368:                  (!memcmp(tAddr,con->tAddr,8))) {
369:         pthread_mutex_unlock(&xbee.conmutex);
370:         return con;
371:       }
372:
373:       /* if there are more, move along, dont want to loose that last item! */
374:       if (con->next == NULL) break;
375:       con = con->next;
376:     }
377:
378:     /* keep hold of the last connection... we will need to link it up later */
379:     ocon = con;
380:   }
381:
382:   /* create a new connection and set its attributes */
383:   con = Xcalloc(sizeof(xbee_con));
384:   con->type = type;
385:   /* is it a 64bit connection? */
386:   if ((type == xbee_64bitRemoteAT) ||
387:       (type == xbee_64bitData) ||
388:       (type == xbee_64bitIO)) {
389:     con->tAddr64 = TRUE;
390:   }
391:   con->atQueue = 0; /* queue AT commands? */
392:   con->txDisableACK = 0; /* disable ACKs? */
393:   con->txBroadcast = 0; /* broadcast? */
394:   con->frameID = frameID;
395:   memcpy(con->tAddr,tAddr,8); /* copy in the remote address */
396:
397:   if (xbee.logfd) {
398:     switch(type) {
399:     case xbee_localAT:
400:       fprintf(xbee.log,"XBee: New local AT connection!\n");
401:       break;
402:     case xbee_16bitRemoteAT:
403:     case xbee_64bitRemoteAT:
404:       fprintf(xbee.log,"XBee: New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
405:       for (i=0;i<(con->tAddr64?8:2);i++) {
406:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
407:       }
408:       fprintf(xbee.log,")\n");
409:       break;
410:     case xbee_16bitData:
411:     case xbee_64bitData:
412:       fprintf(xbee.log,"XBee: New %d-bit data connection! (to: ",(con->tAddr64?64:16));
413:       for (i=0;i<(con->tAddr64?8:2);i++) {
414:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
415:       }
416:       fprintf(xbee.log,")\n");
417:       break;
418:     case xbee_16bitIO:
419:     case xbee_64bitIO:
420:       fprintf(xbee.log,"XBee: New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
421:       for (i=0;i<(con->tAddr64?8:2);i++) {
422:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
423:       }
424:       fprintf(xbee.log,")\n");
425:       break;
```

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

```
511:   pthread_mutex_unlock(&xbee.pktmutex);
512:
513:
514:   Xfree(*con);
515: }
516:
517: /* ################################################################
518:    xbee_senddata
519:    send the specified data to the provided connection */
520: int xbee_senddata(xbee_con *con, char *format, ...) {
521:   int ret;
522:   va_list ap;
523:
524:   ISREADY;
525:
526:   /* xbee_vsenddata() wants a va_list... */
527:   va_start(ap, format);
528:   /* hand it over :) */
529:   ret = xbee_vsenddata(con,format,ap);
530:   va_end(ap);
531:   return ret;
532: }
533:
534: int xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
535:   unsigned char data[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
536:   int length;
537:
538:   ISREADY;
539:
540:   /* make up the data and keep the length, its possible there are nulls in there */
541:   length = vsnprintf((char *)data,128,format,ap);
542:
543:   /* hand it over :) */
544:   return xbee_nsenddata(con,(char *)data,length);
545: }
546:
547: int xbee_nsenddata(xbee_con *con, char *data, int length) {
548:   t_data *pkt;
549:   int i;
550:   unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
551:
552:   ISREADY;
553:
554:   if (!con) return -1;
555:   if (con->type == xbee_unknown) return -1;
556:   if (length > 127) return -1;
557:
558:   if (xbee.logfd) {
559:     fprintf(xbee.log,"XBee: --== TX Packet ============--\n");
560:     fprintf(xbee.log,"XBee: Length: %d\n",length);
561:     for (i=0;i<length;i++) {
562:       fprintf(xbee.log,"XBee: %3d | 0x%02X ",i,data[i]);
563:       if ((data[i] > 32) && (data[i] < 127)) {
564:         fprintf(xbee.log,"'%c'\n",data[i]);
565:       } else{
566:         fprintf(xbee.log," _\n");
567:       }
568:     }
569:   }
570:
571:   /* ######################################### */
572:   /* if: local AT */
573:   if (con->type == xbee_localAT) {
574:     /* AT commands are 2 chars long (plus optional parameter) */
575:     if (length < 2) return -1;
576:
577:     /* use the command? */
578:     buf[0] = ((!con->atQueue)?0x08:0x09);
579:     buf[1] = con->frameID;
580:
581:     /* copy in the data */
582:     for (i=0;i<length;i++) {
583:       buf[i+2] = data[i];
584:     }
585:
586:     /* setup the packet */
587:     pkt = xbee_make_pkt(buf,i+2);
588:     /* send it on */
589:     xbee_send_pkt(pkt);
590:
591:     return 0;
592:
593:   /* ######################################### */
594:   /* if: remote AT */
595:   } else if ((con->type == xbee_16bitRemoteAT) ||
```

```
596:              (con->type == xbee_64bitRemoteAT)) {
597:     if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
598:     buf[0] = 0x17;
599:     buf[1] = con->frameID;
600:
601:     /* copy in the relevant address */
602:     if (con->tAddr64) {
603:       memcpy(&buf[2],con->tAddr,8);
604:       buf[10] = 0xFF;
605:       buf[11] = 0xFE;
606:     } else {
607:       memset(&buf[2],0,8);
608:       memcpy(&buf[10],con->tAddr,2);
609:     }
610:     /* queue the command? */
611:     buf[12] = ((!con->atQueue)?0x02:0x00);
612:
613:     /* copy in the data */
614:     for (i=0;i<length;i++) {
615:       buf[i+13] = data[i];
616:     }
617:
618:     /* setup the packet */
619:     pkt = xbee_make_pkt(buf,i+13);
620:     /* send it on */
621:     xbee_send_pkt(pkt);
622:
623:     return 0;
624:
625:   /* ######################################### */
626:   /* if: 16 or 64bit Data */
627:   } else if ((con->type == xbee_16bitData) ||
628:              (con->type == xbee_64bitData)) {
629:     int offset;
630:
631:     /* if: 16bit Data */
632:     if (con->type == xbee_16bitData) {
633:       buf[0] = 0x01;
634:       offset = 5;
635:       /* copy in the address */
636:       memcpy(&buf[2],con->tAddr,2);
637:
638:     /* if: 64bit Data */
639:     } else { /* 64bit Data */
640:       buf[0] = 0x00;
641:       offset = 11;
642:       /* copy in the address */
643:       memcpy(&buf[2],con->tAddr,8);
644:     }
645:
646:     /* copy frameID */
647:     buf[1] = con->frameID;
648:
649:     /* disable ack? broadcast? */
650:     buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
651:
652:     /* copy in the data */
653:     for (i=0;i<length;i++) {
654:       buf[i+offset] = data[i];
655:     }
656:
657:     /* setup the packet */
658:     pkt = xbee_make_pkt(buf,i+offset);
659:     /* send it on */
660:     xbee_send_pkt(pkt);
661:
662:     return 0;
663:
664:   /* ######################################### */
665:   /* if: I/O */
666:   } else if ((con->type == xbee_64bitIO) ||
667:              (con->type == xbee_16bitIO)) {
668:     /* not currently implemented... is it even allowed? */
669:     fprintf(xbee.log,"******* TODO ********\n");
670:   }
671:
672:   return -2;
673: }
674:
675: /* ################################################################### */
676:    xbee_getpacket
677:    retrieves the next packet destined for the given connection
678:    once the packet has been retrieved, it is removed for the list! */
679: xbee_pkt *xbee_getpacketwait(xbee_con *con) {
680:   xbee_pkt *p;
```

```
681:   int i;
682:
683:   /* 50ms * 20 = 1 second */
684:   for (i = 0; i < 20; i++) {
685:     p = xbee_getpacket(con);
686:     if (p) break;
687:     usleep(50000); /* 50ms */
688:   }
689:
690:   return p;
691: }
692: xbee_pkt *xbee_getpacket(xbee_con *con) {
693:   xbee_pkt *l, *p, *q;
694:   int c;
695:   if (xbee.logfd) {
696:     fprintf(xbee.log,"XBee: --== Get Packet ==========--\n");
697:   }
698:
699:   /* lock the packet mutex */
700:   pthread_mutex_lock(&xbee.pktmutex);
701:
702:   /* if: there are no packets */
703:   if ((p = xbee.pktlist) == NULL) {
704:     pthread_mutex_unlock(&xbee.pktmutex);
705:     if (xbee.logfd) {
706:       fprintf(xbee.log,"XBee: No packets avaliable...\n");
707:     }
708:     return NULL;
709:   }
710:
711:   l = NULL;
712:   q = NULL;
713:   /* get the first avaliable packet for this connection */
714:   do {
715:     /* does the packet match the connection? */
716:     if (xbee_matchpktcon(p,con)) {
717:       q = p;
718:       break;
719:     }
720:     /* move on */
721:     l = p;
722:     p = p->next;
723:   } while (p);
724:
725:   /* if: no packet was found */
726:   if (!q) {
727:     pthread_mutex_unlock(&xbee.pktmutex);
728:     if (xbee.logfd) {
729:       fprintf(xbee.log,"XBee: No packets avaliable (for connection)...\n");
730:     }
731:     return NULL;
732:   }
733:
734:   /* if it was not the first packet */
735:   if (l) {
736:     /* otherwise relink the list */
737:     l->next = p->next;
738:   } else {
739:     /* move the chain along */
740:     xbee.pktlist = p->next;
741:   }
742:
743:   /* unlink this packet from the chain! */
744:   q->next = NULL;
745:
746:   if (xbee.logfd) {
747:     fprintf(xbee.log,"XBee: Got a packet\n");
748:     for (p = xbee.pktlist,c = 0;p;c++,p = p->next);
749:     fprintf(xbee.log,"XBee: Packets left: %d\n",c);
750:   }
751:
752:   /* unlock the packet mutex */
753:   pthread_mutex_unlock(&xbee.pktmutex);
754:
755:   /* and return the packet (must be freed by caller!) */
756:   return q;
757: }
758:
759: /* ################################################################
760:    xbee_matchpktcon - INTERNAL
761:    checks if the packet matches the connection */
762: int xbee_matchpktcon(xbee_pkt *pkt, xbee_con *con) {
763:   /* if: the connection type matches the packet type OR
764:      the connection is 16/64bit remote AT, and the packet is a remote AT response */
765:   if ((pkt->type == con->type) || /* -- */
```

```
766:       ((pkt->type == xbee_remoteAT) && /* -- */
767:        ((con->type == xbee_16bitRemoteAT) ||
768:         (con->type == xbee_64bitRemoteAT)))) {
769:
770:     /* if: the packet is modem status OR
771:        the packet is tx status or AT data and the frame IDs match OR
772:        the addresses match */
773:     if (pkt->type == xbee_modemStatus) return 1;
774:
775:     if ((pkt->type == xbee_txStatus) ||
776:         (pkt->type == xbee_localAT) ||
777:         (pkt->type == xbee_remoteAT)) {
778:       if (pkt->frameID == con->frameID) {
779:         return 1;
780:       }
781:     } else if (pkt->sAddr64 && !memcmp(pkt->Addr64,con->tAddr,8)) {
782:       return 1;
783:     } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16,con->tAddr,2)) {
784:       return 1;
785:     }
786:   }
787:   return 0;
788: }
789:
790: /* ###################################################################
791:    xbee_listen - INTERNAL
792:    the xbee xbee_listen thread
793:    reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
794: void xbee_listen(t_info *info) {
795:   unsigned char c, t, d[128];
796:   unsigned int l, i, chksum, o;
797:   int j;
798:   xbee_pkt *p, *q, *po;
799:   xbee_con *con;
800:   int hasCon;
801:
802:   /* just falls out if the proper 'go-ahead' isn't given */
803:   if (xbee_ready != -1) return;
804:
805:   /* do this forever :) */
806:   while(1) {
807:     /* wait for a valid start byte */
808:     if (xbee_getRawByte() != 0x7E) continue;
809:
810:     if (xbee.logfd) {
811:       fprintf(xbee.log,"XBee: --== RX Packet ===========--\nXBee: Got a packet!...\n");
812:     }
813:
814:     /* get the length */
815:     l = xbee_getByte() << 8;
816:     l += xbee_getByte();
817:
818:     /* check it is a valid length... */
819:     if (!l) {
820:       if (xbee.logfd) {
821:         fprintf(xbee.log,"XBee: Recived zero length packet!\n");
822:       }
823:       continue;
824:     }
825:     if (l > 100) {
826:       if (xbee.logfd) {
827:         fprintf(xbee.log,"XBee: Recived oversized packet! Length: %d\n",l - 1);
828:       }
829:       continue;
830:     }
831:
832:     if (xbee.logfd) {
833:       fprintf(xbee.log,"XBee: Length: %d\n",l - 1);
834:     }
835:
836:     /* get the packet type */
837:     t = xbee_getByte();
838:
839:     /* start the checksum */
840:     chksum = t;
841:
842:     /* suck in all the data */
843:     for (i = 0; l > 1 && i < 128; l--, i++) {
844:       /* get an unescaped byte */
845:       c = xbee_getByte();
846:       d[i] = c;
847:       chksum += c;
848:       if (xbee.logfd) {
849:         fprintf(xbee.log,"XBee: %3d | 0x%02X | ",i,c);
850:         if ((c > 32) && (c < 127)) fprintf(xbee.log,"'%c'\n",c); else fprintf(xbee.log," _\n");
```

```
851:       }
852:     }
853:     i--; /* it went up too many times!... */
854:
855:     /* add the checksum */
856:     chksum += xbee_getByte();
857:
858:     /* check if the whole packet was recieved, or something else occured... unlikely... */
859:     if (l>1) {
860:       if (xbee.logfd) {
861:         fprintf(xbee.log,"XBee: Didn't get whole packet... :(\n");
862:       }
863:       continue;
864:     }
865:
866:     /* check the checksum */
867:     if ((chksum & 0xFF) != 0xFF) {
868:       if (xbee.logfd) {
869:         fprintf(xbee.log,"XBee: Invalid Checksum: 0x%02X\n",chksum);
870:       }
871:       continue;
872:     }
873:
874:     /* make a new packet */
875:     po = p = Xcalloc(sizeof(xbee_pkt));
876:     q = NULL;
877:     p->datalen = 0;
878:
879:     /* ######################################### */
880:     /* if: modem status */
881:     if (t == 0x8A) {
882:       if (xbee.logfd) {
883:         fprintf(xbee.log,"XBee: Packet type: Modem Status (0x8A)\n");
884:         fprintf(xbee.log,"XBee: ");
885:         switch (d[0]) {
886:         case 0x00: fprintf(xbee.log,"Hardware reset"); break;
887:         case 0x01: fprintf(xbee.log,"Watchdog timer reset"); break;
888:         case 0x02: fprintf(xbee.log,"Associated"); break;
889:         case 0x03: fprintf(xbee.log,"Disassociated"); break;
890:         case 0x04: fprintf(xbee.log,"Synchronization lost"); break;
891:         case 0x05: fprintf(xbee.log,"Coordinator realignment"); break;
892:         case 0x06: fprintf(xbee.log,"Coordinator started"); break;
893:         }
894:         fprintf(xbee.log,"...\n");
895:       }
896:       p->type = xbee_modemStatus;
897:
898:       p->sAddr64 = FALSE;
899:       p->dataPkt = FALSE;
900:       p->txStatusPkt = FALSE;
901:       p->modemStatusPkt = TRUE;
902:       p->remoteATPkt = FALSE;
903:       p->IOPkt = FALSE;
904:
905:       /* modem status can only ever give 1 'data' byte */
906:       p->datalen = 1;
907:       p->data[0] = d[0];
908:
909:     /* ######################################### */
910:     /* if: local AT response */
911:     } else if (t == 0x88) {
912:       if (xbee.logfd) {
913:         fprintf(xbee.log,"XBee: Packet type: Local AT Response (0x88)\n");
914:         fprintf(xbee.log,"XBee: FrameID: 0x%02X\n",d[0]);
915:         fprintf(xbee.log,"XBee: AT Command: %c%c\n",d[1],d[2]);
916:         if (d[3] == 0) fprintf(xbee.log,"XBee: Status: OK\n");
917:         else if (d[3] == 1) fprintf(xbee.log,"XBee: Status: Error\n");
918:         else if (d[3] == 2) fprintf(xbee.log,"XBee: Status: Invalid Command\n");
919:         else if (d[3] == 3) fprintf(xbee.log,"XBee: Status: Invalid Parameter\n");
920:       }
921:       p->type = xbee_localAT;
922:
923:       p->sAddr64 = FALSE;
924:       p->dataPkt = FALSE;
925:       p->txStatusPkt = FALSE;
926:       p->modemStatusPkt = FALSE;
927:       p->remoteATPkt = FALSE;
928:       p->IOPkt = FALSE;
929:
930:       p->frameID = d[0];
931:       p->atCmd[0] = d[1];
932:       p->atCmd[1] = d[2];
933:
934:       p->status = d[3];
935:
```

```
936:       /* copy in the data */
937:       p->datalen = i-3;
938:       for (;i>3;i--) p->data[i-4] = d[i];
939:
940:     /* ######################################### */
941:     /* if: remote AT response */
942:     } else if (t == 0x97) {
943:       if (xbee.logfd) {
944:         fprintf(xbee.log,"XBee: Packet type: Remote AT Response (0x97)\n");
945:         fprintf(xbee.log,"XBee: FrameID: 0x%02X\n",d[0]);
946:         fprintf(xbee.log,"XBee: 64-bit Address: ");
947:         for (j=0;j<8;j++) {
948:           fprintf(xbee.log,(j?":%02X":"%02X"),d[1+j]);
949:         }
950:         fprintf(xbee.log,"\n");
951:         fprintf(xbee.log,"XBee: 16-bit Address: ");
952:         for (j=0;j<2;j++) {
953:           fprintf(xbee.log,(j?":%02X":"%02X"),d[9+j]);
954:         }
955:         fprintf(xbee.log,"\n");
956:         fprintf(xbee.log,"XBee: AT Command: %c%c\n",d[11],d[12]);
957:         if (d[13] == 0) fprintf(xbee.log,"XBee: Status: OK\n");
958:         else if (d[13] == 1) fprintf(xbee.log,"XBee: Status: Error\n");
959:         else if (d[13] == 2) fprintf(xbee.log,"XBee: Status: Invalid Command\n");
960:         else if (d[13] == 3) fprintf(xbee.log,"XBee: Status: Invalid Parameter\n");
961:         else if (d[13] == 4) fprintf(xbee.log,"XBee: Status: No Response\n");
962:       }
963:       p->type = xbee_remoteAT;
964:
965:       p->sAddr64 = FALSE;
966:       p->dataPkt = FALSE;
967:       p->txStatusPkt = FALSE;
968:       p->modemStatusPkt = FALSE;
969:       p->remoteATPkt = TRUE;
970:       p->IOPkt = FALSE;
971:
972:       p->frameID = d[0];
973:
974:       p->Addr64[0] = d[1];
975:       p->Addr64[1] = d[2];
976:       p->Addr64[2] = d[3];
977:       p->Addr64[3] = d[4];
978:       p->Addr64[4] = d[5];
979:       p->Addr64[5] = d[6];
980:       p->Addr64[6] = d[7];
981:       p->Addr64[7] = d[8];
982:
983:       p->Addr16[0] = d[9];
984:       p->Addr16[1] = d[10];
985:
986:       p->atCmd[0] = d[11];
987:       p->atCmd[1] = d[12];
988:
989:       p->status = d[13];
990:
991:       /* copy in the data */
992:       p->datalen = i-13;
993:       for (;i>13;i--) p->data[i-14] = d[i];
994:
995:     /* ######################################### */
996:     /* if: TX status */
997:     } else if (t == 0x89) {
998:       if (xbee.logfd) {
999:         fprintf(xbee.log,"XBee: Packet type: TX Status Report (0x89)\n");
1000:        fprintf(xbee.log,"XBee: FrameID: 0x%02X\n",d[0]);
1001:        if (d[1] == 0) fprintf(xbee.log,"XBee: Status: Success\n");
1002:        else if (d[1] == 1) fprintf(xbee.log,"XBee: Status: No ACK\n");
1003:        else if (d[1] == 2) fprintf(xbee.log,"XBee: Status: CCA Failure\n");
1004:        else if (d[1] == 3) fprintf(xbee.log,"XBee: Status: Purged\n");
1005:      }
1006:      p->type = xbee_txStatus;
1007:
1008:      p->sAddr64 = FALSE;
1009:      p->dataPkt = FALSE;
1010:      p->txStatusPkt = TRUE;
1011:      p->modemStatusPkt = FALSE;
1012:      p->remoteATPkt = FALSE;
1013:      p->IOPkt = FALSE;
1014:
1015:      p->frameID = d[0];
1016:
1017:      p->status = d[1];
1018:
1019:      /* never returns data */
1020:      p->datalen = 0;
```

```
1021:
1022:     /* ######################################### */
1023:     /* if: 16 / 64bit data recieve */
1024:     } else if ((t == 0x80) ||
1025:                (t == 0x81)) {
1026:       int offset;
1027:       if (t == 0x80) { /* 64bit */
1028:         offset = 8;
1029:       } else { /* 16bit */
1030:
1031:         offset = 2;
1032:       }
1033:       if (xbee.logfd) {
1034:         fprintf(xbee.log,"XBee: Packet type: %d-bit RX Data (0x%02X)\n",((t == 0x80)?64:16),t);
1035:         fprintf(xbee.log,"XBee: %d-bit Address: ",((t == 0x80)?64:16));
1036:         for (j=0;j<offset;j++) {
1037:           fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1038:         }
1039:         fprintf(xbee.log,"\n");
1040:         fprintf(xbee.log,"XBee: RSSI: -%ddB\n",d[offset]);
1041:         if (d[offset + 1] & 0x02) fprintf(xbee.log,"XBee: Options: Address Broadcast\n");
1042:         if (d[offset + 1] & 0x03) fprintf(xbee.log,"XBee: Options: PAN Broadcast\n");
1043:       }
1044:       p->dataPkt = TRUE;
1045:       p->txStatusPkt = FALSE;
1046:       p->modemStatusPkt = FALSE;
1047:       p->remoteATPkt = FALSE;
1048:       p->IOPkt = FALSE;
1049:
1050:       if (t == 0x80) { /* 64bit */
1051:         p->type = xbee_64bitData;
1052:
1053:         p->sAddr64 = TRUE;
1054:
1055:         p->Addr64[0] = d[0];
1056:         p->Addr64[1] = d[1];
1057:         p->Addr64[2] = d[2];
1058:         p->Addr64[3] = d[3];
1059:         p->Addr64[4] = d[4];
1060:         p->Addr64[5] = d[5];
1061:         p->Addr64[6] = d[6];
1062:         p->Addr64[7] = d[7];
1063:       } else { /* 16bit */
1064:         p->type = xbee_16bitData;
1065:
1066:         p->sAddr64 = FALSE;
1067:
1068:         p->Addr16[0] = d[0];
1069:         p->Addr16[1] = d[1];
1070:       }
1071:
1072:       /* save the RSSI / signal strength
1073:          this can be used with printf as:
1074:          printf("-%ddB\n",p->RSSI); */
1075:       p->RSSI = d[offset];
1076:
1077:       p->status = d[offset + 1];
1078:
1079:       /* copy in the data */
1080:       p->datalen = i-(offset + 1);
1081:       for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
1082:
1083:     /* ######################################### */
1084:     /* if: 16 / 64bit I/O recieve */
1085:     } else if ((t == 0x82) ||
1086:                (t == 0x83)) {
1087:       int offset, samples;
1088:       if (t == 0x82) { /* 64bit */
1089:         offset = 8;
1090:         samples = d[10];
1091:       } else { /* 16bit */
1092:         offset = 2;
1093:         samples = d[4];
1094:       }
1095:       if (xbee.logfd) {
1096:         fprintf(xbee.log,"XBee: Packet type: %d-bit RX I/O Data (0x%02X)\n",((t == 0x82)?64:16),t);
1097:         fprintf(xbee.log,"XBee: %d-bit Address: ",((t == 0x82)?64:16));
1098:         for (j = 0; j < offset; j++) {
1099:           fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1100:         }
1101:         fprintf(xbee.log,"\n");
1102:         fprintf(xbee.log,"XBee: RSSI: -%ddB\n",d[offset]);
1103:         if (d[9] & 0x02) fprintf(xbee.log,"XBee: Options: Address Broadcast\n");
1104:         if (d[9] & 0x02) fprintf(xbee.log,"XBee: Options: PAN Broadcast\n");
1105:         fprintf(xbee.log,"XBee: Samples: %d\n",d[offset + 2]);
```

```
1106:       }
1107:       i = offset + 5;
1108:
1109:       /* each sample is split into its own packet here, for simplicity */
1110:       for (o = samples; o > 0; o--) {
1111:         if (xbee.logfd) {
1112:           fprintf(xbee.log,"XBee: --- Sample %3d -------------\n", o - samples + 1);
1113:         }
1114:         /* if we arent still using the origional packet */
1115:         if (o < samples) {
1116:           /* make a new one and link it up! */
1117:           q = Xcalloc(sizeof(xbee_pkt));
1118:           p->next = q;
1119:           p = q;
1120:         }
1121:
1122:         /* never returns data */
1123:         p->datalen = 0;
1124:
1125:         p->dataPkt = FALSE;
1126:         p->txStatusPkt = FALSE;
1127:         p->modemStatusPkt = FALSE;
1128:         p->remoteATPkt = FALSE;
1129:         p->IOPkt = TRUE;
1130:
1131:         if (t == 0x82) { /* 64bit */
1132:           p->type = xbee_64bitIO;
1133:
1134:           p->sAddr64 = TRUE;
1135:
1136:           p->Addr64[0] = d[0];
1137:           p->Addr64[1] = d[1];
1138:           p->Addr64[2] = d[2];
1139:           p->Addr64[3] = d[3];
1140:           p->Addr64[4] = d[4];
1141:           p->Addr64[5] = d[5];
1142:           p->Addr64[6] = d[6];
1143:           p->Addr64[7] = d[7];
1144:         } else { /* 16bit */
1145:           p->type = xbee_16bitIO;
1146:
1147:           p->sAddr64 = FALSE;
1148:
1149:           p->Addr16[0] = d[0];
1150:           p->Addr16[1] = d[1];
1151:         }
1152:
1153:         /* save the RSSI / signal strength
1154:            this can be used with printf as:
1155:            printf("-%ddB\n",p->RSSI); */
1156:         p->RSSI = d[offset];
1157:
1158:         p->status = d[offset + 1];
1159:
1160:         /* copy in the I/O data mask */
1161:         p->IOmask = (((d[offset + 3]<<8) | d[offset + 4]) & 0x7FFF);
1162:
1163:         /* copy in the digital I/O data */
1164:         p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);
1165:
1166:         /* advance over the digital data, if its there */
1167:         i += (((d[offset + 3]&0x01)||(d[offset + 4]))?2:0);
1168:
1169:         /* copy in the analog I/O data */
1170:         if (d[11]&0x02) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1171:         if (d[11]&0x04) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1172:         if (d[11]&0x08) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1173:         if (d[11]&0x10) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1174:         if (d[11]&0x20) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1175:         if (d[11]&0x40) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1176:         if (xbee.logfd) {
1177:           if (p->IOmask & 0x0001) fprintf(xbee.log,"XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
1178:           if (p->IOmask & 0x0002) fprintf(xbee.log,"XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
1179:           if (p->IOmask & 0x0004) fprintf(xbee.log,"XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
1180:           if (p->IOmask & 0x0008) fprintf(xbee.log,"XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
1181:           if (p->IOmask & 0x0010) fprintf(xbee.log,"XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
1182:           if (p->IOmask & 0x0020) fprintf(xbee.log,"XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
1183:           if (p->IOmask & 0x0040) fprintf(xbee.log,"XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
1184:           if (p->IOmask & 0x0080) fprintf(xbee.log,"XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));
1185:           if (p->IOmask & 0x0100) fprintf(xbee.log,"XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));
1186:           if (p->IOmask & 0x0200) fprintf(xbee.log,"XBee: Analog  0: %.2fv\n",(3.3/1023)*p->IOanalog[0]);
1187:           if (p->IOmask & 0x0400) fprintf(xbee.log,"XBee: Analog  1: %.2fv\n",(3.3/1023)*p->IOanalog[1]);
1188:           if (p->IOmask & 0x0800) fprintf(xbee.log,"XBee: Analog  2: %.2fv\n",(3.3/1023)*p->IOanalog[2]);
1189:           if (p->IOmask & 0x1000) fprintf(xbee.log,"XBee: Analog  3: %.2fv\n",(3.3/1023)*p->IOanalog[3]);
1190:           if (p->IOmask & 0x2000) fprintf(xbee.log,"XBee: Analog  4: %.2fv\n",(3.3/1023)*p->IOanalog[4]);
```

```
1191:           if (p->IOmask & 0x4000) fprintf(xbee.log,"XBee: Analog  5: %.2fv\n",(3.3/1023)*p->IOanalog[5]);
1192:         }
1193:       }
1194:       if (xbee.logfd) {
1195:         fprintf(xbee.log,"XBee: --------------------------\n");
1196:       }
1197:
1198:     /* ######################################### */
1199:     /* if: Unknown */
1200:     } else {
1201:       if (xbee.logfd) {
1202:         fprintf(xbee.log,"XBee: Packet type: Unknown (0x%02X)\n",t);
1203:       }
1204:       p->type = xbee_unknown;
1205:     }
1206:     p->next = NULL;
1207:
1208:     /* lock the connection mutex */
1209:     pthread_mutex_lock(&xbee.conmutex);
1210:
1211:     con = xbee.conlist;
1212:     hasCon = 0;
1213:     while (con) {
1214:       if (xbee_matchpktcon(p,con)) {
1215:         hasCon = 1;
1216:         break;
1217:       }
1218:       con = con->next;
1219:     }
1220:
1221:     /* unlock the connection mutex */
1222:     pthread_mutex_unlock(&xbee.conmutex);
1223:
1224:     /* if the packet doesn't have a connection, don't add it! */
1225:     if (!hasCon) {
1226:       Xfree(p);
1227:       if (xbee.logfd) {
1228:         fprintf(xbee.log,"XBee: Connectionless packet... discarding!\n");
1229:       }
1230:       continue;
1231:     }
1232:
1233:     /* lock the packet mutex, so we can safely add the packet to the list */
1234:     pthread_mutex_lock(&xbee.pktmutex);
1235:     i = 1;
1236:     /* if: the list is empty */
1237:     if (!xbee.pktlist) {
1238:       /* start the list! */
1239:       xbee.pktlist = po;
1240:     } else {
1241:       /* add the packet to the end */
1242:       q = xbee.pktlist;
1243:       while (q->next) {
1244:         q = q->next;
1245:         i++;
1246:       }
1247:       q->next = po;
1248:     }
1249:
1250:     if (xbee.logfd) {
1251:       while (q && q->next) {
1252:         q = q->next;
1253:         i++;
1254:       }
1255:       fprintf(xbee.log,"XBee: --======================--\n");
1256:       fprintf(xbee.log,"XBee: Packets: %d\n",i);
1257:     }
1258:
1259:     po = p = q = NULL;
1260:
1261:     /* unlock the packet mutex */
1262:     pthread_mutex_unlock(&xbee.pktmutex);
1263:   }
1264: }
1265:
1266: /* ################################################################
1267:    xbee_getByte - INTERNAL
1268:    waits for an escaped byte of data */
1269: unsigned char xbee_getByte(void) {
1270:   unsigned char c;
1271:
1272:   ISREADY;
1273:
1274:   /* take a byte */
1275:   c = xbee_getRawByte();
```

```
1276:   /* if its escaped, take another and un-escape */
1277:   if (c == 0x7D) c = xbee_getRawByte() ^ 0x20;
1278:
1279:   return (c & 0xFF);
1280: }
1281:
1282: /* ####################################################################
1283:    xbee_getRawByte - INTERNAL
1284:    waits for a raw byte of data */
1285: unsigned char xbee_getRawByte(void) {
1286:   unsigned char c;
1287:   fd_set fds;
1288:
1289:   ISREADY;
1290:
1291:   /* wait for a read to be possible */
1292:   FD_ZERO(&fds);
1293:   FD_SET(xbee.ttyfd,&fds);
1294:   if (select(xbee.ttyfd+1,&fds,NULL,NULL,NULL) == -1) {
1295:     perror("xbee:xbee_listen():xbee_getRawByte()");
1296:     exit(1);
1297:   }
1298:
1299:   /* read 1 character
1300:      the loop is just incase there actually isnt a byte there to be read... */
1301:   do {
1302:     if (read(xbee.ttyfd,&c,1) == 0) {
1303:       usleep(10);
1304:       continue;
1305:     }
1306:   } while (0);
1307:
1308:   return (c & 0xFF);
1309: }
1310:
1311: /* ####################################################################
1312:    xbee_send_pkt - INTERNAL
1313:    sends a complete packet of data */
1314: void xbee_send_pkt(t_data *pkt) {
1315:   ISREADY;
1316:
1317:
1318:   /* lock the send mutex */
1319:   pthread_mutex_lock(&xbee.sendmutex);
1320:
1321:   /* write and flush the data */
1322:   fwrite(pkt->data,pkt->length,1,xbee.tty);
1323:   fflush(xbee.tty);
1324:
1325:   /* unlock the mutex */
1326:   pthread_mutex_unlock(&xbee.sendmutex);
1327:
1328:   if (xbee.logfd) {
1329:     int i;
1330:     /* prints packet in hex byte-by-byte */
1331:     fprintf(xbee.log,"XBee: TX Packet - ");
1332:     for (i=0;i<pkt->length;i++) {
1333:       fprintf(xbee.log,"0x%02X ",pkt->data[i]);
1334:     }
1335:     fprintf(xbee.log,"\n");
1336:   }
1337:
1338:   /* free the packet */
1339:   Xfree(pkt);
1340: }
1341:
1342: /* ####################################################################
1343:    xbee_make_pkt - INTERNAL
1344:    adds delimiter field
1345:    calculates length and checksum
1346:    escapes bytes */
1347: t_data *xbee_make_pkt(unsigned char *data, int length) {
1348:   t_data *pkt;
1349:   unsigned int l, i, o, t, x, m;
1350:   char d = 0;
1351:
1352:   ISREADY;
1353:
1354:   /* check the data given isnt too long
1355:      100 bytes maximum payload + 12 bytes header information */
1356:   if (length > 100 + 12) return NULL;
1357:
1358:   /* calculate the length of the whole packet
1359:      start, length (MSB), length (LSB), DATA, checksum */
1360:   l = 3 + length + 1;
```

```
1361:
1362:   /* prepare memory */
1363:   pkt = Xcalloc(sizeof(t_data));
1364:
1365:   /* put start byte on */
1366:   pkt->data[0] = 0x7E;
1367:
1368:   /* copy data into packet */
1369:   for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
1370:     /* if: its time for the checksum */
1371:     if (i == length) d = M8((0xFF - M8(t)));
1372:     /* if: its time for the high length byte */
1373:     else if (m == 1) d = M8(length >> 8);
1374:     /* if: its time for the low length byte */
1375:     else if (m == 2) d = M8(length);
1376:     /* if: its time for the normal data */
1377:     else if (m > 2) d = data[i];
1378:
1379:     x = 0;
1380:     /* check for any escapes needed */
1381:     if ((d == 0x11) || /* XON */
1382:         (d == 0x13) || /* XOFF */
1383:         (d == 0x7D) || /* Escape */
1384:         (d == 0x7E)) { /* Frame Delimiter */
1385:       l++;
1386:       pkt->data[o++] = 0x7D;
1387:       x = 1;
1388:     }
1389:
1390:     /* move data in */
1391:     pkt->data[o] = ((!x)?d:d^0x20);
1392:     if (m > 2) {
1393:       i++;
1394:       t += d;
1395:     }
1396:   }
1397:
1398:   /* remember the length */
1399:   pkt->length = l;
1400:
1401:   return pkt;
1402: }
```