```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 read 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:   t_info info;
 134:   struct flock fl;
 135:   struct termios tc;
 136:   speed_t chosenbaud;
 137:
 138:   /* select the baud rate */
 139:   switch (baudrate) {
 140:     case 1200:  chosenbaud = B1200;   break;
 141:     case 2400:  chosenbaud = B2400;   break;
 142:     case 4800:  chosenbaud = B4800;   break;
 143:     case 9600:  chosenbaud = B9600;   break;
 144:     case 19200: chosenbaud = B19200;  break;
 145:     case 38400: chosenbaud = B38400;  break;
 146:     case 57600: chosenbaud = B57600;  break;
 147:     case 115200:chosenbaud = B115200; break;
 148:     default:
 149:       fprintf(stderr,"XBee: Unknown or incompatiable baud rate specified... (%d)\n",baudrate);
 150:       return -1;
 151:   };
 152:
 153:   /* setup the connection mutex */
 154:   xbee.conlist = NULL;
 155:   if (pthread_mutex_init(&xbee.conmutex,NULL)) {
 156:     perror("xbee_setup():pthread_mutex_init(conmutex)");
 157:     return -1;
 158:   }
 159:
 160:   /* setup the packet mutex */
 161:   xbee.pktlist = NULL;
 162:   if (pthread_mutex_init(&xbee.pktmutex,NULL)) {
 163:     perror("xbee_setup():pthread_mutex_init(pktmutex)");
 164:     return -1;
 165:   }
 166:
 167:   /* setup the send mutex */
 168:   if (pthread_mutex_init(&xbee.sendmutex,NULL)) {
 169:     perror("xbee_setup():pthread_mutex_init(sendmutex)");
 170:     return -1;
```

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

```
256:     fclose(xbee.tty);
257:     close(xbee.ttyfd);
258:     xbee.ttyfd = -1;
259:     xbee.tty = NULL;
260:     return -1;
261:   }
262:
263:   /* allow other functions to be used! */
264:   xbee_ready = 1;
265:
266:   /* make a txStatus connection */
267:   xbee.con_txStatus = xbee_newcon('*',xbee_txStatus);
268:
269:   return 0;
270: }
271:
272: /* ################################################################
273:    xbee_con
274:    produces a connection to the specified device and frameID
275:    if a connection had already been made, then this connection will be returned */
276: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
277:   xbee_con *con, *ocon;
278:   unsigned char tAddr[8];
279:   va_list ap;
280:   int t;
281: #ifdef DEBUG
282:   int i;
283: #endif
284:
285:   ISREADY;
286:
287:   if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
288:   else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
289:
290:   va_start(ap,type);
291:   /* if: 64 bit address expected (2 ints) */
292:   if ((type == xbee_64bitRemoteAT) ||
293:       (type == xbee_64bitData) ||
294:       (type == xbee_64bitIO)) {
295:     t = va_arg(ap, int);
296:     tAddr[0] = (t >> 24) & 0xFF;
297:     tAddr[1] = (t >> 16) & 0xFF;
298:     tAddr[2] = (t >>  8) & 0xFF;
299:     tAddr[3] = (t      ) & 0xFF;
300:     t = va_arg(ap, int);
301:     tAddr[4] = (t >> 24) & 0xFF;
302:     tAddr[5] = (t >> 16) & 0xFF;
303:     tAddr[6] = (t >>  8) & 0xFF;
304:     tAddr[7] = (t      ) & 0xFF;
305:
306:   /* if: 16 bit address expected (1 int) */
307:   } else if ((type == xbee_16bitRemoteAT) ||
308:              (type == xbee_16bitData) ||
309:              (type == xbee_16bitIO)) {
310:     t = va_arg(ap, int);
311:     tAddr[0] = (t >>  8) & 0xFF;
312:     tAddr[1] = (t      ) & 0xFF;
313:     tAddr[2] = 0;
314:     tAddr[3] = 0;
315:     tAddr[4] = 0;
316:     tAddr[5] = 0;
317:     tAddr[6] = 0;
318:     tAddr[7] = 0;
319:
320:   /* otherwise clear the address */
321:   } else {
322:     memset(tAddr,0,8);
323:   }
324:   va_end(ap);
325:
326:   /* lock the connection mutex */
327:   pthread_mutex_lock(&xbee.conmutex);
328:
329:   /* are there any connections? */
330:   if (xbee.conlist) {
331:     con = xbee.conlist;
332:     while (con) {
333:       /* if: after a modemStatus, and the types match! */
334:       if ((type == xbee_modemStatus) &&
335:           (con->type == type)) {
336:         pthread_mutex_unlock(&xbee.conmutex);
337:         return con;
338:
339:       /* if: after a txStatus and frameIDs match! */
340:       } else if ((type == xbee_txStatus) &&
```

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

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

```
511:
512:   ISREADY;
513:
514:   /* xbee_vsenddata() wants a va_list... */
515:   va_start(ap, format);
516:   /* hand it over :) */
517:   p = xbee_vsenddata(con,format,ap);
518:   va_end(ap);
519:   return p;
520: }
521:
522: xbee_pkt *xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
523:   t_data *pkt;
524:   int i, length;
525:   unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
526:   unsigned char data[128]; /* ditto */
527:   xbee_pkt *p = NULL; /* response packet */
528:   int to = 100; /* resonse timeout */
529:
530:   ISREADY;
531:
532:   if (!con) return (void *)-1;
533:   if (con->type == xbee_unknown) return (void *)-1;
534:
535:   /* make up the data and keep the length, its possible there are nulls in there */
536:   length = vsnprintf((char *)data,128,format,ap);
537:
538: #ifdef DEBUG
539:   fprintf(stderr,"XBee: --== TX Packet ============--\n");
540:   fprintf(stderr,"XBee: Length: %d\n",length);
541:   for (i=0;i<length;i++) {
542:     fprintf(stderr,"XBee: %3d | 0x%02X ",i,data[i]);
543:     if ((data[i] > 32) && (data[i] < 127)) fprintf(stderr,"'%c'\n",data[i]); else fprintf(stderr," _\n");
544:   }
545: #endif
546:
547:   /* ######################################### */
548:   /* if: local AT */
549:   if (con->type == xbee_localAT) {
550:     /* AT commands are 2 chars long (plus optional parameter) */
551:     if (length < 2) return (void *)-1;
552:
553:     /* use the command? */
554:     buf[0] = ((!con->atQueue)?0x08:0x09);
555:     buf[1] = con->frameID;
556:
557:     /* copy in the data */
558:     for (i=0;i<length;i++) {
559:       buf[i+2] = data[i];
560:     }
561:
562:     /* setup the packet */
563:     pkt = xbee_make_pkt(buf,i+2);
564:     /* send it on */
565:     xbee_send_pkt(pkt);
566:
567:     /* wait for a response packet */
568:     for (; p == NULL && to > 0; to--) {
569:       usleep(25400); /* tuned so that hopefully the first time round will catch the response */
570:       p = xbee_getpacket(con);
571:     }
572:
573:     /* if: no txStatus packet was recieved */
574:     if (to == 0) {
575: #ifdef DEBUG
576:       fprintf(stderr,"XBee: No AT status recieved before timeout\n");
577: #endif
578:       return NULL;
579:     }
580:
581: #ifdef DEBUG
582:     switch (p->status) {
583:     case 0x00: fprintf(stderr,"XBee: AT Status: OK!\n");                break;
584:     case 0x01: fprintf(stderr,"XBee: AT Status: Error\n");              break;
585:     case 0x02: fprintf(stderr,"XBee: AT Status: Invalid Command\n");    break;
586:     case 0x03: fprintf(stderr,"XBee: AT Status: Invalid Parameter\n"); break;
587:     }
588: #endif
589:     return p;
590:   /* ######################################### */
591:   /* if: remote AT */
592:   } else if ((con->type == xbee_16bitRemoteAT) ||
593:              (con->type == xbee_64bitRemoteAT)) {
594:     if (length < 2) return (void *)-1; /* at commands are 2 chars long (plus optional parameter) */
595:     buf[0] = 0x17;
```

```
596:     buf[1] = con->frameID;
597:
598:     /* copy in the relevant address */
599:     if (con->tAddr64) {
600:       memcpy(&buf[2],con->tAddr,8);
601:       buf[10] = 0xFF;
602:       buf[11] = 0xFE;
603:     } else {
604:       memset(&buf[2],0,8);
605:       memcpy(&buf[10],con->tAddr,2);
606:     }
607:     /* queue the command? */
608:     buf[12] = ((!con->atQueue)?0x02:0x00);
609:
610:     /* copy in the data */
611:     for (i=0;i<length;i++) {
612:       buf[i+13] = data[i];
613:     }
614:
615:     /* setup the packet */
616:     pkt = xbee_make_pkt(buf,i+13);
617:     /* send it on */
618:     xbee_send_pkt(pkt);
619:
620:     /* wait for a response packet */
621:     for (; p == NULL && to > 0; to--) {
622:       usleep(25400); /* tuned so that hopefully the first time round will catch the response */
623:       p = xbee_getpacket(con);
624:     }
625:
626:     /* if: no txStatus packet was recieved */
627:     if (to == 0) {
628: #ifdef DEBUG
629:       fprintf(stderr,"XBee: No AT status recieved before timeout\n");
630: #endif
631:       return NULL;
632:     }
633:
634: #ifdef DEBUG
635:     switch (p->status) {
636:     case 0x00: fprintf(stderr,"XBee: AT Status: OK!\n");               break;
637:     case 0x01: fprintf(stderr,"XBee: AT Status: Error\n");             break;
638:     case 0x02: fprintf(stderr,"XBee: AT Status: Invalid Command\n");   break;
639:     case 0x03: fprintf(stderr,"XBee: AT Status: Invalid Parameter\n"); break;
640:     case 0x04: fprintf(stderr,"XBee: AT Status: No Response\n");       break;
641:     }
642: #endif
643:     return p;
644:   /* ######################################### */
645:   /* if: 16 or 64bit Data */
646:   } else if ((con->type == xbee_16bitData) ||
647:              (con->type == xbee_64bitData)) {
648:     int offset;
649:
650:     /* if: 16bit Data */
651:     if (con->type == xbee_16bitData) {
652:       buf[0] = 0x01;
653:       offset = 5;
654:       /* copy in the address */
655:       memcpy(&buf[2],con->tAddr,2);
656:
657:     /* if: 64bit Data */
658:     } else { /* 64bit Data */
659:       buf[0] = 0x00;
660:       offset = 11;
661:       /* copy in the address */
662:       memcpy(&buf[2],con->tAddr,8);
663:     }
664:
665:     /* copy frameID */
666:     buf[1] = con->frameID;
667:
668:     /* disable ack? broadcast? */
669:     buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
670:
671:     /* copy in the data */
672:     for (i=0;i<length;i++) {
673:       buf[i+offset] = data[i];
674:     }
675:
676:     /* setup the packet */
677:     pkt = xbee_make_pkt(buf,i+offset);
678:     /* send it on */
679:     xbee_send_pkt(pkt);
680:
```

```
681:     /* wait for a response packet */
682:     for (; p == NULL && to > 0; to--) {
683:       usleep(25400); /* tuned so that hopefully the first time round will catch the response */
684:       p = xbee_getpacket(xbee.con_txStatus);
685:     }
686:
687:     /* if: no txStatus packet was recieved */
688:     if (to == 0) {
689: #ifdef DEBUG
690:       fprintf(stderr,"XBee: No txStatus recieved before timeout\n");
691: #endif
692:       return NULL;
693:     }
694:
695: #ifdef DEBUG
696:     switch (p->status) {
697:     case 0x00: fprintf(stderr,"XBee: txStatus: Success!\n");    break;
698:     case 0x01: fprintf(stderr,"XBee: txStatus: No ACK\n");      break;
699:     case 0x02: fprintf(stderr,"XBee: txStatus: CCA Failure\n"); break;
700:     case 0x03: fprintf(stderr,"XBee: txStatus: Purged\n");      break;
701:     }
702: #endif
703:     /* return the packet */
704:     return p;
705:   /* ######################################### */
706:   /* if: I/O */
707:   } else if ((con->type == xbee_64bitIO) ||
708:              (con->type == xbee_16bitIO)) {
709:     /* not currently implemented... is it even allowed? */
710:     fprintf(stderr,"******* TODO ********\n");
711:   }
712:
713:   return (void *)-1;
714: }
715:
716: /* ########################################################################
717:    xbee_getpacket
718:    retrieves the next packet destined for the given connection
719:    once the packet has been retrieved, it is removed for the list! */
720: xbee_pkt *xbee_getpacket(xbee_con *con) {
721:   xbee_pkt *l, *p, *q;
722: #ifdef DEBUG
723:   int c;
724:   fprintf(stderr,"XBee: --== Get Packet ==========--\n");
725: #endif
726:
727:   /* lock the packet mutex */
728:   pthread_mutex_lock(&xbee.pktmutex);
729:
730:   /* if: there are no packets */
731:   if ((p = xbee.pktlist) == NULL) {
732:     pthread_mutex_unlock(&xbee.pktmutex);
733: #ifdef DEBUG
734:     fprintf(stderr,"XBee: No packets avaliable...\n");
735: #endif
736:     return NULL;
737:   }
738:
739:   l = NULL;
740:   q = NULL;
741:   /* get the first avaliable packet for this connection */
742:   do {
743:     /* does the packet match the connection? */
744:     if (xbee_matchpktcon(p,con)) {
745:       q = p;
746:       break;
747:     }
748:     /* move on */
749:     l = p;
750:     p = p->next;
751:   } while (p);
752:
753:   /* if: no packet was found */
754:   if (!q) {
755:     pthread_mutex_unlock(&xbee.pktmutex);
756: #ifdef DEBUG
757:     fprintf(stderr,"XBee: No packets avaliable (for connection)...\n");
758: #endif
759:     return NULL;
760:   }
761:
762:   /* if it was the first packet */
763:   if (!l) {
764:     /* move the chain along */
765:     xbee.pktlist = p->next;
```

```
766:   } else {
767:     /* otherwise relink the list */
768:     l->next = p->next;
769:   }
770:
771:   /* unlink this packet from the chain! */
772:   q->next = NULL;
773:
774: #ifdef DEBUG
775:   fprintf(stderr,"XBee: Got a packet\n");
776:   for (p = xbee.pktlist,c = 0;p;c++,p = p->next);
777:   fprintf(stderr,"XBee: Packets left: %d\n",c);
778: #endif
779:
780:   /* unlock the packet mutex */
781:   pthread_mutex_unlock(&xbee.pktmutex);
782:
783:   /* and return the packet (must be freed by caller!) */
784:   return q;
785: }
786:
787: /* ###################################################################
788:    xbee_matchpktcon - INTERNAL
789:    checks if the packet matches the connection */
790: int xbee_matchpktcon(xbee_pkt *pkt, xbee_con *con) {
791:   /* if: the connection type matches the packet type OR
792:      the connection is 16/64bit remote AT, and the packet is a remote AT response */
793:   if ((pkt->type == con->type) || /* -- */
794:       ((pkt->type == xbee_remoteAT) && /* -- */
795:        ((con->type == xbee_16bitRemoteAT) ||
796:         (con->type == xbee_64bitRemoteAT)))) {
797:     /* if: the packet is modem status OR
798:        the packet is tx status or AT data and the frame IDs match OR
799:        the addresses match */
800:     if ((pkt->type == xbee_modemStatus) ||
801:         (((pkt->type == xbee_txStatus) ||
802:           (pkt->type == xbee_localAT) ||
803:           (pkt->type == xbee_remoteAT)) &&
804:          (pkt->frameID == con->frameID)) ||
805:         (!memcmp(pkt->Addr64,con->tAddr,8))) {
806:       return 1;
807:     }
808:   }
809:   return 0;
810: }
811:
812: /* ###################################################################
813:    xbee_listen - INTERNAL
814:    the xbee xbee_listen thread
815:    reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
816: void xbee_listen(t_info *info) {
817:   unsigned char c, t, d[128];
818:   unsigned int l, i, chksum, o;
819: #ifdef DEBUG
820:   int j;
821: #endif
822:   xbee_pkt *p, *q, *po;
823:   xbee_con *con;
824:   int hasCon;
825:
826:   /* just falls out if the proper 'go-ahead' isn't given */
827:   if (xbee_ready != -1) return;
828:
829:   /* do this forever :) */
830:   while(1) {
831:     /* wait for a valid start byte */
832:     if (xbee_getRawByte() != 0x7E) continue;
833:
834: #ifdef DEBUG
835:     fprintf(stderr,"XBee: --== RX Packet ===========--\nXBee: Got a packet!...\n");
836: #endif
837:
838:     /* get the length */
839:     l = xbee_getByte() << 8;
840:     l += xbee_getByte();
841:
842:     /* check it is a valid length... */
843:     if (!l) {
844: #ifdef DEBUG
845:       fprintf(stderr,"XBee: Recived zero length packet!\n");
846: #endif
847:       continue;
848:     }
849:     if (l > 100) {
850: #ifdef DEBUG
```

```
851:       fprintf(stderr,"XBee: Recived oversized packet! Length: %d\n",l - 1);
852: #endif
853:       continue;
854:     }
855:
856: #ifdef DEBUG
857:     fprintf(stderr,"XBee: Length: %d\n",l - 1);
858: #endif
859:
860:     /* get the packet type */
861:     t = xbee_getByte();
862:
863:     /* start the checksum */
864:     chksum = t;
865:
866:     /* suck in all the data */
867:     for (i = 0; l > 1 && i < 128; l--, i++) {
868:       /* get an unescaped byte */
869:       c = xbee_getByte();
870:       d[i] = c;
871:       chksum += c;
872: #ifdef DEBUG
873:       fprintf(stderr,"XBee: %3d | 0x%02X | ",i,c);
874:       if ((c > 32) && (c < 127)) fprintf(stderr,"'%c'\n",c); else fprintf(stderr," _\n");
875: #endif
876:     }
877:     i--; /* it went up too many times!... */
878:
879:     /* add the checksum */
880:     chksum += xbee_getByte();
881:
882:     /* check if the whole packet was recieved, or something else occured... unlikely... */
883:     if (l>1) {
884: #ifdef DEBUG
885:       fprintf(stderr,"XBee: Didn't get whole packet... :(\n");
886: #endif
887:       continue;
888:     }
889:
890:     /* check the checksum */
891:     if ((chksum & 0xFF) != 0xFF) {
892: #ifdef DEBUG
893:       fprintf(stderr,"XBee: Invalid Checksum: 0x%02X\n",chksum);
894: #endif
895:       continue;
896:     }
897:
898:     /* make a new packet */
899:     po = p = Xcalloc(sizeof(xbee_pkt));
900:     q = NULL;
901:     p->datalen = 0;
902:
903:     /* ########################################### */
904:     /* if: modem status */
905:     if (t == 0x8A) {
906: #ifdef DEBUG
907:       fprintf(stderr,"XBee: Packet type: Modem Status (0x8A)\n");
908:       fprintf(stderr,"XBee: ");
909:       switch (d[0]) {
910:       case 0x00: fprintf(stderr,"Hardware reset"); break;
911:       case 0x01: fprintf(stderr,"Watchdog timer reset"); break;
912:       case 0x02: fprintf(stderr,"Associated"); break;
913:       case 0x03: fprintf(stderr,"Disassociated"); break;
914:       case 0x04: fprintf(stderr,"Synchronization lost"); break;
915:       case 0x05: fprintf(stderr,"Coordinator realignment"); break;
916:       case 0x06: fprintf(stderr,"Coordinator started"); break;
917:       }
918:       fprintf(stderr,"...\n");
919: #endif
920:       p->type = xbee_modemStatus;
921:
922:       p->sAddr64 = FALSE;
923:       p->dataPkt = FALSE;
924:       p->txStatusPkt = FALSE;
925:       p->modemStatusPkt = TRUE;
926:       p->remoteATPkt = FALSE;
927:       p->IOPkt = FALSE;
928:
929:       /* modem status can only ever give 1 'data' byte */
930:       p->datalen = 1;
931:       p->data[0] = d[0];
932:
933:     /* ########################################### */
934:     /* if: local AT response */
935:     } else if (t == 0x88) {
```

```
936: #ifdef DEBUG
937:       fprintf(stderr,"XBee: Packet type: Local AT Response (0x88)\n");
938:       fprintf(stderr,"XBee: FrameID: 0x%02X\n",d[0]);
939:       fprintf(stderr,"XBee: AT Command: %c%c\n",d[1],d[2]);
940:       if (d[3] == 0) fprintf(stderr,"XBee: Status: OK\n");
941:       else if (d[3] == 1) fprintf(stderr,"XBee: Status: Error\n");
942:       else if (d[3] == 2) fprintf(stderr,"XBee: Status: Invalid Command\n");
943:       else if (d[3] == 3) fprintf(stderr,"XBee: Status: Invalid Parameter\n");
944: #endif
945:       p->type = xbee_localAT;
946:
947:       p->sAddr64 = FALSE;
948:       p->dataPkt = FALSE;
949:       p->txStatusPkt = FALSE;
950:       p->modemStatusPkt = FALSE;
951:       p->remoteATPkt = FALSE;
952:       p->IOPkt = FALSE;
953:
954:       p->frameID = d[0];
955:       p->atCmd[0] = d[1];
956:       p->atCmd[1] = d[2];
957:
958:       p->status = d[3];
959:
960:       /* copy in the data */
961:       p->datalen = i-3;
962:       for (;i>3;i--) p->data[i-4] = d[i];
963:
964:     /* ######################################### */
965:     /* if: remote AT response */
966:     } else if (t == 0x97) {
967: #ifdef DEBUG
968:       fprintf(stderr,"XBee: Packet type: Remote AT Response (0x97)\n");
969:       fprintf(stderr,"XBee: FrameID: 0x%02X\n",d[0]);
970:       fprintf(stderr,"XBee: 64-bit Address: ");
971:       for (j=0;j<8;j++) {
972:         fprintf(stderr,(j?":%02X":"%02X"),d[1+j]);
973:       }
974:       fprintf(stderr,"\n");
975:       fprintf(stderr,"XBee: 16-bit Address: ");
976:       for (j=0;j<2;j++) {
977:         fprintf(stderr,(j?":%02X":"%02X"),d[9+j]);
978:       }
979:       fprintf(stderr,"\n");
980:       fprintf(stderr,"XBee: AT Command: %c%c\n",d[11],d[12]);
981:       if (d[13] == 0) fprintf(stderr,"XBee: Status: OK\n");
982:       else if (d[13] == 1) fprintf(stderr,"XBee: Status: Error\n");
983:       else if (d[13] == 2) fprintf(stderr,"XBee: Status: Invalid Command\n");
984:       else if (d[13] == 3) fprintf(stderr,"XBee: Status: Invalid Parameter\n");
985:       else if (d[13] == 4) fprintf(stderr,"XBee: Status: No Response\n");
986: #endif
987:       p->type = xbee_remoteAT;
988:
989:       p->sAddr64 = FALSE;
990:       p->dataPkt = FALSE;
991:       p->txStatusPkt = FALSE;
992:       p->modemStatusPkt = FALSE;
993:       p->remoteATPkt = TRUE;
994:       p->IOPkt = FALSE;
995:
996:       p->frameID = d[0];
997:
998:       p->Addr64[0] = d[1];
999:       p->Addr64[1] = d[2];
1000:       p->Addr64[2] = d[3];
1001:       p->Addr64[3] = d[4];
1002:       p->Addr64[4] = d[5];
1003:       p->Addr64[5] = d[6];
1004:       p->Addr64[6] = d[7];
1005:       p->Addr64[7] = d[8];
1006:
1007:       p->Addr16[0] = d[9];
1008:       p->Addr16[1] = d[10];
1009:
1010:       p->atCmd[0] = d[11];
1011:       p->atCmd[1] = d[12];
1012:
1013:       p->status = d[13];
1014:
1015:       /* copy in the data */
1016:       p->datalen = i-13;
1017:       for (;i>13;i--) p->data[i-14] = d[i];
1018:
1019:     /* ######################################### */
1020:     /* if: TX status */
```

```
1021:     } else if (t == 0x89) {
1022: #ifdef DEBUG
1023:       fprintf(stderr,"XBee: Packet type: TX Status Report (0x89)\n");
1024:       fprintf(stderr,"XBee: FrameID: 0x%02X\n",d[0]);
1025:       if (d[1] == 0) fprintf(stderr,"XBee: Status: Success\n");
1026:       else if (d[1] == 1) fprintf(stderr,"XBee: Status: No ACK\n");
1027:       else if (d[1] == 2) fprintf(stderr,"XBee: Status: CCA Failure\n");
1028:       else if (d[1] == 3) fprintf(stderr,"XBee: Status: Purged\n");
1029: #endif
1030:       p->type = xbee_txStatus;
1031:
1032:       p->sAddr64 = FALSE;
1033:       p->dataPkt = FALSE;
1034:       p->txStatusPkt = TRUE;
1035:       p->modemStatusPkt = FALSE;
1036:       p->remoteATPkt = FALSE;
1037:       p->IOPkt = FALSE;
1038:
1039:       p->frameID = d[0];
1040:
1041:       p->status = d[1];
1042:
1043:       /* never returns data */
1044:         p->datalen = 0;
1045:
1046:     /* ########################################### */
1047:     /* if: 16 / 64bit data recieve */
1048:     } else if ((t == 0x80) ||
1049:                (t == 0x81)) {
1050:       int offset;
1051:       if (t == 0x80) { /* 64bit */
1052:         offset = 8;
1053:       } else { /* 16bit */
1054:         offset = 2;
1055:       }
1056: #ifdef DEBUG
1057:       fprintf(stderr,"XBee: Packet type: %d-bit RX Data (0x%02X)\n",((t == 0x80)?64:16),t);
1058:       fprintf(stderr,"XBee: %d-bit Address: ",((t == 0x80)?64:16));
1059:       for (j=0;j<offset;j++) {
1060:         fprintf(stderr,(j?":%02X":"%02X"),d[j]);
1061:       }
1062:       fprintf(stderr,"\n");
1063:       fprintf(stderr,"XBee: RSSI: -%ddB\n",d[offset]);
1064:       if (d[offset + 1] & 0x02) fprintf(stderr,"XBee: Options: Address Broadcast\n");
1065:       if (d[offset + 1] & 0x03) fprintf(stderr,"XBee: Options: PAN Broadcast\n");
1066: #endif
1067:       p->dataPkt = TRUE;
1068:       p->txStatusPkt = FALSE;
1069:       p->modemStatusPkt = FALSE;
1070:       p->remoteATPkt = FALSE;
1071:       p->IOPkt = FALSE;
1072:
1073:       if (t == 0x82) { /* 64bit */
1074:         p->type = xbee_64bitData;
1075:
1076:         p->sAddr64 = TRUE;
1077:
1078:         p->Addr64[0] = d[0];
1079:         p->Addr64[1] = d[1];
1080:         p->Addr64[2] = d[2];
1081:         p->Addr64[3] = d[3];
1082:         p->Addr64[4] = d[4];
1083:         p->Addr64[5] = d[5];
1084:         p->Addr64[6] = d[6];
1085:         p->Addr64[7] = d[7];
1086:       } else { /* 16bit */
1087:         p->type = xbee_16bitData;
1088:
1089:         p->sAddr64 = FALSE;
1090:
1091:         p->Addr16[0] = d[0];
1092:         p->Addr16[1] = d[1];
1093:       }
1094:
1095:       /* save the RSSI / signal strength
1096:          this can be used with printf as:
1097:          printf("-%ddB\n",p->RSSI); */
1098:       p->RSSI = d[offset];
1099:
1100:       p->status = d[offset + 1];
1101:
1102:       /* copy in the data */
1103:       p->datalen = i-(offset + 1);
1104:       for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
1105:
```

```
1106:     /* ######################################### */
1107:     /* if: 16 / 64bit I/O recieve */
1108:     } else if ((t == 0x82) ||
1109:                (t == 0x83)) {
1110:       int offset, samples;
1111:       if (t == 0x82) { /* 64bit */
1112:         offset = 8;
1113:         samples = d[10];
1114:       } else { /* 16bit */
1115:         offset = 2;
1116:         samples = d[4];
1117:       }
1118: #ifdef DEBUG
1119:       fprintf(stderr,"XBee: Packet type: %d-bit RX I/O Data (0x%02X)\n",((t == 0x82)?64:16),t);
1120:       fprintf(stderr,"XBee: %d-bit Address: ",((t == 0x82)?64:16));
1121:       for (j = 0; j < offset; j++) {
1122:         fprintf(stderr,(j?":%02X":"%02X"),d[j]);
1123:       }
1124:       fprintf(stderr,"\n");
1125:       fprintf(stderr,"XBee: RSSI: -%ddB\n",d[offset]);
1126:       if (d[9] & 0x02) fprintf(stderr,"XBee: Options: Address Broadcast\n");
1127:       if (d[9] & 0x02) fprintf(stderr,"XBee: Options: PAN Broadcast\n");
1128:       fprintf(stderr,"XBee: Samples: %d\n",d[offset + 2]);
1129: #endif
1130:       i = offset + 5;
1131:
1132:       /* each sample is split into its own packet here, for simplicity */
1133:       for (o = samples; o > 0; o--) {
1134: #ifdef DEBUG
1135:         fprintf(stderr,"XBee: --- Sample %3d ------------\n", o - samples + 1);
1136: #endif
1137:         /* if we arent still using the origional packet */
1138:         if (o < samples) {
1139:           /* make a new one and link it up! */
1140:           q = Xcalloc(sizeof(xbee_pkt));
1141:           p->next = q;
1142:           p = q;
1143:         }
1144:
1145:         /* never returns data */
1146:         p->datalen = 0;
1147:
1148:         p->dataPkt = FALSE;
1149:         p->txStatusPkt = FALSE;
1150:         p->modemStatusPkt = FALSE;
1151:         p->remoteATPkt = FALSE;
1152:         p->IOPkt = TRUE;
1153:
1154:         if (t == 0x82) { /* 64bit */
1155:           p->type = xbee_64bitIO;
1156:
1157:           p->sAddr64 = TRUE;
1158:
1159:           p->Addr64[0] = d[0];
1160:           p->Addr64[1] = d[1];
1161:           p->Addr64[2] = d[2];
1162:           p->Addr64[3] = d[3];
1163:           p->Addr64[4] = d[4];
1164:           p->Addr64[5] = d[5];
1165:           p->Addr64[6] = d[6];
1166:           p->Addr64[7] = d[7];
1167:         } else { /* 16bit */
1168:           p->type = xbee_16bitIO;
1169:
1170:           p->sAddr64 = FALSE;
1171:
1172:           p->Addr16[0] = d[0];
1173:           p->Addr16[1] = d[1];
1174:         }
1175:
1176:         /* save the RSSI / signal strength
1177:            this can be used with printf as:
1178:            printf("-%ddB\n",p->RSSI); */
1179:         p->RSSI = d[offset];
1180:
1181:         p->status = d[offset + 1];
1182:
1183:         /* copy in the I/O data mask */
1184:         p->IOmask = (((d[offset + 3]<<8) | d[offset + 4]) & 0x7FFF);
1185:
1186:         /* copy in the digital I/O data */
1187:         p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);
1188:
1189:         /* advance over the digital data, if its there */
1190:         i += (((d[offset + 3]&0x01)||(d[offset + 4]))?2:0);
```

```
1191:
1192:         /* copy in the analog I/O data */
1193:         if (d[11]&0x02) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1194:         if (d[11]&0x04) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1195:         if (d[11]&0x08) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1196:         if (d[11]&0x10) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1197:         if (d[11]&0x20) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1198:         if (d[11]&0x40) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1199: #ifdef DEBUG
1200:         if (p->IOmask & 0x0001) fprintf(stderr,"XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
1201:         if (p->IOmask & 0x0002) fprintf(stderr,"XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
1202:         if (p->IOmask & 0x0004) fprintf(stderr,"XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
1203:         if (p->IOmask & 0x0008) fprintf(stderr,"XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
1204:         if (p->IOmask & 0x0010) fprintf(stderr,"XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
1205:         if (p->IOmask & 0x0020) fprintf(stderr,"XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
1206:         if (p->IOmask & 0x0040) fprintf(stderr,"XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
1207:         if (p->IOmask & 0x0080) fprintf(stderr,"XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));
1208:         if (p->IOmask & 0x0100) fprintf(stderr,"XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));
1209:         if (p->IOmask & 0x0200) fprintf(stderr,"XBee: Analog  0: %.2fv\n",(3.3/1023)*p->IOanalog[0]);
1210:         if (p->IOmask & 0x0400) fprintf(stderr,"XBee: Analog  1: %.2fv\n",(3.3/1023)*p->IOanalog[1]);
1211:         if (p->IOmask & 0x0800) fprintf(stderr,"XBee: Analog  2: %.2fv\n",(3.3/1023)*p->IOanalog[2]);
1212:         if (p->IOmask & 0x1000) fprintf(stderr,"XBee: Analog  3: %.2fv\n",(3.3/1023)*p->IOanalog[3]);
1213:         if (p->IOmask & 0x2000) fprintf(stderr,"XBee: Analog  4: %.2fv\n",(3.3/1023)*p->IOanalog[4]);
1214:         if (p->IOmask & 0x4000) fprintf(stderr,"XBee: Analog  5: %.2fv\n",(3.3/1023)*p->IOanalog[5]);
1215: #endif
1216:       }
1217: #ifdef DEBUG
1218:       fprintf(stderr,"XBee: ----------------------------\n");
1219: #endif
1220:
1221:     /* ######################################### */
1222:     /* if: Unknown */
1223:     } else {
1224: #ifdef DEBUG
1225:       fprintf(stderr,"XBee: Packet type: Unknown (0x%02X)\n",t);
1226: #endif
1227:       p->type = xbee_unknown;
1228:     }
1229:     p->next = NULL;
1230:
1231:     /* lock the connection mutex */
1232:     pthread_mutex_lock(&xbee.conmutex);
1233:
1234:     con = xbee.conlist;
1235:     hasCon = 0;
1236:     do {
1237:       if (xbee_matchpktcon(p,con)) {
1238:         hasCon = 1;
1239:         break;
1240:       }
1241:     } while ((con = con->next) != NULL);
1242:
1243:     /* unlock the connection mutex */
1244:     pthread_mutex_unlock(&xbee.conmutex);
1245:
1246:     /* if the packet doesn't have a connection, don't add it! */
1247:     if (!hasCon) {
1248:       Xfree(p);
1249: #ifdef DEBUG
1250:       fprintf(stderr,"XBee: Connectionless packet... discarding!\n");
1251: #endif
1252:       continue;
1253:     }
1254:
1255:     /* lock the packet mutex, so we can safely add the packet to the list */
1256:     pthread_mutex_lock(&xbee.pktmutex);
1257:     i = 1;
1258:     /* if: the list is empty */
1259:     if (!xbee.pktlist) {
1260:       /* start the list! */
1261:       xbee.pktlist = po;
1262:     } else {
1263:       /* add the packet to the end */
1264:       q = xbee.pktlist;
1265:       while (q->next) {
1266:         q = q->next;
1267:         i++;
1268:       }
1269:       q->next = po;
1270:     }
1271:
1272: #ifdef DEBUG
1273:     while (q && q->next) {
1274:       q = q->next;
1275:       i++;
```

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

```
1361:
1362:   /* free the packet */
1363:   Xfree(pkt);
1364: }
1365:
1366: /* ###################################################################
1367:    xbee_make_pkt - INTERNAL
1368:    adds delimiter field
1369:    calculates length and checksum
1370:    escapes bytes */
1371: t_data *xbee_make_pkt(unsigned char *data, int length) {
1372:   t_data *pkt;
1373:   unsigned int l, i, o, t, x, m;
1374:   char d = 0;
1375:
1376:   ISREADY;
1377:
1378:   /* check the data given isnt too long
1379:      100 bytes maximum payload + 12 bytes header information */
1380:   if (length > 100 + 12) return NULL;
1381:
1382:   /* calculate the length of the whole packet
1383:      start, length (MSB), length (LSB), DATA, checksum */
1384:   l = 3 + length + 1;
1385:
1386:   /* prepare memory */
1387:   pkt = Xcalloc(sizeof(t_data));
1388:
1389:   /* put start byte on */
1390:   pkt->data[0] = 0x7E;
1391:
1392:   /* copy data into packet */
1393:   for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
1394:     /* if: its time for the checksum */
1395:     if (i == length) d = M8((0xFF - M8(t)));
1396:     /* if: its time for the high length byte */
1397:     else if (m == 1) d = M8(length >> 8);
1398:     /* if: its time for the low length byte */
1399:     else if (m == 2) d = M8(length);
1400:     /* if: its time for the normal data */
1401:     else if (m > 2) d = data[i];
1402:
1403:     x = 0;
1404:     /* check for any escapes needed */
1405:     if ((d == 0x11) || /* XON */
1406:         (d == 0x13) || /* XOFF */
1407:         (d == 0x7D) || /* Escape */
1408:         (d == 0x7E)) { /* Frame Delimiter */
1409:       l++;
1410:       pkt->data[o++] = 0x7D;
1411:       x = 1;
1412:     }
1413:
1414:     /* move data in */
1415:     pkt->data[o] = ((!x)?d:d^0x20);
1416:     if (m > 2) {
1417:       i++;
1418:       t += d;
1419:     }
1420:   }
1421:
1422:   /* remember the length */
1423:   pkt->length = l;
1424:
1425:   return pkt;
1426: }
```