

```

1: #include "globals.h"
2: #include "api.h"
3:
4: #define ISREADY \
5:     if (!xbee_ready) { \
6:         printf("XBee: Run xbee_setup() first!...\n"); \
7:         exit(1); \
8:     }
9:
10: int xbee_ready = 0;
11: void *xbee_shm = NULL;
12:
13: /* ##### */
14: /* ### Memory Handling ##### */
15: /* ##### */
16:
17: /* malloc */
18: void *Xmalloc(size_t size) {
19:     void *t;
20:     t = malloc(size);
21:     if (!t) {
22:         perror("xbee:malloc()");
23:         exit(1);
24:     }
25:     return t;
26: }
27:
28: /* calloc */
29: void *Xcalloc(size_t size) {
30:     void *t;
31:     t = calloc(1, size);
32:     if (!t) {
33:         perror("xbee:calloc()");
34:         exit(1);
35:     }
36:     return t;
37: }
38:
39: /* realloc */
40: void *Xrealloc(void *ptr, size_t size) {
41:     void *t;
42:     t = realloc(ptr, size);
43:     if (!t) {
44:         perror("xbee:realloc()");
45:         exit(1);
46:     }
47:     return t;
48: }
49:
50: /* free */
51: void Xfree2(void **ptr) {
52:     free(*ptr);
53:     *ptr = NULL;
54: }
55:
56: /* ##### */
57: /* ### XBee Functions ##### */
58: /* ##### */
59:
60: /* #####
61: xbee_setup
62: opens xbee serial port & creates xbee read thread
63: the xbee must be configured for API mode 2, and 57600 baud
64: THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
65: void xbee_setup(char *path) {
66:     t_info info;
67:     struct termios tc;
68:
69:     xbee.conlist = NULL;
70:     if (pthread_mutex_init(&xbee.conmutex, NULL)) {
71:         perror("xbee_setup():pthread_mutex_init(conmutex)");
72:         exit(1);
73:     }
74:
75:     xbee.pktlist = NULL;
76:     if (pthread_mutex_init(&xbee.pktmutex, NULL)) {
77:         perror("xbee_setup():pthread_mutex_init(pktmutex)");
78:         exit(1);
79:     }
80:
81:     xbee.path = path;
82:
83:     /* open the serial port */
84:     if ((xbee.ttyfd = open(path, O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1) {
85:         perror("xbee_setup():open()");

```

```

86:     xbee.path = NULL;
87:     xbee.ttyfd = -1;
88:     xbee.tty = NULL;
89:     exit(1);
90: }
91: /* setup the baud rate - 57600 8N1*/
92: tcgetattr(xbee.ttyfd, &tc);
93: cfsetispeed(&tc, B57600); /* set input baud rate to 57600 */
94: cfsetospeed(&tc, B57600); /* set output baud rate to 57600 */
95: /* input flags */
96: tc.c_iflag |= IGNBRK; /* enable ignoring break */
97: tc.c_iflag &= ~(IGNPAR | PARMRK); /* disable parity checks */
98: tc.c_iflag &= ~INPCK; /* disable parity checking */
99: tc.c_iflag &= ~ISTRIP; /* disable stripping 8th bit */
100: tc.c_iflag &= ~(INLCR | ICRNL); /* disable translating NL <-> CR */
101: tc.c_iflag &= ~IGNCR; /* disable ignoring CR */
102: tc.c_iflag &= ~(IXON | IXOFF); /* disable XON/XOFF flow control */
103: /* output flags */
104: tc.c_oflag &= ~OPOST; /* disable output processing */
105: tc.c_oflag &= ~(ONLCR | OCRNL); /* disable translating NL <-> CR */
106: tc.c_oflag &= ~OFILL; /* disable fill characters */
107: /* control flags */
108: tc.c_cflag |= CREAD; /* enable reciever */
109: tc.c_cflag &= ~PARENB; /* disable parity */
110: tc.c_cflag &= ~CSTOPB; /* disable 2 stop bits */
111: tc.c_cflag &= ~CSIZE; /* remove size flag... */
112: tc.c_cflag |= CS8; /* ...enable 8 bit characters */
113: tc.c_cflag |= HUPCL; /* enable lower control lines on close - hang up */
114: /* local flags */
115: tc.c_lflag &= ~ISIG; /* disable generating signals */
116: tc.c_lflag &= ~ICANON; /* disable canonical mode - line by line */
117: tc.c_lflag &= ~ECHO; /* disable echoing characters */
118: tc.c_lflag &= ~NOFLSH; /* disable flushing on SIGINT */
119: tc.c_lflag &= ~IEXTEN; /* disable input processing */
120: tcsetattr(xbee.ttyfd, TCSANOW, &tc);
121:
122:
123: if ((xbee.tty = fdopen(xbee.ttyfd, "r+")) == NULL) {
124:     perror("xbee_setup():fdopen()");
125:     xbee.path = NULL;
126:     close(xbee.ttyfd);
127:     xbee.ttyfd = -1;
128:     xbee.tty = NULL;
129:     exit(1);
130: }
131:
132: fflush(xbee.tty);
133:
134: /* now im ready */
135: xbee_ready = 1;
136:
137: /* can start xbee_listen thread now */
138: if (pthread_create(&xbee.listent, NULL, (void (*)(void *))xbee_listen, (void *)&info) != 0) {
139:     perror("xbee_setup():pthread_create()");
140:     exit(1);
141: }
142: }
143:
144: /* #####
145: xbee_con
146: produces a connection to the specified device and frameID
147: if a connection had already been made, then this connection will be returned */
148: xbee_con *xbee_newcon(unsigned char *tAddr, unsigned char frameID, xbee_types type) {
149:     xbee_con *con, *ocon;
150: #ifdef DEBUG
151:     int i;
152: #endif
153:
154:     ISREADY;
155:
156:     if (!type || type == xbee_unknown) type = xbee_localAT;
157:     else if (type == xbee_remoteAT) type = xbee_16bitRemoteAT;
158:
159:     pthread_mutex_lock(&xbee.conmutex);
160:
161:     if (xbee.conlist) {
162:         con = xbee.conlist;
163:         while (con) {
164:             if ((type == con->type) &&
165:                 (frameID == con->frameID)) {
166:
167:                 if (type == xbee_localAT) {
168:                     /* already has connection to local modem with that frameID */
169:                     pthread_mutex_unlock(&xbee.conmutex);
170:                     return con;

```

```

171:         } else if ((type == con->type) &&
172:
173:                 (((type == xbee_16bitRemoteAT) ||
174:                  (type == xbee_16bitData) ||
175:                  (type == xbee_16bitIO)) &&
176:                 (!memcmp(tAddr, con->tAddr, 2))) ||
177:
178:                 (((type == xbee_64bitRemoteAT) ||
179:                  (type == xbee_64bitData) ||
180:                  (type == xbee_64bitIO)) &&
181:                 (!memcmp(tAddr, con->tAddr, 8)))) {
182:         /* addressing modes & addresses match */
183:         pthread_mutex_unlock(&xbee.conmutex);
184:         return con;
185:     }
186: }
187: if (con->next == NULL) break;
188: con = con->next;
189: }
190: ocon = con;
191: }
192: con = Xcalloc(sizeof(xbee_con));
193: con->type = type;
194: if ((type == xbee_64bitRemoteAT) ||
195:     (type == xbee_64bitData) ||
196:     (type == xbee_64bitIO)) {
197:     con->tAddr64 = TRUE;
198: }
199: con->atQueue = 0;
200: con->txDisableACK = 0;
201: con->txBroadcast = 0;
202: con->frameID = frameID;
203: if (type != xbee_localAT) {
204:     if (con->tAddr64) {
205:         memcpy(con->tAddr, tAddr, 8);
206:     } else {
207:         memcpy(con->tAddr, tAddr, 2);
208:         memset(&con->tAddr[2], 0, 6);
209:     }
210: }
211:
212: #ifdef DEBUG
213: switch(type) {
214: case xbee_localAT:
215:     printf("XBee: New local AT connection!\n");
216:     break;
217: case xbee_16bitRemoteAT:
218: case xbee_64bitRemoteAT:
219:     printf("XBee: New %d-bit remote AT connection! (to: ", (con->tAddr64?64:16));
220:     for (i=0; i<(con->tAddr64?8:2); i++) {
221:         printf((i?"%02X":"%02X"), tAddr[i]);
222:     }
223:     printf(")\n");
224:     break;
225: case xbee_16bitData:
226: case xbee_64bitData:
227:     printf("XBee: New %d-bit data connection! (to: ", (con->tAddr64?64:16));
228:     for (i=0; i<(con->tAddr64?8:2); i++) {
229:         printf((i?"%02X":"%02X"), tAddr[i]);
230:     }
231:     printf(")\n");
232:     break;
233: case xbee_16bitIO:
234: case xbee_64bitIO:
235:     printf("XBee: New %d-bit IO connection! (to: ", (con->tAddr64?64:16));
236:     for (i=0; i<(con->tAddr64?8:2); i++) {
237:         printf((i?"%02X":"%02X"), tAddr[i]);
238:     }
239:     printf(")\n");
240:     break;
241: case xbee_txStatus:
242:     printf("XBee: New status connection!\n");
243:     break;
244: case xbee_modemStatus:
245:     break;
246: case xbee_unknown:
247: default:
248:     printf("XBee: New unknown connection!\n");
249: }
250: #endif
251:
252: con->next = NULL;
253: if (xbee.conlist) {
254:     ocon->next = con;
255: } else {

```

```

256:     xbee.conlist = con;
257: }
258: pthread_mutex_unlock(&xbee.conmutex);
259: return con;
260: }
261:
262: /* #####
263: xbee_senddata
264: send the specified data to the provided connection */
265: int xbee_senddata(xbee_con *con, char *format, ...) {
266:     t_data *pkt;
267:     int i, length;
268:     unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
269:     unsigned char data[128]; /* ditto */
270:     va_list ap;
271:
272:     ISREADY;
273:
274:     va_start(ap, format);
275:     length = vsnprintf((char *)data, 128, format, ap);
276:     va_end(ap);
277:
278: #ifdef DEBUG
279:     printf("XBee: --- TX Packet -----\n");
280:     printf("XBee: Length: %d\n", length);
281:     for (i=0; i<length; i++) {
282:         printf("XBee: %3d | 0x%02X ", i, data[i]);
283:         if ((data[i] > 32) && (data[i] < 127)) printf("' %c'\n", data[i]); else printf(" _\n");
284:     }
285: #endif
286:
287:     if (!con) return -1;
288:     if (con->type == xbee_unknown) return -1;
289:
290:     /* #####
291:     /* local AT mode */
292:     if (con->type == xbee_localAT) {
293:         if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter */
294:         if (!con->atQueue) {
295:             buf[0] = 0x08;
296:         } else {
297:             buf[1] = 0x09;
298:         }
299:         buf[1] = con->frameID;
300:         for (i=0; i<length; i++) {
301:             buf[i+2] = data[i];
302:         }
303:         pkt = xbee_make_pkt(buf, i+2);
304:         xbee_send_pkt(pkt);
305:         return 1;
306:     }
307:     if ((con->type == xbee_16bitRemoteAT) ||
308:         (con->type == xbee_64bitRemoteAT)) {
309:         /* #####
310:         /* remote AT mode */
311:         buf[0] = 0x17;
312:         buf[1] = con->frameID;
313:         if (con->tAddr64) {
314:             memcpy(&buf[2], con->tAddr, 8);
315:             buf[10] = 0xFF;
316:             buf[11] = 0xFE;
317:         } else {
318:             memset(&buf[2], 0, 8);
319:             memcpy(&buf[10], con->tAddr, 2);
320:         }
321:         buf[12] = ((!con->atQueue)?0x02:0x00);
322:         for (i=0; i<length; i++) {
323:             buf[i+13] = data[i];
324:         }
325:         pkt = xbee_make_pkt(buf, i+13);
326:         xbee_send_pkt(pkt);
327:         return 1;
328:     } else if (con->type == xbee_64bitData) {
329:         /* #####
330:         /* 64bit Data */
331:         buf[0] = 0x00;
332:         buf[1] = con->frameID;
333:         memcpy(&buf[2], con->tAddr, 8);
334:         buf[10] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
335:         for (i=0; i<length; i++) {
336:             buf[i+11] = data[i];
337:         }
338:         pkt = xbee_make_pkt(buf, i+11);
339:         xbee_send_pkt(pkt);
340:         return 1;

```

```

341: } else if (con->type == xbee_16bitData) {
342: /* ##### */
343: /* 16bit Data */
344: buf[0] = 0x01;
345: buf[1] = con->frameID;
346: memcpy(&buf[2], con->tAddr, 2);
347: buf[4] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
348: for (i=0; i<length; i++) {
349:     buf[i+5] = data[i];
350: }
351: pkt = xbee_make_pkt(buf, i+5);
352: xbee_send_pkt(pkt);
353: return 1;
354: } else if ((con->type == xbee_64bitIO) ||
355:            (con->type == xbee_16bitIO)) {
356:     printf("***** TODO *****\n");
357: }
358: return 0;
359: }
360:
361: /* #####
362: xbee_getpacket
363: retrieves the next packet destined for the given connection
364: once the packet has been retrieved, it is removed for the list! */
365: xbee_pkt *xbee_getpacket(xbee_con *con) {
366:     xbee_pkt *l, *p, *q;
367: #ifdef DEBUG
368:     int c;
369:     printf("XBee: ---- Get Packet ----- \n");
370: #endif
371:     pthread_mutex_lock(&xbee.pktmutex);
372:
373:     if ((p = xbee.pktlist) == NULL) {
374:         pthread_mutex_unlock(&xbee.pktmutex);
375: #ifdef DEBUG
376:         printf("XBee: No packets available... \n");
377: #endif
378:         return NULL;
379:     }
380:
381:     l = NULL;
382:     q = NULL;
383:     do {
384:         if ((p->type == con->type) ||
385:             ((p->type == xbee_remoteAT) &&
386:              (con->type == xbee_16bitRemoteAT)) ||
387:             ((p->type == xbee_remoteAT) &&
388:              (con->type == xbee_64bitRemoteAT))) {
389:             if ((p->type == xbee_localAT) &&
390:                 (con->frameID == p->frameID)) ||
391:                 ((p->type == xbee_remoteAT) &&
392:                  (con->frameID == p->frameID)) ||
393:                 ((con->tAddr64 && !memcmp(con->tAddr, p->Addr64, 8)) ||
394:                  (!con->tAddr64 && !memcmp(con->tAddr, p->Addr16, 2)))) {
395:                 q = p;
396:                 break;
397:             }
398:         }
399:         l = p;
400:         p = p->next;
401:     } while (p);
402:
403:     if (!q) {
404:         pthread_mutex_unlock(&xbee.pktmutex);
405: #ifdef DEBUG
406:         printf("XBee: No packets available (for connection)... \n");
407: #endif
408:         return NULL;
409:     }
410:
411:     if (!l) {
412:         xbee.pktlist = p->next;
413:     } else {
414:         l->next = p->next;
415:     }
416:
417: #ifdef DEBUG
418:     printf("XBee: Got a packet\n");
419:     for (p = xbee.pktlist, c = 0; p; c++, p = p->next);
420:     printf("XBee: Packets left: %d\n", c);
421: #endif
422: }
423:
424:
425:

```

```

426: pthread_mutex_unlock(&xbee.pktmutex);
427:
428: return q;
429: }
430:
431: /* #####
432: xbee_listen - INTERNAL
433: the xbee_listen thread
434: reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
435: void xbee_listen(t_info *info) {
436:     unsigned char c, t, d[128];
437:     unsigned int l, i, s, o;
438: #ifdef DEBUG
439:     int j;
440: #endif
441:     xbee_pkt *p, *q, *po;
442:
443:     ISREADY;
444:
445:     while(1) {
446:
447:         c = xbee_getRawByte();
448:
449:         if (c != 0x7E) continue;
450: #ifdef DEBUG
451:         printf("XBee: ---- RX Packet =====--\nXBee: Got a packet!...\n");
452: #endif
453:
454:         l = xbee_getByte() << 8;
455:         l += xbee_getByte();
456:
457:         if (!l) continue;
458:
459: #ifdef DEBUG
460:         printf("XBee: Length: %d\n", l - 1);
461: #endif
462:
463:         t = xbee_getByte();
464:
465:         for (i=0, s=0; l>1 && i<128; l--, i++) {
466:             c = xbee_getByte();
467:             d[i] = c;
468:             s += c;
469: #ifdef DEBUG
470:             printf("XBee: %3d | 0x%02X ", i, c);
471:             if ((c > 32) && (c < 127)) printf("'"c'\n", c); else printf(" _\n");
472: #endif
473:         }
474:         i--; /* it went up too many times! */
475:         c = xbee_getByte();
476:         s += c;
477:         s &= 0xFF;
478: #ifdef DEBUG
479:         printf("XBee: Checksum: 0x%02X   Result: 0x%02X\n", c, s);
480: #endif
481:         if (l>1) {
482: #ifdef DEBUG
483:             printf("XBee: Didn't get whole packet... :(\n");
484: #endif
485:             continue;
486:         }
487:
488:         po = p = Xcalloc(sizeof(xbee_pkt));
489:         q = NULL;
490:         p->datalen = l;
491:
492:         /* #####
493:         /* modem status */
494:         if (t == 0x8A) {
495: #ifdef DEBUG
496:             printf("XBee: Packet type: Modem Status (0x8A)\n");
497:             printf("XBee: ");
498:             switch (d[0]) {
499:                 case 0x00: printf("Hardware reset"); break;
500:                 case 0x01: printf("Watchdog timer reset"); break;
501:                 case 0x02: printf("Associated"); break;
502:                 case 0x03: printf("Disassociated"); break;
503:                 case 0x04: printf("Synchronization lost"); break;
504:                 case 0x05: printf("Coordinator realignment"); break;
505:                 case 0x06: printf("Coordinator started"); break;
506:             }
507:             printf("...\n");
508: #endif
509:             p->type = xbee_modemStatus;
510:

```

```

511:     p->sAddr64 = FALSE;
512:     p->dataPkt = FALSE;
513:     p->txStatusPkt = FALSE;
514:     p->modemStatusPkt = TRUE;
515:     p->remoteATPkt = FALSE;
516:     p->IOPkt = FALSE;
517:
518:     p->datalen = 1;
519:     p->data[0] = d[0];
520:     /* ##### */
521:     /* local AT response */
522:     } else if (t == 0x88) {
523: #ifdef DEBUG
524:     printf("XBee: Packet type: Local AT Response (0x88)\n");
525:     printf("XBee: FrameID: 0x%02X\n",d[0]);
526:     printf("XBee: AT Command: %c%c\n",d[1],d[2]);
527:     if (d[3] == 0) printf("XBee: Status: OK\n");
528:     else if (d[3] == 1) printf("XBee: Status: Error\n");
529:     else if (d[3] == 2) printf("XBee: Status: Invalid Command\n");
530:     else if (d[3] == 3) printf("XBee: Status: Invalid Parameter\n");
531: #endif
532:     p->type = xbee_localAT;
533:
534:     p->sAddr64 = FALSE;
535:     p->dataPkt = FALSE;
536:     p->txStatusPkt = FALSE;
537:     p->modemStatusPkt = FALSE;
538:     p->remoteATPkt = FALSE;
539:     p->IOPkt = FALSE;
540:
541:     p->frameID = d[0];
542:     p->atCmd[0] = d[1];
543:     p->atCmd[1] = d[2];
544:
545:     p->status = d[3];
546:
547:     p->datalen = i-3;
548:     for (;i>3;i--) p->data[i-4] = d[i];
549:     /* ##### */
550:     /* remote AT response */
551:     } else if (t == 0x97) {
552: #ifdef DEBUG
553:     printf("XBee: Packet type: Remote AT Response (0x97)\n");
554:     printf("XBee: FrameID: 0x%02X\n",d[0]);
555:     printf("XBee: 64-bit Address: ");
556:     for (j=0;j<8;j++) {
557:         printf((j?"%02X ":"%02X"),d[1+j]);
558:     }
559:     printf("\n");
560:     printf("XBee: 16-bit Address: ");
561:     for (j=0;j<2;j++) {
562:         printf((j?"%02X ":"%02X"),d[9+j]);
563:     }
564:     printf("\n");
565:     printf("XBee: AT Command: %c%c\n",d[11],d[12]);
566:     if (d[13] == 0) printf("XBee: Status: OK\n");
567:     else if (d[13] == 1) printf("XBee: Status: Error\n");
568:     else if (d[13] == 2) printf("XBee: Status: Invalid Command\n");
569:     else if (d[13] == 3) printf("XBee: Status: Invalid Parameter\n");
570:     else if (d[13] == 4) printf("XBee: Status: No Response\n");
571: #endif
572:     p->type = xbee_remoteAT;
573:
574:     p->sAddr64 = FALSE;
575:     p->dataPkt = FALSE;
576:     p->txStatusPkt = FALSE;
577:     p->modemStatusPkt = FALSE;
578:     p->remoteATPkt = TRUE;
579:     p->IOPkt = FALSE;
580:
581:     p->frameID = d[0];
582:
583:     p->Addr64[0] = d[1];
584:     p->Addr64[1] = d[2];
585:     p->Addr64[2] = d[3];
586:     p->Addr64[3] = d[4];
587:     p->Addr64[4] = d[5];
588:     p->Addr64[5] = d[6];
589:     p->Addr64[6] = d[7];
590:     p->Addr64[7] = d[8];
591:
592:     p->Addr16[0] = d[9];
593:     p->Addr16[1] = d[10];
594:
595:     p->atCmd[0] = d[11];

```

```

596:     p->atCmd[1] = d[12];
597:
598:     p->status = d[13];
599:
600:     p->datalen = i-13;
601:     for (;i>13;i--) p->data[i-14] = d[i];
602:     /* ##### */
603:     /* TX status */
604:     } else if (t == 0x89) {
605: #ifdef DEBUG
606:     printf("XBee: Packet type: TX Status Report (0x89)\n");
607:     printf("XBee: FrameID: 0x%02X\n",d[0]);
608:     if (d[1] == 0) printf("XBee: Status: Success\n");
609:     else if (d[1] == 1) printf("XBee: Status: No ACK\n");
610:     else if (d[1] == 2) printf("XBee: Status: CCA Failure\n");
611:     else if (d[1] == 3) printf("XBee: Status: Purged\n");
612: #endif
613:     p->type = xbee_txStatus;
614:
615:     p->sAddr64 = FALSE;
616:     p->dataPkt = FALSE;
617:     p->txStatusPkt = TRUE;
618:     p->modemStatusPkt = FALSE;
619:     p->remoteATPkt = FALSE;
620:     p->IOPkt = FALSE;
621:
622:     p->frameID = d[0];
623:
624:     p->status = d[1];
625:     /* ##### */
626:     /* 64bit address recieve */
627:     } else if (t == 0x80) {
628: #ifdef DEBUG
629:     printf("XBee: Packet type: 64-bit RX Data (0x80)\n");
630:     printf("XBee: 64-bit Address: ");
631:     for (j=0;j<8;j++) {
632:         printf((j?"%02X":"%02X"),d[j]);
633:     }
634:     printf("\n");
635:     printf("XBee: RSSI: -%ddB\n",d[8]);
636:     if (d[9] & 0x02) printf("XBee: Options: Address Broadcast\n");
637:     if (d[9] & 0x03) printf("XBee: Options: PAN Broadcast\n");
638: #endif
639:     p->type = xbee_64bitData;
640:
641:     p->sAddr64 = TRUE;
642:     p->dataPkt = TRUE;
643:     p->txStatusPkt = FALSE;
644:     p->modemStatusPkt = FALSE;
645:     p->remoteATPkt = FALSE;
646:     p->IOPkt = FALSE;
647:
648:     p->Addr64[0] = d[0];
649:     p->Addr64[1] = d[1];
650:     p->Addr64[2] = d[2];
651:     p->Addr64[3] = d[3];
652:     p->Addr64[4] = d[4];
653:     p->Addr64[5] = d[5];
654:     p->Addr64[6] = d[6];
655:     p->Addr64[7] = d[7];
656:
657:     p->RSSI = d[8];
658:
659:     p->status = d[9];
660:
661:     p->datalen = i-9;
662:     for (;i>9;i--) p->data[i-10] = d[i];
663:     /* ##### */
664:     /* 16bit address recieve */
665:     } else if (t == 0x81) {
666: #ifdef DEBUG
667:     printf("XBee: Packet type: 16-bit RX Data (0x81)\n");
668:     printf("XBee: 16-bit Address: ");
669:     for (j=0;j<2;j++) {
670:         printf((j?"%02X":"%02X"),d[j]);
671:     }
672:     printf("\n");
673:     printf("XBee: RSSI: -%ddB\n",d[2]);
674:     if (d[3] & 0x02) printf("XBee: Options: Address Broadcast\n");
675:     if (d[3] & 0x03) printf("XBee: Options: PAN Broadcast\n");
676: #endif
677:     p->type = xbee_16bitData;
678:
679:     p->sAddr64 = FALSE;
680:     p->dataPkt = TRUE;

```



```

681:     p->txStatusPkt = FALSE;
682:     p->modemStatusPkt = FALSE;
683:     p->remoteATPkt = FALSE;
684:     p->IOPkt = FALSE;
685:
686:     p->Addr16[0] = d[0];
687:     p->Addr16[1] = d[1];
688:
689:     p->RSSI = d[2];
690:
691:     p->status = d[3];
692:
693:     p->datalen = i-3;
694:     for (;i>3;i--) p->data[i-4] = d[i];
695:     /* ##### */
696:     /* 64bit I/O recieve */
697:     } else if (t == 0x82) {
698: #ifdef DEBUG
699:     printf("XBee: Packet type: 64-bit RX I/O Data (0x82)\n");
700:     printf("XBee: 64-bit Address: ");
701:     for (j=0;j<8;j++) {
702:         printf((j?"%02X":"%02X"),d[j]);
703:     }
704:     printf("\n");
705:     printf("XBee: RSSI: -%dB\n",d[8]);
706:     if (d[9] & 0x02) printf("XBee: Options: Address Broadcast\n");
707:     if (d[9] & 0x02) printf("XBee: Options: PAN Broadcast\n");
708:     printf("XBee: Samples: %d\n",d[10]);
709: #endif
710:     i = 13;
711:
712:     for (o=d[10];o>0;o--) {
713: #ifdef DEBUG
714:         printf("XBee: --- Sample %3d ----- \n",o-d[10]+1);
715: #endif
716:         if (o<d[10]) {
717:             q = Xcalloc(sizeof(xbee_pkt));
718:             p->next = q;
719:             p = q;
720:             p->datalen = 1;
721:         }
722:
723:         p->type = xbee_64bitIO;
724:
725:         p->sAddr64 = TRUE;
726:         p->dataPkt = FALSE;
727:         p->txStatusPkt = FALSE;
728:         p->modemStatusPkt = FALSE;
729:         p->remoteATPkt = FALSE;
730:         p->IOPkt = TRUE;
731:
732:         p->Addr64[0] = d[0];
733:         p->Addr64[1] = d[1];
734:         p->Addr64[2] = d[2];
735:         p->Addr64[3] = d[3];
736:         p->Addr64[4] = d[4];
737:         p->Addr64[5] = d[5];
738:         p->Addr64[6] = d[6];
739:         p->Addr64[7] = d[7];
740:
741:         p->RSSI = d[8];
742:
743:         p->status = d[9];
744:
745:         p->IOMask = (((d[11]<<8) | d[12]) & 0x7FFF);
746:         p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);
747:
748:         i += (((d[11]&0x01)|(d[12]))?2:0);
749:
750:         if (d[11]&0x02) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
751:         if (d[11]&0x04) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
752:         if (d[11]&0x08) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
753:         if (d[11]&0x10) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
754:         if (d[11]&0x20) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
755:         if (d[11]&0x40) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
756: #ifdef DEBUG
757:         if (p->IOMask & 0x0001) printf("XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
758:         if (p->IOMask & 0x0002) printf("XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
759:         if (p->IOMask & 0x0004) printf("XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
760:         if (p->IOMask & 0x0008) printf("XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
761:         if (p->IOMask & 0x0010) printf("XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
762:         if (p->IOMask & 0x0020) printf("XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
763:         if (p->IOMask & 0x0040) printf("XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
764:         if (p->IOMask & 0x0080) printf("XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));
765:         if (p->IOMask & 0x0100) printf("XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));

```

```

766:         if (p->IOMask & 0x0200) printf("XBee: Analog 0: %.2fv\n", (3.3/1023)*p->IOanalog[0]);
767:         if (p->IOMask & 0x0400) printf("XBee: Analog 1: %.2fv\n", (3.3/1023)*p->IOanalog[1]);
768:         if (p->IOMask & 0x0800) printf("XBee: Analog 2: %.2fv\n", (3.3/1023)*p->IOanalog[2]);
769:         if (p->IOMask & 0x1000) printf("XBee: Analog 3: %.2fv\n", (3.3/1023)*p->IOanalog[3]);
770:         if (p->IOMask & 0x2000) printf("XBee: Analog 4: %.2fv\n", (3.3/1023)*p->IOanalog[4]);
771:         if (p->IOMask & 0x4000) printf("XBee: Analog 5: %.2fv\n", (3.3/1023)*p->IOanalog[5]);
772:     #endif
773: }
774: #ifdef DEBUG
775:     printf("XBee: ----- \n");
776: #endif
777: } else if (t == 0x83) {
778:     /* ##### */
779:     /* 16bit I/O recieve */
780:     #ifdef DEBUG
781:         printf("XBee: Packet type: 16-bit RX I/O Data (0x83)\n");
782:         printf("XBee: 64-bit Address: ");
783:         for (j=0; j<2; j++) {
784:             printf((j?"%02X":"%02X"), d[j]);
785:         }
786:         printf("\n");
787:         printf("XBee: RSSI: -%dB\n", d[2]);
788:         if (d[3] & 0x02) printf("XBee: Options: Address Broadcast\n");
789:         if (d[3] & 0x02) printf("XBee: Options: PAN Broadcast\n");
790:         printf("XBee: Samples: %d\n", d[4]);
791:     #endif
792:
793:     i = 7;
794:
795:     for (o=d[4]; o>0; o--) {
796:         #ifdef DEBUG
797:             printf("XBee: --- Sample %3d ----- \n", o-d[4]+1);
798:         #endif
799:         if (o<d[4]) {
800:             q = Xcalloc(sizeof(xbee_pkt));
801:             p->next = q;
802:             p = q;
803:             p->datalen = 1;
804:         }
805:
806:         p->type = xbee_16bitIO;
807:
808:         p->sAddr64 = FALSE;
809:         p->dataPkt = FALSE;
810:         p->txStatusPkt = FALSE;
811:         p->modemStatusPkt = FALSE;
812:         p->remoteATPkt = FALSE;
813:         p->IOPkt = TRUE;
814:
815:         p->Addr16[0] = d[0];
816:         p->Addr16[1] = d[1];
817:
818:         p->RSSI = d[2];
819:
820:         p->status = d[3];
821:
822:         p->IOMask = (((d[5]<<8) | d[6]) & 0x7FFF);
823:         p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);
824:
825:         i += (((d[5]&0x01) | (d[6]))?2:0);
826:
827:         if (d[5]&0x02) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF); i+=2;}
828:         if (d[5]&0x04) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF); i+=2;}
829:         if (d[5]&0x08) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF); i+=2;}
830:         if (d[5]&0x10) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF); i+=2;}
831:         if (d[5]&0x20) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF); i+=2;}
832:         if (d[5]&0x40) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF); i+=2;}
833:     #ifdef DEBUG
834:         if (p->IOMask & 0x0001) printf("XBee: Digital 0: %c\n", ((p->IOdata & 0x0001)?'1':'0'));
835:         if (p->IOMask & 0x0002) printf("XBee: Digital 1: %c\n", ((p->IOdata & 0x0002)?'1':'0'));
836:         if (p->IOMask & 0x0004) printf("XBee: Digital 2: %c\n", ((p->IOdata & 0x0004)?'1':'0'));
837:         if (p->IOMask & 0x0008) printf("XBee: Digital 3: %c\n", ((p->IOdata & 0x0008)?'1':'0'));
838:         if (p->IOMask & 0x0010) printf("XBee: Digital 4: %c\n", ((p->IOdata & 0x0010)?'1':'0'));
839:         if (p->IOMask & 0x0020) printf("XBee: Digital 5: %c\n", ((p->IOdata & 0x0020)?'1':'0'));
840:         if (p->IOMask & 0x0040) printf("XBee: Digital 6: %c\n", ((p->IOdata & 0x0040)?'1':'0'));
841:         if (p->IOMask & 0x0080) printf("XBee: Digital 7: %c\n", ((p->IOdata & 0x0080)?'1':'0'));
842:         if (p->IOMask & 0x0100) printf("XBee: Digital 8: %c\n", ((p->IOdata & 0x0100)?'1':'0'));
843:         if (p->IOMask & 0x0200) printf("XBee: Analog 0: %.2fv\n", (3.3/1023)*p->IOanalog[0]);
844:         if (p->IOMask & 0x0400) printf("XBee: Analog 1: %.2fv\n", (3.3/1023)*p->IOanalog[1]);
845:         if (p->IOMask & 0x0800) printf("XBee: Analog 2: %.2fv\n", (3.3/1023)*p->IOanalog[2]);
846:         if (p->IOMask & 0x1000) printf("XBee: Analog 3: %.2fv\n", (3.3/1023)*p->IOanalog[3]);
847:         if (p->IOMask & 0x2000) printf("XBee: Analog 4: %.2fv\n", (3.3/1023)*p->IOanalog[4]);
848:         if (p->IOMask & 0x4000) printf("XBee: Analog 5: %.2fv\n", (3.3/1023)*p->IOanalog[5]);
849:     #endif
850: }

```

```

851: #ifdef DEBUG
852:     printf("XBee: -----\\n");
853: #endif
854:     /* ##### */
855:     /* Unknown */
856: } else {
857: #ifdef DEBUG
858:     printf("XBee: Packet type: Unknown (0x%02X)\\n",t);
859: #endif
860:     p->type = xbee_unknown;
861: }
862: p->next = NULL;
863:
864: pthread_mutex_lock(&xbee.pktmutex);
865: i = 1;
866: if (!xbee.pktlist) {
867:     xbee.pktlist = po;
868: } else {
869:     q = xbee.pktlist;
870:     while (q->next) {
871:         q = q->next;
872:         i++;
873:     }
874:     q->next = po;
875: }
876:
877: #ifdef DEBUG
878:     while (q && q->next) {
879:         q = q->next;
880:         i++;
881:     }
882:     printf("XBee: -----\\n");
883:     printf("XBee: Packets: %d\\n",i);
884: #endif
885:
886:     po = p = q = NULL;
887:     pthread_mutex_unlock(&xbee.pktmutex);
888: }
889: }
890:
891: /* #####
892: xbee_getByte - INTERNAL
893: waits for an escaped byte of data */
894: unsigned char xbee_getByte(void) {
895:     unsigned char c;
896:
897:     ISREADY;
898:
899:     c = xbee_getRawByte();
900:     if (c == 0x7D) c = xbee_getRawByte() ^ 0x20;
901:
902:     return (c & 0xFF);
903: }
904:
905: /* #####
906: xbee_getRawByte - INTERNAL
907: waits for a raw byte of data */
908: unsigned char xbee_getRawByte(void) {
909:     unsigned char c;
910:     fd_set fds;
911:
912:     ISREADY;
913:
914:     FD_ZERO(&fds);
915:     FD_SET(xbee.ttyfd,&fds);
916:
917:     if (select(xbee.ttyfd+1,&fds,NULL,NULL,NULL) == -1) {
918:         perror("xbee:xbee_listen():xbee_getByte()");
919:         exit(1);
920:     }
921:
922:     do {
923:         if (read(xbee.ttyfd,&c,1) == 0) {
924:             usleep(10);
925:             continue;
926:         }
927:     } while (0);
928:
929:     return (c & 0xFF);
930: }
931:
932: /* #####
933: xbee_send_pkt - INTERNAL
934: sends a complete packet of data */
935: void xbee_send_pkt(t_data *pkt) {

```

```

936:     ISREADY;
937:
938:     /* write and flush the data */
939:     fwrite(pkt->data,pkt->length,1,xbee.tty);
940:     fflush(xbee.tty);
941:
942: #ifdef DEBUG
943:     {
944:         int i;
945:         /* prints packet in hex byte-by-byte */
946:         printf("XBee: TX Packet - ");
947:         for (i=0;i<pkt->length;i++) {
948:             printf("0x%02X ",pkt->data[i]);
949:         }
950:         printf("\n");
951:     }
952: #endif
953:
954:     xbee_destroy_pkt(pkt);
955: }
956:
957: /* #####
958: xbex_make_pkt - INTERNAL
959: adds delimiter field
960: calculates length and checksum
961: escapes bytes */
962: t_data *xbex_make_pkt(unsigned char *data, int length) {
963:     t_data *pkt;
964:     unsigned int l, i, o, t, x, m;
965:     char d = 0;
966:
967:     ISREADY;
968:
969:     /* check the data given isnt too long */
970:     if (length > 0xFFFF) return NULL;
971:
972:     /* calculate the length of the whole packet */
973:     l = 3 + length + 1;
974:
975:     /* prepare memory */
976:     pkt = Xmalloc(sizeof(t_data));
977:
978:     /* put start byte on */
979:     pkt->data[0] = 0x7E;
980:
981:     /* copy data into packet */
982:     for (t=0,i=0,o=1,m=1;i<=length;o++,m++) {
983:         if (i == length) {
984:             d = M8((0xFF - M8(t)));
985:         }
986:         else if (m == 1) d = M8(length >> 8);
987:         else if (m == 2) d = M8(length);
988:         else if (m > 2) d = data[i];
989:         x = 0;
990:         /* check for any escapes needed */
991:         if ((d == 0x11) || /* XON */
992:             (d == 0x13) || /* XOFF */
993:             (d == 0x7D) || /* Escape */
994:             (d == 0x7E)) { /* Frame Delimiter */
995:             l++;
996:             pkt->data[o++] = 0x7D;
997:             x = 1;
998:         }
999:
1000:         /* move data in */
1001:         pkt->data[o] = (!x)?(d):(d^0x20);
1002:         if (m > 2) {
1003:             i++;
1004:             t += d;
1005:         }
1006:     }
1007:
1008:     /* remember the length */
1009:     pkt->length = l;
1010:
1011:     return pkt;
1012: }
1013:
1014: /* #####
1015: xbex_destroy_pkt - INTERNAL
1016: free's the packet memory */
1017: void xbee_destroy_pkt(t_data *pkt) {
1018:
1019:     ISREADY;
1020:

```

```
1021:  /* free the stuff! */  
1022:  Xfree(pkt);  
1023: }
```