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

#define ISREADY                                 \
  if (!xbee_ready) {                            \
    printf("XBee: Run xbee_setup() first!...\n"); \
    exit(1);                                    \
  }

int xbee_ready = 0;
void *xbee_shm = NULL;

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

/* malloc */
void *Xmalloc(size_t size) {
  void *t;
  t = malloc(size);
  if (!t) {
    perror("xbee:malloc()");
    exit(1);
  }
  return t;
}

/* calloc */
void *Xcalloc(size_t size) {
  void *t;
  t = calloc(1, size);
  if (!t) {
    perror("xbee:calloc()");
    exit(1);
  }
  return t;
}

/* realloc */
void *Xrealloc(void *ptr, size_t size) {
  void *t;
  t = realloc(ptr,size);
  if (!t) {
    perror("xbee:realloc()");
    exit(1);
  }
  return t;
}

/* free */
void Xfree2(void **ptr) {
  free(*ptr);
  *ptr = NULL;
}

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

/* ########################################################################
   xbee_setup
   opens xbee serial port & creates xbee read thread
   the xbee must be configured for API mode 2, and 57600 baud
   THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
int xbee_setup(char *path, int baudrate) {
  t_info info;
  struct termios tc;
  speed_t chosenbaud = 57600;

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

  xbee.conlist = NULL;
  if (pthread_mutex_init(&xbee.conmutex,NULL)) {
```

```
  86:     perror("xbee_setup():pthread_mutex_init(conmutex)");
  87:     return -1;
  88:   }
  89:
  90:   xbee.pktlist = NULL;
  91:   if (pthread_mutex_init(&xbee.pktmutex,NULL)) {
  92:     perror("xbee_setup():pthread_mutex_init(pktmutex)");
  93:     return -1;
  94:   }
  95:
  96:   xbee.path = path;
  97:
  98:   /* open the serial port */
  99:   if ((xbee.ttyfd = open(path,O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1) {
 100:     perror("xbee_setup():open()");
 101:     xbee.path = NULL;
 102:     xbee.ttyfd = -1;
 103:     xbee.tty = NULL;
 104:     return -1;
 105:   }
 106:   /* setup the baud rate - 57600 8N1*/
 107:   tcgetattr(xbee.ttyfd, &tc);
 108:   cfsetispeed(&tc, chosenbaud);       /* set input baud rate to 57600 */
 109:   cfsetospeed(&tc, chosenbaud);       /* set output baud rate to 57600 */
 110:   /* input flags */
 111:   tc.c_iflag |= IGNBRK;           /* enable ignoring break */
 112:   tc.c_iflag &= ~(IGNPAR | PARMRK);/* disable parity checks */
 113:   tc.c_iflag &= ~INPCK;           /* disable parity checking */
 114:   tc.c_iflag &= ~ISTRIP;          /* disable stripping 8th bit */
 115:   tc.c_iflag &= ~(INLCR | ICRNL); /* disable translating NL <-> CR */
 116:   tc.c_iflag &= ~IGNCR;           /* disable ignoring CR */
 117:   tc.c_iflag &= ~(IXON | IXOFF);  /* disable XON/XOFF flow control */
 118:   /* output flags */
 119:   tc.c_oflag &= ~OPOST;           /* disable output processing */
 120:   tc.c_oflag &= ~(ONLCR | OCRNL); /* disable translating NL <-> CR */
 121:   tc.c_oflag &= ~OFILL;           /* disable fill characters */
 122:   /* control flags */
 123:   tc.c_cflag |= CREAD;            /* enable reciever */
 124:   tc.c_cflag &= ~PARENB;          /* disable parity */
 125:   tc.c_cflag &= ~CSTOPB;          /* disable 2 stop bits */
 126:   tc.c_cflag &= ~CSIZE;           /* remove size flag... */
 127:   tc.c_cflag |= CS8;              /* ...enable 8 bit characters */
 128:   tc.c_cflag |= HUPCL;            /* enable lower control lines on close - hang up */
 129:   /* local flags */
 130:   tc.c_lflag &= ~ISIG;            /* disable generating signals */
 131:   tc.c_lflag &= ~ICANON;          /* disable canonical mode - line by line */
 132:   tc.c_lflag &= ~ECHO;            /* disable echoing characters */
 133:   tc.c_lflag &= ~NOFLSH;          /* disable flushing on SIGINT */
 134:   tc.c_lflag &= ~IEXTEN;          /* disable input processing */
 135:   tcsetattr(xbee.ttyfd, TCSANOW, &tc);
 136:
 137:
 138:   if ((xbee.tty = fdopen(xbee.ttyfd,"r+")) == NULL) {
 139:     perror("xbee_setup():fdopen()");
 140:     xbee.path = NULL;
 141:     close(xbee.ttyfd);
 142:     xbee.ttyfd = -1;
 143:     xbee.tty = NULL;
 144:     return -1;
 145:   }
 146:
 147:   fflush(xbee.tty);
 148:
 149:   /* allow the listen thread to start */
 150:   xbee_ready = -1;
 151:
 152:   /* can start xbee_listen thread now */
 153:   if (pthread_create(&xbee.listent,NULL,(void *(*)(void *))xbee_listen,(void *)&info) != 0) {
 154:     perror("xbee_setup():pthread_create()");
 155:     return -1;
 156:   }
 157:
 158:   /* allow other functions to be used! */
 159:   xbee_ready = 1;
 160:   return 0;
 161: }
 162:
 163: /* ############################################################################
 164:    xbee_con
 165:    produces a connection to the specified device and frameID
 166:    if a connection had already been made, then this connection will be returned */
 167: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
 168:   xbee_con *con, *ocon;
 169:   unsigned char tAddr[8];
 170:   va_list ap;
```

```
171:   int t;
172: #ifdef DEBUG
173:   int i;
174: #endif
175:
176:   ISREADY;
177:
178:   va_start(ap,type);
179:   if ((type == xbee_64bitRemoteAT) ||
180:       (type == xbee_64bitData) ||
181:       (type == xbee_64bitIO)) {
182:     /* 64 bit address expected (2 ints) */
183:     t = va_arg(ap, int);
184:     tAddr[0] = (t >> 24) & 0xFF;
185:     tAddr[1] = (t >> 16) & 0xFF;
186:     tAddr[2] = (t >>  8) & 0xFF;
187:     tAddr[3] = (t      ) & 0xFF;
188:     t = va_arg(ap, int);
189:     tAddr[4] = (t >> 24) & 0xFF;
190:     tAddr[5] = (t >> 16) & 0xFF;
191:     tAddr[6] = (t >>  8) & 0xFF;
192:     tAddr[7] = (t      ) & 0xFF;
193:   } else if ((type == xbee_16bitRemoteAT) ||
194:              (type == xbee_16bitData) ||
195:              (type == xbee_16bitIO)) {
196:     /* 16 bit address expected (1 int) */
197:     t = va_arg(ap, int);
198:     tAddr[0] = (t >>  8) & 0xFF;
199:     tAddr[1] = (t      ) & 0xFF;
200:     tAddr[2] = 0;
201:     tAddr[3] = 0;
202:     tAddr[4] = 0;
203:     tAddr[5] = 0;
204:     tAddr[6] = 0;
205:     tAddr[7] = 0;
206:   }
207:   va_end(ap);
208:
209:   if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
210:   else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
211:
212:   pthread_mutex_lock(&xbee.conmutex);
213:
214:   if (xbee.conlist) {
215:     con = xbee.conlist;
216:     while (con) {
217:       if ((type == con->type) &&
218:           (frameID == con->frameID)) {
219:
220:         if (type == xbee_localAT) {
221:           /* already has connection to local modem with that frameID */
222:           pthread_mutex_unlock(&xbee.conmutex);
223:           return con;
224:         } else if ((type == con->type) &&
225:
226:                    ((((type == xbee_16bitRemoteAT) ||
227:                       (type == xbee_16bitData) ||
228:                       (type == xbee_16bitIO)) &&
229:                      (!memcmp(tAddr,con->tAddr,2))) ||
230:
231:                     (((type == xbee_64bitRemoteAT) ||
232:                       (type == xbee_64bitData) ||
233:                       (type == xbee_64bitIO)) &&
234:                      (!memcmp(tAddr,con->tAddr,8))))) {
235:           /* addressing modes & addresses match */
236:           pthread_mutex_unlock(&xbee.conmutex);
237:           return con;
238:         }
239:       }
240:       if (con->next == NULL) break;
241:       con = con->next;
242:     }
243:     ocon = con;
244:   }
245:   con = Xcalloc(sizeof(xbee_con));
246:   con->type = type;
247:   if ((type == xbee_64bitRemoteAT) ||
248:       (type == xbee_64bitData) ||
249:       (type == xbee_64bitIO)) {
250:     con->tAddr64 = TRUE;
251:   }
252:   con->atQueue = 0;
253:   con->txDisableACK = 0;
254:   con->txBroadcast = 0;
255:   con->frameID = frameID;
```

```
256:   if (type != xbee_localAT) {
257:     if (con->tAddr64) {
258:       memcpy(con->tAddr,tAddr,8);
259:     } else {
260:       memcpy(con->tAddr,tAddr,2);
261:       memset(&con->tAddr[2],0,6);
262:     }
263:   }
264:
265: #ifdef DEBUG
266:   switch(type) {
267:     case xbee_localAT:
268:       printf("XBee: New local AT connection!\n");
269:       break;
270:     case xbee_16bitRemoteAT:
271:     case xbee_64bitRemoteAT:
272:       printf("XBee: New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
273:       for (i=0;i<(con->tAddr64?8:2);i++) {
274:         printf((i?":%02X":"%02X"),tAddr[i]);
275:       }
276:       printf(")\n");
277:       break;
278:     case xbee_16bitData:
279:     case xbee_64bitData:
280:       printf("XBee: New %d-bit data connection! (to: ",(con->tAddr64?64:16));
281:       for (i=0;i<(con->tAddr64?8:2);i++) {
282:         printf((i?":%02X":"%02X"),tAddr[i]);
283:       }
284:       printf(")\n");
285:       break;
286:     case xbee_16bitIO:
287:     case xbee_64bitIO:
288:       printf("XBee: New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
289:       for (i=0;i<(con->tAddr64?8:2);i++) {
290:         printf((i?":%02X":"%02X"),tAddr[i]);
291:       }
292:       printf(")\n");
293:       break;
294:     case xbee_txStatus:
295:       printf("XBee: New status connection!\n");
296:       break;
297:     case xbee_modemStatus:
298:       break;
299:     case xbee_unknown:
300:     default:
301:       printf("XBee: New unknown connection!\n");
302:   }
303: #endif
304:
305:   con->next = NULL;
306:   if (xbee.conlist) {
307:     ocon->next = con;
308:   } else {
309:     xbee.conlist = con;
310:   }
311:   pthread_mutex_unlock(&xbee.conmutex);
312:   return con;
313: }
314:
315: /* ################################################################
316:    xbee_senddata
317:    send the specified data to the provided connection */
318: int xbee_senddata(xbee_con *con, char *format, ...) {
319:   t_data *pkt;
320:   int i, length;
321:   unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
322:   unsigned char data[128]; /* ditto */
323:   va_list ap;
324:
325:   ISREADY;
326:
327:   va_start(ap, format);
328:   length = vsnprintf((char *)data,128,format,ap);
329:   va_end(ap);
330:
331: #ifdef DEBUG
332:   printf("XBee: --== TX Packet ============--\n");
333:   printf("XBee: Length: %d\n",length);
334:   for (i=0;i<length;i++) {
335:     printf("XBee: %3d | 0x%02X ",i,data[i]);
336:     if ((data[i] > 32) && (data[i] < 127)) printf("'%c'\n",data[i]); else printf(" _\n");
337:   }
338: #endif
339:
340:   if (!con) return -1;
```

```
341:   if (con->type == xbee_unknown) return -1;
342:
343:   /* ######################################### */
344:   /* local AT mode */
345:   if (con->type == xbee_localAT) {
346:     if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
347:     if (!con->atQueue) {
348:       buf[0] = 0x08;
349:     } else {
350:       buf[1] = 0x09;
351:     }
352:     buf[1] = con->frameID;
353:     for (i=0;i<length;i++) {
354:       buf[i+2] = data[i];
355:     }
356:     pkt = xbee_make_pkt(buf,i+2);
357:     xbee_send_pkt(pkt);
358:     return 1;
359:   }
360:   if ((con->type == xbee_16bitRemoteAT) ||
361:       (con->type == xbee_64bitRemoteAT)) {
362:     /* ######################################### */
363:     /* remote AT mode */
364:     buf[0] = 0x17;
365:     buf[1] = con->frameID;
366:     if (con->tAddr64) {
367:       memcpy(&buf[2],con->tAddr,8);
368:       buf[10] = 0xFF;
369:       buf[11] = 0xFE;
370:     } else {
371:       memset(&buf[2],0,8);
372:       memcpy(&buf[10],con->tAddr,2);
373:     }
374:     buf[12] = ((!con->atQueue)?0x02:0x00);
375:     for (i=0;i<length;i++) {
376:       buf[i+13] = data[i];
377:     }
378:     pkt = xbee_make_pkt(buf,i+13);
379:     xbee_send_pkt(pkt);
380:     return 1;
381:   } else if (con->type == xbee_64bitData) {
382:     /* ######################################### */
383:     /* 64bit Data */
384:     buf[0] = 0x00;
385:     buf[1] = con->frameID;
386:     memcpy(&buf[2],con->tAddr,8);
387:     buf[10] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
388:     for (i=0;i<length;i++) {
389:       buf[i+11] = data[i];
390:     }
391:     pkt = xbee_make_pkt(buf,i+11);
392:     xbee_send_pkt(pkt);
393:     return 1;
394:   } else if (con->type == xbee_16bitData) {
395:     /* ######################################### */
396:     /* 16bit Data */
397:     buf[0] = 0x01;
398:     buf[1] = con->frameID;
399:     memcpy(&buf[2],con->tAddr,2);
400:     buf[4] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
401:     for (i=0;i<length;i++) {
402:       buf[i+5] = data[i];
403:     }
404:     pkt = xbee_make_pkt(buf,i+5);
405:     xbee_send_pkt(pkt);
406:     return 1;
407:   } else if ((con->type == xbee_64bitIO) ||
408:              (con->type == xbee_16bitIO)) {
409:     printf("******* TODO ********\n");
410:   }
411:   return 0;
412: }
413:
414: /* ###################################################################
415:    xbee_getpacket
416:    retrieves the next packet destined for the given connection
417:    once the packet has been retrieved, it is removed for the list! */
418: xbee_pkt *xbee_getpacket(xbee_con *con) {
419:   xbee_pkt *l, *p, *q;
420: #ifdef DEBUG
421:   int c;
422:   printf("XBee: --== Get Packet ==========--\n");
423: #endif
424:
425:   pthread_mutex_lock(&xbee.pktmutex);
```

```
426:
427:   if ((p = xbee.pktlist) == NULL) {
428:     pthread_mutex_unlock(&xbee.pktmutex);
429: #ifdef DEBUG
430:     printf("XBee: No packets avaliable...\n");
431: #endif
432:     return NULL;
433:   }
434:
435:   l = NULL;
436:   q = NULL;
437:   do {
438:     if ((p->type == con->type) || /* -- */
439:         ((p->type == xbee_remoteAT) &&
440:          (con->type == xbee_16bitRemoteAT)) || /* -- */
441:         ((p->type == xbee_remoteAT) &&
442:          (con->type == xbee_64bitRemoteAT))) { /* -- */
443:       /* if: the connection type matches the packet type OR
444:              the connection is 16bit remote AT, and the packet is a remote AT response OR
445:              the connection is 64bit remote AT, and the packet is a remote AT response */
446:       if ((((p->type == xbee_localAT) ||
447:             (p->type == xbee_remoteAT)) &&
448:            (con->frameID == p->frameID)) ||
449:           ((con->tAddr64 && !memcmp(con->tAddr,p->Addr64,8)) ||
450:            (!con->tAddr64 && !memcmp(con->tAddr,p->Addr16,2)))) {
451:         /* if: the packet is AT data, and the frame IDs match OR
452:                the corresponding addresses match */
453:         q = p;
454:         break;
455:       }
456:     }
457:     l = p;
458:     p = p->next;
459:   } while (p);
460:
461:   if (!q) {
462:     pthread_mutex_unlock(&xbee.pktmutex);
463: #ifdef DEBUG
464:     printf("XBee: No packets avaliable (for connection)...\n");
465: #endif
466:     return NULL;
467:   }
468:
469:   if (!l) {
470:     xbee.pktlist = p->next;
471:   } else {
472:     l->next = p->next;
473:   }
474:
475: #ifdef DEBUG
476:   printf("XBee: Got a packet\n");
477:   for (p = xbee.pktlist,c = 0;p;c++,p = p->next);
478:   printf("XBee: Packets left: %d\n",c);
479: #endif
480:
481:   pthread_mutex_unlock(&xbee.pktmutex);
482:
483:   return q;
484: }
485:
486: /* ########################################################################
487:    xbee_listen - INTERNAL
488:    the xbee xbee_listen thread
489:    reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
490: void xbee_listen(t_info *info) {
491:   unsigned char c, t, d[128];
492:   unsigned int l, i, s, o;
493: #ifdef DEBUG
494:   int j;
495: #endif
496:   xbee_pkt *p, *q, *po;
497:
498:   /* just falls out if the proper 'go-ahead' isn't given */
499:   if (xbee_ready != -1) return;
500:
501:   while(1) {
502:
503:     c = xbee_getRawByte();
504:
505:     if (c != 0x7E) continue;
506: #ifdef DEBUG
507:     printf("XBee: --== RX Packet ==========--\nXBee: Got a packet!...\n");
508: #endif
509:
510:     l = xbee_getByte() << 8;
```

```
511:     l += xbee_getByte();
512:
513:     if (!l) continue;
514:
515: #ifdef DEBUG
516:     printf("XBee: Length: %d\n",l - 1);
517: #endif
518:
519:     t = xbee_getByte();
520:
521:     for (i=0,s=0;l>1 && i<128;l--,i++) {
522:       c = xbee_getByte();
523:       d[i] = c;
524:       s += c;
525: #ifdef DEBUG
526:       printf("XBee: %3d | 0x%02X ",i,c);
527:       if ((c > 32) && (c < 127)) printf("'%c'\n",c); else printf(" _\n");
528: #endif
529:     }
530:     i--; /* it went up too many times! */
531:     c = xbee_getByte();
532:     s += c;
533:     s &= 0xFF;
534: #ifdef DEBUG
535:     printf("XBee: Checksum: 0x%02X   Result: 0x%02X\n",c,s);
536: #endif
537:     if (l>1) {
538: #ifdef DEBUG
539:       printf("XBee: Didn't get whole packet... :(\n");
540: #endif
541:       continue;
542:     }
543:
544:     po = p = Xcalloc(sizeof(xbee_pkt));
545:     q = NULL;
546:     p->datalen = l;
547:
548:     /* ######################################### */
549:     /* modem status */
550:     if (t == 0x8A) {
551: #ifdef DEBUG
552:       printf("XBee: Packet type: Modem Status (0x8A)\n");
553:       printf("XBee: ");
554:       switch (d[0]) {
555:       case 0x00: printf("Hardware reset"); break;
556:       case 0x01: printf("Watchdog timer reset"); break;
557:       case 0x02: printf("Associated"); break;
558:       case 0x03: printf("Disassociated"); break;
559:       case 0x04: printf("Synchronization lost"); break;
560:       case 0x05: printf("Coordinator realignment"); break;
561:       case 0x06: printf("Coordinator started"); break;
562:       }
563:       printf("...\n");
564: #endif
565:       p->type = xbee_modemStatus;
566:
567:       p->sAddr64 = FALSE;
568:       p->dataPkt = FALSE;
569:       p->txStatusPkt = FALSE;
570:       p->modemStatusPkt = TRUE;
571:       p->remoteATPkt = FALSE;
572:       p->IOPkt = FALSE;
573:
574:       p->datalen = 1;
575:       p->data[0] = d[0];
576:     /* ######################################### */
577:     /* local AT response */
578:     } else if (t == 0x88) {
579: #ifdef DEBUG
580:       printf("XBee: Packet type: Local AT Response (0x88)\n");
581:       printf("XBee: FrameID: 0x%02X\n",d[0]);
582:       printf("XBee: AT Command: %c%c\n",d[1],d[2]);
583:       if (d[3] == 0) printf("XBee: Status: OK\n");
584:       else if (d[3] == 1) printf("XBee: Status: Error\n");
585:       else if (d[3] == 2) printf("XBee: Status: Invalid Command\n");
586:       else if (d[3] == 3) printf("XBee: Status: Invalid Parameter\n");
587: #endif
588:       p->type = xbee_localAT;
589:
590:       p->sAddr64 = FALSE;
591:       p->dataPkt = FALSE;
592:       p->txStatusPkt = FALSE;
593:       p->modemStatusPkt = FALSE;
594:       p->remoteATPkt = FALSE;
595:       p->IOPkt = FALSE;
```

```
596:
597:       p->frameID = d[0];
598:       p->atCmd[0] = d[1];
599:       p->atCmd[1] = d[2];
600:
601:       p->status = d[3];
602:
603:       p->datalen = i-3;
604:       for (;i>3;i--) p->data[i-4] = d[i];
605:     /* ######################################### */
606:     /* remote AT response */
607:     } else if (t == 0x97) {
608: #ifdef DEBUG
609:       printf("XBee: Packet type: Remote AT Response (0x97)\n");
610:       printf("XBee: FrameID: 0x%02X\n",d[0]);
611:       printf("XBee: 64-bit Address: ");
612:       for (j=0;j<8;j++) {
613:         printf((j?":%02X":"%02X"),d[1+j]);
614:       }
615:       printf("\n");
616:       printf("XBee: 16-bit Address: ");
617:       for (j=0;j<2;j++) {
618:         printf((j?":%02X":"%02X"),d[9+j]);
619:       }
620:       printf("\n");
621:       printf("XBee: AT Command: %c%c\n",d[11],d[12]);
622:       if (d[13] == 0) printf("XBee: Status: OK\n");
623:       else if (d[13] == 1) printf("XBee: Status: Error\n");
624:       else if (d[13] == 2) printf("XBee: Status: Invalid Command\n");
625:       else if (d[13] == 3) printf("XBee: Status: Invalid Parameter\n");
626:       else if (d[13] == 4) printf("XBee: Status: No Response\n");
627: #endif
628:       p->type = xbee_remoteAT;
629:
630:       p->sAddr64 = FALSE;
631:       p->dataPkt = FALSE;
632:       p->txStatusPkt = FALSE;
633:       p->modemStatusPkt = FALSE;
634:       p->remoteATPkt = TRUE;
635:       p->IOPkt = FALSE;
636:
637:       p->frameID = d[0];
638:
639:       p->Addr64[0] = d[1];
640:       p->Addr64[1] = d[2];
641:       p->Addr64[2] = d[3];
642:       p->Addr64[3] = d[4];
643:       p->Addr64[4] = d[5];
644:       p->Addr64[5] = d[6];
645:       p->Addr64[6] = d[7];
646:       p->Addr64[7] = d[8];
647:
648:       p->Addr16[0] = d[9];
649:       p->Addr16[1] = d[10];
650:
651:       p->atCmd[0] = d[11];
652:       p->atCmd[1] = d[12];
653:
654:       p->status = d[13];
655:
656:       p->datalen = i-13;
657:       for (;i>13;i--) p->data[i-14] = d[i];
658:     /* ######################################### */
659:     /* TX status */
660:     } else if (t == 0x89) {
661: #ifdef DEBUG
662:       printf("XBee: Packet type: TX Status Report (0x89)\n");
663:       printf("XBee: FrameID: 0x%02X\n",d[0]);
664:       if (d[1] == 0) printf("XBee: Status: Success\n");
665:       else if (d[1] == 1) printf("XBee: Status: No ACK\n");
666:       else if (d[1] == 2) printf("XBee: Status: CCA Failure\n");
667:       else if (d[1] == 3) printf("XBee: Status: Purged\n");
668: #endif
669:       p->type = xbee_txStatus;
670:
671:       p->sAddr64 = FALSE;
672:       p->dataPkt = FALSE;
673:       p->txStatusPkt = TRUE;
674:       p->modemStatusPkt = FALSE;
675:       p->remoteATPkt = FALSE;
676:       p->IOPkt = FALSE;
677:
678:       p->frameID = d[0];
679:
680:       p->status = d[1];
```

```
681:     /* ######################################### */
682:     /* 64bit address recieve */
683:     } else if (t == 0x80) {
684: #ifdef DEBUG
685:       printf("XBee: Packet type: 64-bit RX Data (0x80)\n");
686:       printf("XBee: 64-bit Address: ");
687:       for (j=0;j<8;j++) {
688:         printf((j?":%02X":"%02X"),d[j]);
689:       }
690:       printf("\n");
691:       printf("XBee: RSSI: -%ddB\n",d[8]);
692:       if (d[9] & 0x02) printf("XBee: Options: Address Broadcast\n");
693:       if (d[9] & 0x03) printf("XBee: Options: PAN Broadcast\n");
694: #endif
695:       p->type = xbee_64bitData;
696:
697:       p->sAddr64 = TRUE;
698:       p->dataPkt = TRUE;
699:       p->txStatusPkt = FALSE;
700:       p->modemStatusPkt = FALSE;
701:       p->remoteATPkt = FALSE;
702:       p->IOPkt = FALSE;
703:
704:       p->Addr64[0] = d[0];
705:       p->Addr64[1] = d[1];
706:       p->Addr64[2] = d[2];
707:       p->Addr64[3] = d[3];
708:       p->Addr64[4] = d[4];
709:       p->Addr64[5] = d[5];
710:       p->Addr64[6] = d[6];
711:       p->Addr64[7] = d[7];
712:
713:       p->RSSI = d[8];
714:
715:       p->status = d[9];
716:
717:       p->datalen = i-9;
718:       for (;i>9;i--) p->data[i-10] = d[i];
719:     /* ######################################### */
720:     /* 16bit address recieve */
721:     } else if (t == 0x81) {
722: #ifdef DEBUG
723:       printf("XBee: Packet type: 16-bit RX Data (0x81)\n");
724:       printf("XBee: 16-bit Address: ");
725:       for (j=0;j<2;j++) {
726:         printf((j?":%02X":"%02X"),d[j]);
727:       }
728:       printf("\n");
729:       printf("XBee: RSSI: -%ddB\n",d[2]);
730:       if (d[3] & 0x02) printf("XBee: Options: Address Broadcast\n");
731:       if (d[3] & 0x03) printf("XBee: Options: PAN Broadcast\n");
732: #endif
733:       p->type = xbee_16bitData;
734:
735:       p->sAddr64 = FALSE;
736:       p->dataPkt = TRUE;
737:       p->txStatusPkt = FALSE;
738:       p->modemStatusPkt = FALSE;
739:       p->remoteATPkt = FALSE;
740:       p->IOPkt = FALSE;
741:
742:       p->Addr16[0] = d[0];
743:       p->Addr16[1] = d[1];
744:
745:       p->RSSI = d[2];
746:
747:       p->status = d[3];
748:
749:       p->datalen = i-3;
750:       for (;i>3;i--) p->data[i-4] = d[i];
751:     /* ######################################### */
752:     /* 64bit I/O recieve */
753:     } else if (t == 0x82) {
754: #ifdef DEBUG
755:       printf("XBee: Packet type: 64-bit RX I/O Data (0x82)\n");
756:       printf("XBee: 64-bit Address: ");
757:       for (j=0;j<8;j++) {
758:         printf((j?":%02X":"%02X"),d[j]);
759:       }
760:       printf("\n");
761:       printf("XBee: RSSI: -%ddB\n",d[8]);
762:       if (d[9] & 0x02) printf("XBee: Options: Address Broadcast\n");
763:       if (d[9] & 0x02) printf("XBee: Options: PAN Broadcast\n");
764:       printf("XBee: Samples: %d\n",d[10]);
765: #endif
```

```
766:       i = 13;
767:
768:       for (o=d[10];o>0;o--) {
769: #ifdef DEBUG
770:         printf("XBee: --- Sample %3d ------------\n",o-d[10]+1);
771: #endif
772:         if (o<d[10]) {
773:           q = Xcalloc(sizeof(xbee_pkt));
774:           p->next = q;
775:           p = q;
776:           p->datalen = l;
777:         }
778:
779:         p->type = xbee_64bitIO;
780:
781:         p->sAddr64 = TRUE;
782:         p->dataPkt = FALSE;
783:         p->txStatusPkt = FALSE;
784:         p->modemStatusPkt = FALSE;
785:         p->remoteATPkt = FALSE;
786:         p->IOPkt = TRUE;
787:
788:         p->Addr64[0] = d[0];
789:         p->Addr64[1] = d[1];
790:         p->Addr64[2] = d[2];
791:         p->Addr64[3] = d[3];
792:         p->Addr64[4] = d[4];
793:         p->Addr64[5] = d[5];
794:         p->Addr64[6] = d[6];
795:         p->Addr64[7] = d[7];
796:
797:         p->RSSI = d[8];
798:
799:         p->status = d[9];
800:
801:         p->IOmask = (((d[11]<<8) | d[12]) & 0x7FFF);
802:         p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);
803:
804:         i += (((d[11]&0x01)||(d[12]))?2:0);
805:
806:         if (d[11]&0x02) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
807:         if (d[11]&0x04) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
808:         if (d[11]&0x08) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
809:         if (d[11]&0x10) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
810:         if (d[11]&0x20) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
811:         if (d[11]&0x40) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
812: #ifdef DEBUG
813:         if (p->IOmask & 0x0001) printf("XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
814:         if (p->IOmask & 0x0002) printf("XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
815:         if (p->IOmask & 0x0004) printf("XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
816:         if (p->IOmask & 0x0008) printf("XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
817:         if (p->IOmask & 0x0010) printf("XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
818:         if (p->IOmask & 0x0020) printf("XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
819:         if (p->IOmask & 0x0040) printf("XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
820:         if (p->IOmask & 0x0080) printf("XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));
821:         if (p->IOmask & 0x0100) printf("XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));
822:         if (p->IOmask & 0x0200) printf("XBee: Analog  0: %.2fv\n",(3.3/1023)*p->IOanalog[0]);
823:         if (p->IOmask & 0x0400) printf("XBee: Analog  1: %.2fv\n",(3.3/1023)*p->IOanalog[1]);
824:         if (p->IOmask & 0x0800) printf("XBee: Analog  2: %.2fv\n",(3.3/1023)*p->IOanalog[2]);
825:         if (p->IOmask & 0x1000) printf("XBee: Analog  3: %.2fv\n",(3.3/1023)*p->IOanalog[3]);
826:         if (p->IOmask & 0x2000) printf("XBee: Analog  4: %.2fv\n",(3.3/1023)*p->IOanalog[4]);
827:         if (p->IOmask & 0x4000) printf("XBee: Analog  5: %.2fv\n",(3.3/1023)*p->IOanalog[5]);
828: #endif
829:       }
830: #ifdef DEBUG
831:       printf("XBee: ---------------------------\n");
832: #endif
833:     } else if (t == 0x83) {
834:     /* ########################################### */
835:     /* 16bit I/O recieve */
836: #ifdef DEBUG
837:       printf("XBee: Packet type: 16-bit RX I/O Data (0x83)\n");
838:       printf("XBee: 16-bit Address: ");
839:       for (j=0;j<2;j++) {
840:         printf((j?":%02X":"%02X"),d[j]);
841:       }
842:       printf("\n");
843:       printf("XBee: RSSI: -%ddB\n",d[2]);
844:       if (d[3] & 0x02) printf("XBee: Options: Address Broadcast\n");
845:       if (d[3] & 0x02) printf("XBee: Options: PAN Broadcast\n");
846:       printf("XBee: Samples: %d\n",d[4]);
847: #endif
848:
849:       i = 7;
850:
```

```
851:       for (o=d[4];o>0;o--) {
852: #ifdef DEBUG
853:         printf("XBee: --- Sample %3d ------------\n",o-d[4]+1);
854: #endif
855:         if (o<d[4]) {
856:           q = Xcalloc(sizeof(xbee_pkt));
857:           p->next = q;
858:           p = q;
859:           p->datalen = l;
860:         }
861:
862:         p->type = xbee_16bitIO;
863:
864:         p->sAddr64 = FALSE;
865:         p->dataPkt = FALSE;
866:         p->txStatusPkt = FALSE;
867:         p->modemStatusPkt = FALSE;
868:         p->remoteATPkt = FALSE;
869:         p->IOPkt = TRUE;
870:
871:         p->Addr16[0] = d[0];
872:         p->Addr16[1] = d[1];
873:
874:         p->RSSI = d[2];
875:
876:         p->status = d[3];
877:
878:         p->IOmask = (((d[5]<<8) | d[6]) & 0x7FFF);
879:         p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);
880:
881:         i += (((d[5]&0x01)||(d[6]))?2:0);
882:
883:         if (d[5]&0x02) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
884:         if (d[5]&0x04) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
885:         if (d[5]&0x08) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
886:         if (d[5]&0x10) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
887:         if (d[5]&0x20) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
888:         if (d[5]&0x40) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
889: #ifdef DEBUG
890:         if (p->IOmask & 0x0001) printf("XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
891:         if (p->IOmask & 0x0002) printf("XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
892:         if (p->IOmask & 0x0004) printf("XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
893:         if (p->IOmask & 0x0008) printf("XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
894:         if (p->IOmask & 0x0010) printf("XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
895:         if (p->IOmask & 0x0020) printf("XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
896:         if (p->IOmask & 0x0040) printf("XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
897:         if (p->IOmask & 0x0080) printf("XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));
898:         if (p->IOmask & 0x0100) printf("XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));
899:         if (p->IOmask & 0x0200) printf("XBee: Analog  0: %.2fv\n",(3.3/1023)*p->IOanalog[0]);
900:         if (p->IOmask & 0x0400) printf("XBee: Analog  1: %.2fv\n",(3.3/1023)*p->IOanalog[1]);
901:         if (p->IOmask & 0x0800) printf("XBee: Analog  2: %.2fv\n",(3.3/1023)*p->IOanalog[2]);
902:         if (p->IOmask & 0x1000) printf("XBee: Analog  3: %.2fv\n",(3.3/1023)*p->IOanalog[3]);
903:         if (p->IOmask & 0x2000) printf("XBee: Analog  4: %.2fv\n",(3.3/1023)*p->IOanalog[4]);
904:         if (p->IOmask & 0x4000) printf("XBee: Analog  5: %.2fv\n",(3.3/1023)*p->IOanalog[5]);
905: #endif
906:       }
907: #ifdef DEBUG
908:       printf("XBee: ---------------------------\n");
909: #endif
910:     /* ######################################### */
911:     /* Unknown */
912:     } else {
913: #ifdef DEBUG
914:       printf("XBee: Packet type: Unknown (0x%02X)\n",t);
915: #endif
916:       p->type = xbee_unknown;
917:     }
918:     p->next = NULL;
919:
920:     pthread_mutex_lock(&xbee.pktmutex);
921:     i = 1;
922:     if (!xbee.pktlist) {
923:       xbee.pktlist = po;
924:     } else {
925:       q = xbee.pktlist;
926:       while (q->next) {
927:         q = q->next;
928:         i++;
929:       }
930:       q->next = po;
931:     }
932:
933: #ifdef DEBUG
934:     while (q && q->next) {
935:       q = q->next;
```

```
936:       i++;
937:     }
938:     printf("XBee: --======================--\n");
939:     printf("XBee: Packets: %d\n",i);
940: #endif
941:
942:     po = p = q = NULL;
943:     pthread_mutex_unlock(&xbee.pktmutex);
944:   }
945: }
946:
947: /* ############################################################################
948:    xbee_getByte - INTERNAL
949:    waits for an escaped byte of data */
950: unsigned char xbee_getByte(void) {
951:   unsigned char c;
952:
953:   ISREADY;
954:
955:   c = xbee_getRawByte();
956:   if (c == 0x7D) c = xbee_getRawByte() ^ 0x20;
957:
958:   return (c & 0xFF);
959: }
960:
961: /* ############################################################################
962:    xbee_getRawByte - INTERNAL
963:    waits for a raw byte of data */
964: unsigned char xbee_getRawByte(void) {
965:   unsigned char c;
966:   fd_set fds;
967:
968:   ISREADY;
969:
970:   FD_ZERO(&fds);
971:   FD_SET(xbee.ttyfd,&fds);
972:
973:   if (select(xbee.ttyfd+1,&fds,NULL,NULL,NULL) == -1) {
974:     perror("xbee:xbee_listen():xbee_getByte()");
975:     exit(1);
976:   }
977:
978:   do {
979:     if (read(xbee.ttyfd,&c,1) == 0) {
980:       usleep(10);
981:       continue;
982:     }
983:   } while (0);
984:
985:   return (c & 0xFF);
986: }
987:
988: /* ############################################################################
989:    xbee_send_pkt - INTERNAL
990:    sends a complete packet of data */
991: void xbee_send_pkt(t_data *pkt) {
992:   ISREADY;
993:
994:   /* write and flush the data */
995:   fwrite(pkt->data,pkt->length,1,xbee.tty);
996:   fflush(xbee.tty);
997:
998: #ifdef DEBUG
999:   {
1000:     int i;
1001:     /* prints packet in hex byte-by-byte */
1002:     printf("XBee: TX Packet - ");
1003:     for (i=0;i<pkt->length;i++) {
1004:       printf("0x%02X ",pkt->data[i]);
1005:     }
1006:     printf("\n");
1007:   }
1008: #endif
1009:
1010:   xbee_destroy_pkt(pkt);
1011: }
1012:
1013: /* ############################################################################
1014:    xbee_make_pkt - INTERNAL
1015:    adds delimiter field
1016:    calculates length and checksum
1017:    escapes bytes */
1018: t_data *xbee_make_pkt(unsigned char *data, int length) {
1019:   t_data *pkt;
1020:   unsigned int l, i, o, t, x, m;
```

```
1021:   char d = 0;
1022:
1023:   ISREADY;
1024:
1025:   /* check the data given isnt too long */
1026:   if (length > 0xFFFF) return NULL;
1027:
1028:   /* calculate the length of the whole packet */
1029:   l = 3 + length + 1;
1030:
1031:   /* prepare memory */
1032:   pkt = Xmalloc(sizeof(t_data));
1033:
1034:   /* put start byte on */
1035:   pkt->data[0] = 0x7E;
1036:
1037:   /* copy data into packet */
1038:   for (t=0,i=0,o=1,m=1;i<=length;o++,m++) {
1039:     if (i == length) {
1040:       d = M8((0xFF - M8(t)));
1041:     }
1042:     else if (m == 1) d = M8(length >> 8);
1043:     else if (m == 2) d = M8(length);
1044:     else if (m > 2) d = data[i];
1045:     x = 0;
1046:     /* check for any escapes needed */
1047:     if ((d == 0x11) || /* XON */
1048:         (d == 0x13) || /* XOFF */
1049:         (d == 0x7D) || /* Escape */
1050:         (d == 0x7E)) { /* Frame Delimiter */
1051:       l++;
1052:       pkt->data[o++] = 0x7D;
1053:       x = 1;
1054:     }
1055:
1056:     /* move data in */
1057:     pkt->data[o] = (!x)?(d):(d^0x20);
1058:     if (m > 2) {
1059:       i++;
1060:       t += d;
1061:     }
1062:   }
1063:
1064:   /* remember the length */
1065:   pkt->length = l;
1066:
1067:   return pkt;
1068: }
1069:
1070: /* ######################################################################
1071:    xbee_destroy_pkt - INTERNAL
1072:    free's the packet memory */
1073: void xbee_destroy_pkt(t_data *pkt) {
1074:
1075:   ISREADY;
1076:
1077:   /* free the stuff! */
1078:   Xfree(pkt);
1079: }
```