

```

1: /*
2:     libxbee - a C library to aid the use of Digi's Series 1 XBee modules
3:     running in API mode (AP=2).
4:
5:     Copyright (C) 2009 Attie Grande (attie@attie.co.uk)
6:
7:     This program is free software: you can redistribute it and/or modify
8:     it under the terms of the GNU General Public License as published by
9:     the Free Software Foundation, either version 3 of the License, or
10:    (at your option) any later version.
11:
12:    This program is distributed in the hope that it will be useful,
13:    but WITHOUT ANY WARRANTY; without even the implied warranty of
14:    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
15:    GNU General Public License for more details.
16:
17:    You should have received a copy of the GNU General Public License
18:    along with this program. If not, see <http://www.gnu.org/licenses/>.
19: */
20:
21: #include "globals.h"
22: #include "api.h"
23:
24: /* ready flag.
25:    needs to be set to -1 so that the listen thread can begin.
26:    then 1 so that functions can be used (after setup of course...) */
27: volatile int xbee_ready = 0;
28:
29: /* ##### */
30: /* ### Memory Handling ##### */
31: /* ##### */
32:
33: /* malloc wrapper function */
34: static void *Xmalloc(size_t size) {
35:     void *t;
36:     t = malloc(size);
37:     if (!t) {
38:         /* uhoh... thats pretty bad... */
39:         perror("xbee:malloc()");
40:         exit(1);
41:     }
42:     return t;
43: }
44:
45: /* calloc wrapper function */
46: static void *Xcalloc(size_t size) {
47:     void *t;
48:     t = calloc(1, size);
49:     if (!t) {
50:         /* uhoh... thats pretty bad... */
51:         perror("xbee:calloc()");
52:         exit(1);
53:     }
54:     return t;
55: }
56:
57: /* realloc wrapper function */
58: static void *Xrealloc(void *ptr, size_t size) {
59:     void *t;
60:     t = realloc(ptr, size);
61:     if (!t) {
62:         /* uhoh... thats pretty bad... */
63:         perror("xbee:realloc()");
64:         exit(1);
65:     }
66:     return t;
67: }
68:
69: /* free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
70: static void Xfree2(void **ptr) {
71:     free(*ptr);
72:     *ptr = NULL;
73: }
74:
75: /* ##### */
76: /* ### Helper Functions ##### */
77: /* ##### */
78:
79: /* #####
80:    returns 1 if the packet has data for the digital input else 0 */
81: int xbee_hasdigital(xbee_pkt *pkt, int input) {
82:     int mask = 0x0001;
83:     if (input < 0 || input > 7) return 0;
84:
85:     mask <= input;

```

```

86:     return !(pkt->IOmask & mask);
87: }
88:
89: /* #####
90:  returns 1 if the digital input is high else 0 (or 0 if no digital data present) */
91: int xbee_getdigital(xbee_pkt *pkt, int input) {
92:     int mask = 0x0001;
93:     if (input < 0 || input > 7) return 0;
94:
95:     if (!xbee_hasdigital(pkt,input)) return 0;
96:
97:     mask <= input;
98:     return !(pkt->IOdata & mask);
99: }
100:
101: /* #####
102:  returns 1 if the packet has data for the analog input else 0 */
103: int xbee_hasanalog(xbee_pkt *pkt, int input) {
104:     int mask = 0x0200;
105:     if (input < 0 || input > 5) return 0;
106:
107:     mask <= input;
108:     return !(pkt->IOmask & mask);
109: }
110:
111: /* #####
112:  returns analog input as a voltage if vRef is non-zero, else raw value (or 0 if no analog data present) */
113: double xbee_getanalog(xbee_pkt *pkt, int input, double Vref) {
114:     if (input < 0 || input > 5) return 0;
115:     if (!xbee_hasanalog(pkt,input)) return 0;
116:
117:     if (Vref) return (Vref / 1023) * pkt->IOanalog[input];
118:     return pkt->IOanalog[input];
119: }
120:
121: /* ##### */
122: /* ### XBee Functions ##### */
123: /* ##### */
124:
125: /* #####
126:  xbee_setup
127:  opens xbee serial port & creates xbee listen thread
128:  the xbee must be configured for API mode 2
129:  THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
130: int xbee_setup(char *path, int baudrate) {
131:     return xbee_setuplog(path,baudrate,0);
132: }
133: int xbee_setuplog(char *path, int baudrate, int logfd) {
134:     t_info info;
135:     struct flock fl;
136:     struct termios tc;
137:     speed_t chosenbaud;
138:
139: #ifdef DEBUG
140:     xbee.logfd = ((logfd)?logfd:stdout);
141: #else
142:     xbee.logfd = logfd;
143: #endif
144:     if (xbee.logfd) {
145:         xbee.log = fdopen(xbee.logfd,"w");
146:         if (!xbee.log) {
147:             /* errno == 9 is bad file descriptor (probably not provided) */
148:             if (errno != 9) perror("Failed opening logfile");
149:             xbee.logfd = 0;
150:         }
151:     }
152:
153:     /* select the baud rate */
154:     switch (baudrate) {
155:         case 1200: chosenbaud = B1200; break;
156:         case 2400: chosenbaud = B2400; break;
157:         case 4800: chosenbaud = B4800; break;
158:         case 9600: chosenbaud = B9600; break;
159:         case 19200: chosenbaud = B19200; break;
160:         case 38400: chosenbaud = B38400; break;
161:         case 57600: chosenbaud = B57600; break;
162:         case 115200: chosenbaud = B115200; break;
163:         default:
164:             fprintf(stderr,"XBee: Unknown or incompatiable baud rate specified... (%d)\n",baudrate);
165:             return -1;
166:     };
167:
168:     /* setup the connection mutex */
169:     xbee.conlist = NULL;
170:     if (pthread_mutex_init(&xbee.conmutex,NULL)) {

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```



```

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

```

```

766:
767:  /* unlink this packet from the chain! */
768:  q->next = NULL;
769:
770:  if (xbee.logfd) {
771:      fprintf(xbee.log,"XBee: Got a packet\n");
772:      for (p = xbee.pktlist,c = 0;p;c++,p = p->next);
773:      fprintf(xbee.log,"XBee: Packets left: %d\n",c);
774:  }
775:
776:  /* unlock the packet mutex */
777:  pthread_mutex_unlock(&xbee.pktmutex);
778:
779:  /* and return the packet (must be freed by caller!) */
780:  return q;
781: }
782:
783: /* #####
784:  xbee_matchpktcon - INTERNAL
785:  checks if the packet matches the connection */
786: static int xbee_matchpktcon(xbee_pkt *pkt, xbee_con *con) {
787:  /* if: the connection type matches the packet type OR
788:     the connection is 16/64bit remote AT, and the packet is a remote AT response */
789:  if ((pkt->type == con->type) || /* -- */
790:      ((pkt->type == xbee_remoteAT) && /* -- */
791:       ((con->type == xbee_16bitRemoteAT) ||
792:        (con->type == xbee_64bitRemoteAT)))) {
793:
794:      /* if: the packet is modem status OR
795:         the packet is tx status or AT data and the frame IDs match OR
796:         the addresses match */
797:      if (pkt->type == xbee_modemStatus) return 1;
798:
799:      if ((pkt->type == xbee_txStatus) ||
800:          (pkt->type == xbee_localAT) ||
801:          (pkt->type == xbee_remoteAT)) {
802:          if (pkt->frameID == con->frameID) {
803:              return 1;
804:          }
805:      } else if (pkt->sAddr64 && !memcmp(pkt->Addr64,con->tAddr,8)) {
806:          return 1;
807:      } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16,con->tAddr,2)) {
808:          return 1;
809:      }
810:  }
811:  return 0;
812: }
813:
814: /* #####
815:  xbee_parse_io - INTERNAL
816:  parses the data given into the packet io information */
817: static int xbee_parse_io(xbee_pkt *p, unsigned char *d, int offset, int i) {
818:  /* copy in the I/O data mask */
819:  p->IOMask = (((d[offset]<<8) | d[offset + 1]) & 0x7FFF);
820:
821:  /* copy in the digital I/O data */
822:  p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);
823:
824:  /* advance over the digital data, if its there */
825:  i += ((p->IOMask & 0x01FF)?2:0);
826:
827:  /* copy in the analog I/O data */
828:  if (p->IOMask & 0x0200) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
829:  if (p->IOMask & 0x0400) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
830:  if (p->IOMask & 0x0800) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
831:  if (p->IOMask & 0x1000) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
832:  if (p->IOMask & 0x2000) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
833:  if (p->IOMask & 0x4000) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
834:  if (xbee.logfd) {
835:      if (p->IOMask & 0x0001)
836:          fprintf(xbee.log,"XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
837:      if (p->IOMask & 0x0002)
838:          fprintf(xbee.log,"XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
839:      if (p->IOMask & 0x0004)
840:          fprintf(xbee.log,"XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
841:      if (p->IOMask & 0x0008)
842:          fprintf(xbee.log,"XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
843:      if (p->IOMask & 0x0010)
844:          fprintf(xbee.log,"XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
845:      if (p->IOMask & 0x0020)
846:          fprintf(xbee.log,"XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
847:      if (p->IOMask & 0x0040)
848:          fprintf(xbee.log,"XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
849:      if (p->IOMask & 0x0080)
850:          fprintf(xbee.log,"XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));

```

```

851:     if (p->IOmask & 0x0100)
852:         fprintf(xbee.log,"XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));
853:     if (p->IOmask & 0x0200)
854:         fprintf(xbee.log,"XBee: Analog 0: %d (~%.2fv)\n",p->IOanalog[0],(3.3/1023)*p->IOanalog[0]);
855:     if (p->IOmask & 0x0400)
856:         fprintf(xbee.log,"XBee: Analog 1: %d (~%.2fv)\n",p->IOanalog[1],(3.3/1023)*p->IOanalog[1]);
857:     if (p->IOmask & 0x0800)
858:         fprintf(xbee.log,"XBee: Analog 2: %d (~%.2fv)\n",p->IOanalog[2],(3.3/1023)*p->IOanalog[2]);
859:     if (p->IOmask & 0x1000)
860:         fprintf(xbee.log,"XBee: Analog 3: %d (~%.2fv)\n",p->IOanalog[3],(3.3/1023)*p->IOanalog[3]);
861:     if (p->IOmask & 0x2000)
862:         fprintf(xbee.log,"XBee: Analog 4: %d (~%.2fv)\n",p->IOanalog[4],(3.3/1023)*p->IOanalog[4]);
863:     if (p->IOmask & 0x4000)
864:         fprintf(xbee.log,"XBee: Analog 5: %d (~%.2fv)\n",p->IOanalog[5],(3.3/1023)*p->IOanalog[5]);
865: }
866:
867: return i;
868: }
869:
870: /* #####
871: xbee_listen_wrapper - INTERNAL
872: the xbee_listen wrapper. Prints an error when xbee_listen ends */
873: static void xbee_listen_wrapper(t_info *info) {
874:     int ret;
875:
876:     /* just falls out if the proper 'go-ahead' isn't given */
877:     if (xbee_ready != -1) return;
878:     /* now allow the parent to continue */
879:     xbee_ready = -2;
880:
881:     info->i = -1;
882:     for (;;) {
883:         ret = xbee_listen(info);
884:         if (xbee.logfd) {
885:             fprintf(xbee.log,"XBee: xbee_listen() returned [%d]... Restarting in 250ms!\n",ret);
886:         }
887:         usleep(25000);
888:     }
889: }
890:
891: /* xbee_listen - INTERNAL
892: the xbee xbee_listen thread
893: reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
894: static int xbee_listen(t_info *info) {
895:     unsigned char c, t, d[128];
896:     unsigned int l, i, chksum, o;
897:     int j;
898:     xbee_pkt *p, *q, *po;
899:     xbee_con *con;
900:     int hasCon;
901:
902:     /* just falls out if the proper 'go-ahead' isn't given */
903:     if (info->i != -1) return -1;
904:
905:     /* do this forever :) */
906:     while(1) {
907:         /* wait for a valid start byte */
908:         if (xbee_getRawByte() != 0x7E) continue;
909:
910:         if (xbee.logfd) {
911:             fprintf(xbee.log,"XBee: ---- RX Packet =====--\nXBee: Got a packet!...\n");
912:         }
913:
914:         /* get the length */
915:         l = xbee_getByte() << 8;
916:         l += xbee_getByte();
917:
918:         /* check it is a valid length... */
919:         if (!l) {
920:             if (xbee.logfd) {
921:                 fprintf(xbee.log,"XBee: Recived zero length packet!\n");
922:             }
923:             continue;
924:         }
925:         if (l > 100) {
926:             if (xbee.logfd) {
927:                 fprintf(xbee.log,"XBee: Recived oversized packet! Length: %d\n",l - 1);
928:             }
929:             continue;
930:         }
931:
932:         if (xbee.logfd) {
933:             fprintf(xbee.log,"XBee: Length: %d\n",l - 1);
934:         }
935:

```

```

936:      /* get the packet type */
937:      t = xbee_getByte();
938:
939:      /* start the checksum */
940:      chksum = t;
941:
942:      /* suck in all the data */
943:      for (i = 0; i > 1 && i < 128; i--, i++) {
944:          /* get an unescaped byte */
945:          c = xbee_getByte();
946:          d[i] = c;
947:          chksum += c;
948:          if (xbee.logfd) {
949:              fprintf(xbee.log, "XBee: %3d | 0x%02X | ", i, c);
950:              if ((c > 32) && (c < 127)) fprintf(xbee.log, "%c'\n", c); else fprintf(xbee.log, " _\n");
951:          }
952:      }
953:      i--; /* it went up too many times!... */
954:
955:      /* add the checksum */
956:      chksum += xbee_getByte();
957:
958:      /* check if the whole packet was recieved, or something else occurred... unlikely... */
959:      if (l>1) {
960:          if (xbee.logfd) {
961:              fprintf(xbee.log, "XBee: Didn't get whole packet... :(\n");
962:          }
963:          continue;
964:      }
965:
966:      /* check the checksum */
967:      if ((chksum & 0xFF) != 0xFF) {
968:          if (xbee.logfd) {
969:              fprintf(xbee.log, "XBee: Invalid Checksum: 0x%02X\n", chksum);
970:          }
971:          continue;
972:      }
973:
974:      /* make a new packet */
975:      po = p = Xcalloc(sizeof(xbee_pkt));
976:      q = NULL;
977:      p->datalen = 0;
978:
979:      /* ##### */
980:      /* if: modem status */
981:      if (t == 0x8A) {
982:          if (xbee.logfd) {
983:              fprintf(xbee.log, "XBee: Packet type: Modem Status (0x8A)\n");
984:              fprintf(xbee.log, "XBee: ");
985:              switch (d[0]) {
986:                  case 0x00: fprintf(xbee.log, "Hardware reset"); break;
987:                  case 0x01: fprintf(xbee.log, "Watchdog timer reset"); break;
988:                  case 0x02: fprintf(xbee.log, "Associated"); break;
989:                  case 0x03: fprintf(xbee.log, "Disassociated"); break;
990:                  case 0x04: fprintf(xbee.log, "Synchronization lost"); break;
991:                  case 0x05: fprintf(xbee.log, "Coordinator realignment"); break;
992:                  case 0x06: fprintf(xbee.log, "Coordinator started"); break;
993:              }
994:              fprintf(xbee.log, "... \n");
995:          }
996:          p->type = xbee_modemStatus;
997:
998:          p->sAddr64 = FALSE;
999:          p->dataPkt = FALSE;
1000:          p->txStatusPkt = FALSE;
1001:          p->modemStatusPkt = TRUE;
1002:          p->remoteATPkt = FALSE;
1003:          p->IOPkt = FALSE;
1004:
1005:          /* modem status can only ever give 1 'data' byte */
1006:          p->datalen = 1;
1007:          p->data[0] = d[0];
1008:
1009:          /* ##### */
1010:          /* if: local AT response */
1011:      } else if (t == 0x88) {
1012:          if (xbee.logfd) {
1013:              fprintf(xbee.log, "XBee: Packet type: Local AT Response (0x88)\n");
1014:              fprintf(xbee.log, "XBee: FrameID: 0x%02X\n", d[0]);
1015:              fprintf(xbee.log, "XBee: AT Command: %c%c\n", d[1], d[2]);
1016:              if (d[3] == 0) fprintf(xbee.log, "XBee: Status: OK\n");
1017:              else if (d[3] == 1) fprintf(xbee.log, "XBee: Status: Error\n");
1018:              else if (d[3] == 2) fprintf(xbee.log, "XBee: Status: Invalid Command\n");
1019:              else if (d[3] == 3) fprintf(xbee.log, "XBee: Status: Invalid Parameter\n");
1020:          }

```

```

1021:     p->type = xbee_localAT;
1022:
1023:     p->sAddr64 = FALSE;
1024:     p->dataPkt = FALSE;
1025:     p->txStatusPkt = FALSE;
1026:     p->modemStatusPkt = FALSE;
1027:     p->remoteATPkt = FALSE;
1028:     p->IOPkt = FALSE;
1029:
1030:     p->frameID = d[0];
1031:     p->atCmd[0] = d[1];
1032:     p->atCmd[1] = d[2];
1033:
1034:     p->status = d[3];
1035:
1036:     /* copy in the data */
1037:     p->datalen = i-3;
1038:     for (;i>3;i--) p->data[i-4] = d[i];
1039:
1040:     /* ##### */
1041:     /* if: remote AT response */
1042:     } else if (t == 0x97) {
1043:         if (xbee.logfd) {
1044:             fprintf(xbee.log,"XBee: Packet type: Remote AT Response (0x97)\n");
1045:             fprintf(xbee.log,"XBee: FrameID: 0x%02X\n",d[0]);
1046:             fprintf(xbee.log,"XBee: 64-bit Address: ");
1047:             for (j=0;j<8;j++) {
1048:                 fprintf(xbee.log,(j?"%02X":"%02X"),d[1+j]);
1049:             }
1050:             fprintf(xbee.log,"\n");
1051:             fprintf(xbee.log,"XBee: 16-bit Address: ");
1052:             for (j=0;j<2;j++) {
1053:                 fprintf(xbee.log,(j?"%02X":"%02X"),d[9+j]);
1054:             }
1055:             fprintf(xbee.log,"\n");
1056:             fprintf(xbee.log,"XBee: AT Command: %c%c\n",d[11],d[12]);
1057:             if (d[13] == 0) fprintf(xbee.log,"XBee: Status: OK\n");
1058:             else if (d[13] == 1) fprintf(xbee.log,"XBee: Status: Error\n");
1059:             else if (d[13] == 2) fprintf(xbee.log,"XBee: Status: Invalid Command\n");
1060:             else if (d[13] == 3) fprintf(xbee.log,"XBee: Status: Invalid Parameter\n");
1061:             else if (d[13] == 4) fprintf(xbee.log,"XBee: Status: No Response\n");
1062:         }
1063:         p->type = xbee_remoteAT;
1064:
1065:         p->sAddr64 = FALSE;
1066:         p->dataPkt = FALSE;
1067:         p->txStatusPkt = FALSE;
1068:         p->modemStatusPkt = FALSE;
1069:         p->remoteATPkt = TRUE;
1070:         p->IOPkt = FALSE;
1071:
1072:         p->frameID = d[0];
1073:
1074:         p->Addr64[0] = d[1];
1075:         p->Addr64[1] = d[2];
1076:         p->Addr64[2] = d[3];
1077:         p->Addr64[3] = d[4];
1078:         p->Addr64[4] = d[5];
1079:         p->Addr64[5] = d[6];
1080:         p->Addr64[6] = d[7];
1081:         p->Addr64[7] = d[8];
1082:
1083:         p->Addr16[0] = d[9];
1084:         p->Addr16[1] = d[10];
1085:
1086:         p->atCmd[0] = d[11];
1087:         p->atCmd[1] = d[12];
1088:
1089:         p->status = d[13];
1090:
1091:         if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
1092:             /* parse the io data */
1093:             if (xbee.logfd) fprintf(xbee.log,"XBee: --- Sample ----- \n");
1094:             xbee_parse_io(p, d, 15, 17);
1095:             if (xbee.logfd) fprintf(xbee.log,"XBee: ----- \n");
1096:         } else {
1097:             /* copy in the data */
1098:             p->datalen = i-13;
1099:             for (;i>13;i--) p->data[i-14] = d[i];
1100:         }
1101:
1102:     /* ##### */
1103:     /* if: TX status */
1104:     } else if (t == 0x89) {
1105:         if (xbee.logfd) {

```

```

1106:         fprintf(xbee.log,"XBee: Packet type: TX Status Report (0x89)\n");
1107:         fprintf(xbee.log,"XBee: FrameID: 0x%02X\n",d[0]);
1108:         if (d[1] == 0) fprintf(xbee.log,"XBee: Status: Success\n");
1109:         else if (d[1] == 1) fprintf(xbee.log,"XBee: Status: No ACK\n");
1110:         else if (d[1] == 2) fprintf(xbee.log,"XBee: Status: CCA Failure\n");
1111:         else if (d[1] == 3) fprintf(xbee.log,"XBee: Status: Purged\n");
1112:     }
1113:     p->type = xbee_txStatus;
1114:
1115:     p->sAddr64 = FALSE;
1116:     p->dataPkt = FALSE;
1117:     p->txStatusPkt = TRUE;
1118:     p->modemStatusPkt = FALSE;
1119:     p->remoteATPkt = FALSE;
1120:     p->IOPkt = FALSE;
1121:
1122:     p->frameID = d[0];
1123:
1124:     p->status = d[1];
1125:
1126:     /* never returns data */
1127:     p->datalen = 0;
1128:
1129:     /* ##### */
1130:     /* if: 16 / 64bit data recieve */
1131: } else if ((t == 0x80) ||
1132:           (t == 0x81)) {
1133:     int offset;
1134:     if (t == 0x80) { /* 64bit */
1135:         offset = 8;
1136:     } else { /* 16bit */
1137:
1138:         offset = 2;
1139:     }
1140:     if (xbee.logfd) {
1141:         fprintf(xbee.log,"XBee: Packet type: %d-bit RX Data (0x%02X)\n",((t == 0x80)?64:16),t);
1142:         fprintf(xbee.log,"XBee: %d-bit Address: ",((t == 0x80)?64:16));
1143:         for (j=0;j<offset;j++) {
1144:             fprintf(xbee.log,(j?"%02X":"%02X"),d[j]);
1145:         }
1146:         fprintf(xbee.log,"\n");
1147:         fprintf(xbee.log,"XBee: RSSI: -%ddb\n",d[offset]);
1148:         if (d[offset + 1] & 0x02) fprintf(xbee.log,"XBee: Options: Address Broadcast\n");
1149:         if (d[offset + 1] & 0x03) fprintf(xbee.log,"XBee: Options: PAN Broadcast\n");
1150:     }
1151:     p->dataPkt = TRUE;
1152:     p->txStatusPkt = FALSE;
1153:     p->modemStatusPkt = FALSE;
1154:     p->remoteATPkt = FALSE;
1155:     p->IOPkt = FALSE;
1156:
1157:     if (t == 0x80) { /* 64bit */
1158:         p->type = xbee_64bitData;
1159:
1160:         p->sAddr64 = TRUE;
1161:
1162:         p->Addr64[0] = d[0];
1163:         p->Addr64[1] = d[1];
1164:         p->Addr64[2] = d[2];
1165:         p->Addr64[3] = d[3];
1166:         p->Addr64[4] = d[4];
1167:         p->Addr64[5] = d[5];
1168:         p->Addr64[6] = d[6];
1169:         p->Addr64[7] = d[7];
1170:     } else { /* 16bit */
1171:         p->type = xbee_16bitData;
1172:
1173:         p->sAddr64 = FALSE;
1174:
1175:         p->Addr16[0] = d[0];
1176:         p->Addr16[1] = d[1];
1177:     }
1178:
1179:     /* save the RSSI / signal strength
1180:     this can be used with printf as:
1181:     printf("-%ddb\n",p->RSSI); */
1182:     p->RSSI = d[offset];
1183:
1184:     p->status = d[offset + 1];
1185:
1186:     /* copy in the data */
1187:     p->datalen = i-(offset + 1);
1188:     for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
1189:
1190:     /* ##### */

```

```

1191:      /* if: 16 / 64bit I/O recieve */
1192:    } else if ((t == 0x82) ||
1193:              (t == 0x83)) {
1194:      int offset, samples;
1195:      if (t == 0x82) { /* 64bit */
1196:        offset = 8;
1197:        samples = d[10];
1198:      } else { /* 16bit */
1199:        offset = 2;
1200:        samples = d[4];
1201:      }
1202:      if (xbee.logfd) {
1203:        fprintf(xbee.log, "XBee: Packet type: %d-bit RX I/O Data (0x%02X)\n", ((t == 0x82)?64:16), t);
1204:        fprintf(xbee.log, "XBee: %d-bit Address: ", ((t == 0x82)?64:16));
1205:        for (j = 0; j < offset; j++) {
1206:          fprintf(xbee.log, (j?"%02X":"%02X"), d[j]);
1207:        }
1208:        fprintf(xbee.log, "\n");
1209:        fprintf(xbee.log, "XBee: RSSI: -%ddB\n", d[offset]);
1210:        if (d[9] & 0x02) fprintf(xbee.log, "XBee: Options: Address Broadcast\n");
1211:        if (d[9] & 0x02) fprintf(xbee.log, "XBee: Options: PAN Broadcast\n");
1212:        fprintf(xbee.log, "XBee: Samples: %d\n", d[offset + 2]);
1213:      }
1214:      i = offset + 5;
1215:
1216:      /* each sample is split into its own packet here, for simplicity */
1217:      for (o = samples; o > 0; o--) {
1218:        if (xbee.logfd) {
1219:          fprintf(xbee.log, "XBee: --- Sample %3d ----- \n", o - samples + 1);
1220:        }
1221:        /* if we arent still using the origional packet */
1222:        if (o < samples) {
1223:          /* make a new one and link it up! */
1224:          q = Xcalloc(sizeof(xbee_pkt));
1225:          p->next = q;
1226:          p = q;
1227:        }
1228:
1229:        /* never returns data */
1230:        p->datalen = 0;
1231:
1232:        p->dataPkt = FALSE;
1233:        p->txStatusPkt = FALSE;
1234:        p->modemStatusPkt = FALSE;
1235:        p->remoteATPkt = FALSE;
1236:        p->IOPkt = TRUE;
1237:
1238:        if (t == 0x82) { /* 64bit */
1239:          p->type = xbee_64bitIO;
1240:
1241:          p->sAddr64 = TRUE;
1242:
1243:          p->Addr64[0] = d[0];
1244:          p->Addr64[1] = d[1];
1245:          p->Addr64[2] = d[2];
1246:          p->Addr64[3] = d[3];
1247:          p->Addr64[4] = d[4];
1248:          p->Addr64[5] = d[5];
1249:          p->Addr64[6] = d[6];
1250:          p->Addr64[7] = d[7];
1251:        } else { /* 16bit */
1252:          p->type = xbee_16bitIO;
1253:
1254:          p->sAddr64 = FALSE;
1255:
1256:          p->Addr16[0] = d[0];
1257:          p->Addr16[1] = d[1];
1258:        }
1259:
1260:        /* save the RSSI / signal strength
1261:        this can be used with printf as:
1262:        printf("-%ddB\n", p->RSSI); */
1263:        p->RSSI = d[offset];
1264:
1265:        p->status = d[offset + 1];
1266:
1267:        /* parse the io data */
1268:        i = xbee_parse_io(p, d, offset + 3, i);
1269:      }
1270:      if (xbee.logfd) {
1271:        fprintf(xbee.log, "XBee: ----- \n");
1272:      }
1273:
1274:      /* ##### */
1275:      /* if: Unknown */

```



```

1276:     } else {
1277:         if (xbee.logfd) {
1278:             fprintf(xbee.log,"XBee: Packet type: Unknown (0x%02X)\n",t);
1279:         }
1280:         p->type = xbee_unknown;
1281:     }
1282:     p->next = NULL;
1283:
1284:     /* lock the connection mutex */
1285:     pthread_mutex_lock(&xbee.conmutex);
1286:
1287:     con = xbee.conlist;
1288:     hasCon = 0;
1289:     while (con) {
1290:         if (xbee_matchpktcon(p,con)) {
1291:             hasCon = 1;
1292:             break;
1293:         }
1294:         con = con->next;
1295:     }
1296:
1297:     /* unlock the connection mutex */
1298:     pthread_mutex_unlock(&xbee.conmutex);
1299:
1300:     /* if the packet doesn't have a connection, don't add it! */
1301:     if (!hasCon) {
1302:         Xfree(p);
1303:         if (xbee.logfd) {
1304:             fprintf(xbee.log,"XBee: Connectionless packet... discarding!\n");
1305:         }
1306:         continue;
1307:     }
1308:
1309:     /* lock the packet mutex, so we can safely add the packet to the list */
1310:     pthread_mutex_lock(&xbee.pktmutex);
1311:     i = 1;
1312:     /* if: the list is empty */
1313:     if (!xbee.pktlist) {
1314:         /* start the list! */
1315:         xbee.pktlist = po;
1316:     } else {
1317:         /* add the packet to the end */
1318:         q = xbee.pktlist;
1319:         while (q->next) {
1320:             q = q->next;
1321:             i++;
1322:         }
1323:         q->next = po;
1324:     }
1325:
1326:     if (xbee.logfd) {
1327:         while (q && q->next) {
1328:             q = q->next;
1329:             i++;
1330:         }
1331:         fprintf(xbee.log,"XBee: -----n");
1332:         fprintf(xbee.log,"XBee: Packets: %d\n",i);
1333:     }
1334:
1335:     po = p = q = NULL;
1336:
1337:     /* unlock the packet mutex */
1338:     pthread_mutex_unlock(&xbee.pktmutex);
1339: }
1340: }
1341:
1342: /* #####
1343: xbee_getByte - INTERNAL
1344: waits for an escaped byte of data */
1345: static unsigned char xbee_getByte(void) {
1346:     unsigned char c;
1347:
1348:     ISREADY;
1349:
1350:     /* take a byte */
1351:     c = xbee_getRawByte();
1352:     /* if its escaped, take another and un-escape */
1353:     if (c == 0x7D) c = xbee_getRawByte() ^ 0x20;
1354:
1355:     return (c & 0xFF);
1356: }
1357:
1358: /* #####
1359: xbee_getRawByte - INTERNAL
1360: waits for a raw byte of data */

```



```

1361: static unsigned char xbee_getRawByte(void) {
1362:     unsigned char c;
1363:     fd_set fds;
1364:
1365:     ISREADY;
1366:
1367:     /* wait for a read to be possible */
1368:     FD_ZERO(&fds);
1369:     FD_SET(xbee.ttyfd,&fds);
1370:     if (select(xbee.ttyfd+1,&fds,NULL,NULL,NULL) == -1) {
1371:         perror("xbee:xbee_listen():xbee_getRawByte()");
1372:         exit(1);
1373:     }
1374:
1375:     /* read 1 character
1376:     the loop is just incase there actually isnt a byte there to be read... */
1377:     do {
1378:         if (read(xbee.ttyfd,&c,1) == 0) {
1379:             usleep(10);
1380:             continue;
1381:         }
1382:     } while (0);
1383:
1384:     return (c & 0xFF);
1385: }
1386:
1387: /* #####
1388: xbee_send_pkt - INTERNAL
1389: sends a complete packet of data */
1390: static void xbee_send_pkt(t_data *pkt) {
1391:     ISREADY;
1392:
1393:
1394:     /* lock the send mutex */
1395:     pthread_mutex_lock(&xbee.sendmutex);
1396:
1397:     /* write and flush the data */
1398:     fwrite(pkt->data,pkt->length,1,xbee.tty);
1399:     fflush(xbee.tty);
1400:
1401:     /* unlock the mutex */
1402:     pthread_mutex_unlock(&xbee.sendmutex);
1403:
1404:     if (xbee.logfd) {
1405:         int i;
1406:         /* prints packet in hex byte-by-byte */
1407:         fprintf(xbee.log,"XBee: TX Packet - ");
1408:         for (i=0;i<pkt->length;i++) {
1409:             fprintf(xbee.log,"%0x%02X ",pkt->data[i]);
1410:         }
1411:         fprintf(xbee.log,"\n");
1412:     }
1413:
1414:     /* free the packet */
1415:     Xfree(pkt);
1416: }
1417:
1418: /* #####
1419: xbee_make_pkt - INTERNAL
1420: adds delimiter field
1421: calculates length and checksum
1422: escapes bytes */
1423: static t_data *xbee_make_pkt(unsigned char *data, int length) {
1424:     t_data *pkt;
1425:     unsigned int l, i, o, t, x, m;
1426:     char d = 0;
1427:
1428:     ISREADY;
1429:
1430:     /* check the data given isnt too long
1431:     100 bytes maximum payload + 12 bytes header information */
1432:     if (length > 100 + 12) return NULL;
1433:
1434:     /* calculate the length of the whole packet
1435:     start, length (MSB), length (LSB), DATA, checksum */
1436:     l = 3 + length + 1;
1437:
1438:     /* prepare memory */
1439:     pkt = Xcalloc(sizeof(t_data));
1440:
1441:     /* put start byte on */
1442:     pkt->data[0] = 0x7E;
1443:
1444:     /* copy data into packet */
1445:     for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {

```

```
1446:      /* if: its time for the checksum */
1447:      if (i == length) d = M8((0xFF - M8(t)));
1448:      /* if: its time for the high length byte */
1449:      else if (m == 1) d = M8(length >> 8);
1450:      /* if: its time for the low length byte */
1451:      else if (m == 2) d = M8(length);
1452:      /* if: its time for the normal data */
1453:      else if (m > 2) d = data[i];
1454:
1455:      x = 0;
1456:      /* check for any escapes needed */
1457:      if ((d == 0x11) || /* XON */
1458:          (d == 0x13) || /* XOFF */
1459:          (d == 0x7D) || /* Escape */
1460:          (d == 0x7E)) { /* Frame Delimiter */
1461:          l++;
1462:          pkt->data[o++] = 0x7D;
1463:          x = 1;
1464:      }
1465:
1466:      /* move data in */
1467:      pkt->data[o] = ((!x)?d:d^0x20);
1468:      if (m > 2) {
1469:          i++;
1470:          t += d;
1471:      }
1472:  }
1473:
1474:  /* remember the length */
1475:  pkt->length = l;
1476:
1477:  return pkt;
1478: }
```