```c
#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 */
void xbee_setup(char *path) {
  t_info info;
  struct termios tc;

  xbee.conlist = NULL;
  if (pthread_mutex_init(&xbee.conmutex,NULL)) {
    perror("xbee_setup():pthread_mutex_init(conmutex)");
    exit(1);
  }

  xbee.pktlist = NULL;
  if (pthread_mutex_init(&xbee.pktmutex,NULL)) {
    perror("xbee_setup():pthread_mutex_init(pktmutex)");
    exit(1);
  }

  xbee.path = path;

  /* open the serial port */
  if ((xbee.ttyfd = open(path,O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1) {
    perror("xbee_setup():open()");
    xbee.path = NULL;
    xbee.ttyfd = -1;
    xbee.tty = NULL;
    exit(1);
  }
  /* setup the baud rate - 57600 8N1*/
  tcgetattr(xbee.ttyfd, &tc);
  cfsetispeed(&tc, B57600);       /* set input baud rate to 57600 */
  cfsetospeed(&tc, B57600);       /* set output baud rate to 57600 */
  /* input flags */
  tc.c_iflag |= IGNBRK;           /* enable ignoring break */
  tc.c_iflag &= ~(IGNPAR | PARMRK);/* disable parity checks */
  tc.c_iflag &= ~INPCK;           /* disable parity checking */
  tc.c_iflag &= ~ISTRIP;          /* disable stripping 8th bit */
  tc.c_iflag &= ~(INLCR | ICRNL); /* disable translating NL <-> CR */
  tc.c_iflag &= ~IGNCR;           /* disable ignoring CR */
  tc.c_iflag &= ~(IXON | IXOFF);  /* disable XON/XOFF flow control */
  /* output flags */
  tc.c_oflag &= ~OPOST;           /* disable output processing */
  tc.c_oflag &= ~(ONLCR | OCRNL); /* disable translating NL <-> CR */
  tc.c_oflag &= ~OFILL;           /* disable fill characters */
  /* control flags */
  tc.c_cflag |= CREAD;            /* enable reciever */
  tc.c_cflag &= ~PARENB;          /* disable parity */
  tc.c_cflag &= ~CSTOPB;          /* disable 2 stop bits */
  tc.c_cflag &= ~CSIZE;           /* remove size flag... */
  tc.c_cflag |= CS8;              /* ...enable 8 bit characters */
  tc.c_cflag |= HUPCL;            /* enable lower control lines on close - hang up */
  /* local flags */
  tc.c_lflag &= ~ISIG;            /* disable generating signals */
  tc.c_lflag &= ~ICANON;          /* disable canonical mode - line by line */
  tc.c_lflag &= ~ECHO;            /* disable echoing characters */
  tc.c_lflag &= ~NOFLSH;          /* disable flushing on SIGINT */
  tc.c_lflag &= ~IEXTEN;          /* disable input processing */
  tcsetattr(xbee.ttyfd, TCSANOW, &tc);


  if ((xbee.tty = fdopen(xbee.ttyfd,"r+")) == NULL) {
    perror("xbee_setup():fdopen()");
    xbee.path = NULL;
    close(xbee.ttyfd);
    xbee.ttyfd = -1;
    xbee.tty = NULL;
    exit(1);
  }

  fflush(xbee.tty);

  /* now im ready */
  xbee_ready = 1;

  /* can start xbee_listen thread now */
  if (pthread_create(&xbee.listent,NULL,(void *(*)(void *))xbee_listen,(void *)&info) != 0) {
    perror("xbee_setup():pthread_create()");
    exit(1);
  }
}

/* #################################################################
   xbee_con
   produces a connection to the specified device and frameID
   if a connection had already been made, then this connection will be returned */
xbee_con *xbee_newcon(unsigned char *tAddr, unsigned char frameID, xbee_types type) {
  xbee_con *con, *ocon;
#ifdef DEBUG
  int i;
#endif

  ISREADY;

  if (!type || type == xbee_unknown) type = xbee_localAT;
  else if (type == xbee_remoteAT) type = xbee_16bitRemoteAT;

  pthread_mutex_lock(&xbee.conmutex);

  if (xbee.conlist) {
    con = xbee.conlist;
    while (con) {
      if ((type == con->type) &&
          (frameID == con->frameID)) {

        if (type == xbee_localAT) {
          /* already has connection to local modem with that frameID */
          pthread_mutex_unlock(&xbee.conmutex);
          return con;
        } else if ((type == con->type) &&

                   ((((type == xbee_16bitRemoteAT) ||
                      (type == xbee_16bitData) ||
                      (type == xbee_16bitIO)) &&
                     (!memcmp(tAddr,con->tAddr,2))) ||

                    (((type == xbee_64bitRemoteAT) ||
                      (type == xbee_64bitData) ||
                      (type == xbee_64bitIO)) &&
                     (!memcmp(tAddr,con->tAddr,8))))) {
          /* addressing modes & addresses match */
          pthread_mutex_unlock(&xbee.conmutex);
          return con;
        }
      }
      if (con->next == NULL) break;
      con = con->next;
    }
    ocon = con;
  }
  con = Xcalloc(sizeof(xbee_con));
  con->type = type;
  if ((type == xbee_64bitRemoteAT) ||
      (type == xbee_64bitData) ||
      (type == xbee_64bitIO)) {
    con->tAddr64 = TRUE;
  }
  con->atQueue = 0;
  con->txDisableACK = 0;
  con->txBroadcast = 0;
  con->frameID = frameID;
  if (type != xbee_localAT) {
    if (con->tAddr64) {
      memcpy(con->tAddr,tAddr,8);
    } else {
      memcpy(con->tAddr,tAddr,2);
      memset(&con->tAddr[2],0,6);
    }
  }

#ifdef DEBUG
  switch(type) {
  case xbee_localAT:
    printf("XBee: New local AT connection!\n");
    break;
  case xbee_16bitRemoteAT:
  case xbee_64bitRemoteAT:
    printf("XBee: New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
    for (i=0;i<(con->tAddr64?8:2);i++) {
      printf((i?":%02X":"%02X"),tAddr[i]);
    }
    printf(")\n");
    break;
  case xbee_16bitData:
  case xbee_64bitData:
    printf("XBee: New %d-bit data connection! (to: ",(con->tAddr64?64:16));
    for (i=0;i<(con->tAddr64?8:2);i++) {
      printf((i?":%02X":"%02X"),tAddr[i]);
    }
    printf(")\n");
    break;
  case xbee_16bitIO:
  case xbee_64bitIO:
    printf("XBee: New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
    for (i=0;i<(con->tAddr64?8:2);i++) {
      printf((i?":%02X":"%02X"),tAddr[i]);
    }
    printf(")\n");
    break;
  case xbee_txStatus:
    printf("XBee: New status connection!\n");
    break;
  case xbee_modemStatus:
    break;
  case xbee_unknown:
  default:
    printf("XBee: New unknown connection!\n");
  }
#endif

  con->next = NULL;
  if (xbee.conlist) {
    ocon->next = con;
  } else {
    xbee.conlist = con;
  }
  pthread_mutex_unlock(&xbee.conmutex);
  return con;
}

/* #################################################################
   xbee_senddata
   send the specified data to the provided connection */
int xbee_senddata(xbee_con *con, char *format, ...) {
  t_data *pkt;
  int i, length;
  unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
  unsigned char data[128]; /* ditto */
  va_list ap;

  ISREADY;

  va_start(ap, format);
  length = vsnprintf((char *)data,128,format,ap);
  va_end(ap);

#ifdef DEBUG
  printf("XBee: --== TX Packet ============--\n");
  printf("XBee: Length: %d\n",length);
  for (i=0;i<length;i++) {
    printf("XBee: %3d | 0x%02X ",i,data[i]);
    if ((data[i] > 32) && (data[i] < 127)) printf("'%c'\n",data[i]); else printf(" _\n");
  }
#endif

  if (!con) return -1;
  if (con->type == xbee_unknown) return -1;

  /* ########################################## */
  /* local AT mode */
  if (con->type == xbee_localAT) {
    if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter */
    if (!con->atQueue) {
      buf[0] = 0x08;
    } else {
      buf[1] = 0x09;
    }
    buf[1] = con->frameID;
    for (i=0;i<length;i++) {
      buf[i+2] = data[i];
    }
    pkt = xbee_make_pkt(buf,i+2);
    xbee_send_pkt(pkt);
    return 1;
  }
  if ((con->type == xbee_16bitRemoteAT) ||
      (con->type == xbee_64bitRemoteAT)) {
    /* ########################################## */
    /* remote AT mode */
    buf[0] = 0x17;
    buf[1] = con->frameID;
    if (con->tAddr64) {
      memcpy(&buf[2],con->tAddr,8);
      buf[10] = 0xFF;
      buf[11] = 0xFE;
    } else {
      memset(&buf[2],0,8);
      memcpy(&buf[10],con->tAddr,2);
    }
    buf[12] = ((!con->atQueue)?0x02:0x00);
    for (i=0;i<length;i++) {
      buf[i+13] = data[i];
    }
    pkt = xbee_make_pkt(buf,i+13);
    xbee_send_pkt(pkt);
    return 1;
  } else if (con->type == xbee_64bitData) {
    /* ########################################## */
    /* 64bit Data */
    buf[0] = 0x00;
    buf[1] = con->frameID;
    memcpy(&buf[2],con->tAddr,8);
    buf[10] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
    for (i=0;i<length;i++) {
      buf[i+11] = data[i];
    }
    pkt = xbee_make_pkt(buf,i+11);
    xbee_send_pkt(pkt);
    return 1;
  } else if (con->type == xbee_16bitData) {
    /* ########################################## */
    /* 16bit Data */
    buf[0] = 0x01;
    buf[1] = con->frameID;
    memcpy(&buf[2],con->tAddr,2);
    buf[4] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
    for (i=0;i<length;i++) {
      buf[i+5] = data[i];
    }
    pkt = xbee_make_pkt(buf,i+5);
    xbee_send_pkt(pkt);
    return 1;
  } else if ((con->type == xbee_64bitIO) ||
             (con->type == xbee_16bitIO)) {
    printf("******* TODO ********\n");
  }
  return 0;
}

/* ################################################################
   xbee_getpacket
   retrieves the next packet destined for the given connection
   once the packet has been retrieved, it is removed for the list! */
xbee_pkt *xbee_getpacket(xbee_con *con) {
  xbee_pkt *l, *p, *q;
#ifdef DEBUG
  int c;
  printf("XBee: --== Get Packet ==========--\n");
#endif

  pthread_mutex_lock(&xbee.pktmutex);

  if ((p = xbee.pktlist) == NULL) {
    pthread_mutex_unlock(&xbee.pktmutex);
#ifdef DEBUG
    printf("XBee: No packets avaliable...\n");
#endif
    return NULL;
  }

  l = NULL;
  q = NULL;
  do {
    if ((p->type == con->type) ||
        ((p->type == xbee_remoteAT) &&
         (con->type == xbee_16bitRemoteAT)) ||
        ((p->type == xbee_remoteAT) &&
         (con->type == xbee_64bitRemoteAT))) {
      if (((p->type == xbee_localAT) &&
           (con->frameID == p->frameID)) ||

          ((p->type == xbee_remoteAT) &&
           (con->frameID == p->frameID)) ||

          ((con->tAddr64 && !memcmp(con->tAddr,p->Addr64,8)) ||
           (!con->tAddr64 && !memcmp(con->tAddr,p->Addr16,2)))) {
        q = p;
        break;
      }
    }
    l = p;
    p = p->next;
  } while (p);

  if (!q) {
    pthread_mutex_unlock(&xbee.pktmutex);
#ifdef DEBUG
    printf("XBee: No packets avaliable (for connection)...\n");
#endif
    return NULL;
  }

  if (!l) {
    xbee.pktlist = p->next;
  } else {
    l->next = p->next;
  }

#ifdef DEBUG
  printf("XBee: Got a packet\n");
  for (p = xbee.pktlist,c = 0;p;c++,p = p->next);
  printf("XBee: Packets left: %d\n",c);
#endif

```

```c
  pthread_mutex_unlock(&xbee.pktmutex);

  return q;
}

/* ##################################################################
   xbee_listen - INTERNAL
   the xbee xbee_listen thread
   reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
void xbee_listen(t_info *info) {
  unsigned char c, t, d[128];
  unsigned int l, i, s, o;
#ifdef DEBUG
  int j;
#endif
  xbee_pkt *p, *q, *po;

  ISREADY;

  while(1) {

    c = xbee_getRawByte();

    if (c != 0x7E) continue;
#ifdef DEBUG
    printf("XBee: --== RX Packet ===========--\nXBee: Got a packet!...\n");
#endif

    l = xbee_getByte() << 8;
    l += xbee_getByte();

    if (!l) continue;

#ifdef DEBUG
    printf("XBee: Length: %d\n",l - 1);
#endif

    t = xbee_getByte();

    for (i=0,s=0;l>1 && i<128;l--,i++) {
      c = xbee_getByte();
      d[i] = c;
      s += c;
#ifdef DEBUG
      printf("XBee: %3d | 0x%02X ",i,c);
      if ((c > 32) && (c < 127)) printf("'%c'\n",c); else printf(" _\n");
#endif
    }
    i--; /* it went up too many times! */
    c = xbee_getByte();
    s += c;
    s &= 0xFF;
#ifdef DEBUG
    printf("XBee: Checksum: 0x%02X   Result: 0x%02X\n",c,s);
#endif
    if (l>1) {
#ifdef DEBUG
      printf("XBee: Didn't get whole packet... :(\n");
#endif
      continue;
    }

    po = p = Xcalloc(sizeof(xbee_pkt));
    q = NULL;
    p->datalen = l;

    /* ########################################## */
    /* modem status */
    if (t == 0x8A) {
#ifdef DEBUG
      printf("XBee: Packet type: Modem Status (0x8A)\n");
      printf("XBee: ");
      switch (d[0]) {
      case 0x00: printf("Hardware reset"); break;
      case 0x01: printf("Watchdog timer reset"); break;
      case 0x02: printf("Associated"); break;
      case 0x03: printf("Disassociated"); break;
      case 0x04: printf("Synchronization lost"); break;
      case 0x05: printf("Coordinator realignment"); break;
      case 0x06: printf("Coordinator started"); break;
      }
      printf("...\n");
#endif
      p->type = xbee_modemStatus;

```

```
        p->sAddr64 = FALSE;
        p->dataPkt = FALSE;
        p->txStatusPkt = FALSE;
        p->modemStatusPkt = TRUE;
        p->remoteATPkt = FALSE;
        p->IOPkt = FALSE;

        p->datalen = 1;
        p->data[0] = d[0];
      /* ########################################## */
      /* local AT response */
      } else if (t == 0x88) {
#ifdef DEBUG
        printf("XBee: Packet type: Local AT Response (0x88)\n");
        printf("XBee: FrameID: 0x%02X\n",d[0]);
        printf("XBee: AT Command: %c%c\n",d[1],d[2]);
        if (d[3] == 0) printf("XBee: Status: OK\n");
        else if (d[3] == 1) printf("XBee: Status: Error\n");
        else if (d[3] == 2) printf("XBee: Status: Invalid Command\n");
        else if (d[3] == 3) printf("XBee: Status: Invalid Parameter\n");
#endif
        p->type = xbee_localAT;

        p->sAddr64 = FALSE;
        p->dataPkt = FALSE;
        p->txStatusPkt = FALSE;
        p->modemStatusPkt = FALSE;
        p->remoteATPkt = FALSE;
        p->IOPkt = FALSE;

        p->frameID = d[0];
        p->atCmd[0] = d[1];
        p->atCmd[1] = d[2];

        p->status = d[3];

        p->datalen = i-3;
        for (;i>3;i--) p->data[i-4] = d[i];
      /* ########################################## */
      /* remote AT response */
      } else if (t == 0x97) {
#ifdef DEBUG
        printf("XBee: Packet type: Remote AT Response (0x97)\n");
        printf("XBee: FrameID: 0x%02X\n",d[0]);
        printf("XBee: 64-bit Address: ");
        for (j=0;j<8;j++) {
          printf((j?":%02X":"%02X"),d[1+j]);
        }
        printf("\n");
        printf("XBee: 16-bit Address: ");
        for (j=0;j<2;j++) {
          printf((j?":%02X":"%02X"),d[9+j]);
        }
        printf("\n");
        printf("XBee: AT Command: %c%c\n",d[11],d[12]);
        if (d[13] == 0) printf("XBee: Status: OK\n");
        else if (d[13] == 1) printf("XBee: Status: Error\n");
        else if (d[13] == 2) printf("XBee: Status: Invalid Command\n");
        else if (d[13] == 3) printf("XBee: Status: Invalid Parameter\n");
        else if (d[13] == 4) printf("XBee: Status: No Response\n");
#endif
        p->type = xbee_remoteAT;

        p->sAddr64 = FALSE;
        p->dataPkt = FALSE;
        p->txStatusPkt = FALSE;
        p->modemStatusPkt = FALSE;
        p->remoteATPkt = TRUE;
        p->IOPkt = FALSE;

        p->frameID = d[0];

        p->Addr64[0] = d[1];
        p->Addr64[1] = d[2];
        p->Addr64[2] = d[3];
        p->Addr64[3] = d[4];
        p->Addr64[4] = d[5];
        p->Addr64[5] = d[6];
        p->Addr64[6] = d[7];
        p->Addr64[7] = d[8];

        p->Addr16[0] = d[9];
        p->Addr16[1] = d[10];

        p->atCmd[0] = d[11];
```

```c
      p->atCmd[1] = d[12];

      p->status = d[13];

      p->datalen = i-13;
      for (;i>13;i--) p->data[i-14] = d[i];
    /* ########################################### */
    /* TX status */
    } else if (t == 0x89) {
#ifdef DEBUG
      printf("XBee: Packet type: TX Status Report (0x89)\n");
      printf("XBee: FrameID: 0x%02X\n",d[0]);
      if (d[1] == 0) printf("XBee: Status: Success\n");
      else if (d[1] == 1) printf("XBee: Status: No ACK\n");
      else if (d[1] == 2) printf("XBee: Status: CCA Failure\n");
      else if (d[1] == 3) printf("XBee: Status: Purged\n");
#endif
      p->type = xbee_txStatus;

      p->sAddr64 = FALSE;
      p->dataPkt = FALSE;
      p->txStatusPkt = TRUE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = FALSE;
      p->IOPkt = FALSE;

      p->frameID = d[0];

      p->status = d[1];
    /* ########################################### */
    /* 64bit address recieve */
    } else if (t == 0x80) {
#ifdef DEBUG
      printf("XBee: Packet type: 64-bit RX Data (0x80)\n");
      printf("XBee: 64-bit Address: ");
      for (j=0;j<8;j++) {
        printf((j?":%02X":"%02X"),d[j]);
      }
      printf("\n");
      printf("XBee: RSSI: -%ddB\n",d[8]);
      if (d[9] & 0x02) printf("XBee: Options: Address Broadcast\n");
      if (d[9] & 0x03) printf("XBee: Options: PAN Broadcast\n");
#endif
      p->type = xbee_64bitData;

      p->sAddr64 = TRUE;
      p->dataPkt = TRUE;
      p->txStatusPkt = FALSE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = FALSE;
      p->IOPkt = FALSE;

      p->Addr64[0] = d[0];
      p->Addr64[1] = d[1];
      p->Addr64[2] = d[2];
      p->Addr64[3] = d[3];
      p->Addr64[4] = d[4];
      p->Addr64[5] = d[5];
      p->Addr64[6] = d[6];
      p->Addr64[7] = d[7];

      p->RSSI = d[8];

      p->status = d[9];

      p->datalen = i-9;
      for (;i>9;i--) p->data[i-10] = d[i];
    /* ########################################### */
    /* 16bit address recieve */
    } else if (t == 0x81) {
#ifdef DEBUG
      printf("XBee: Packet type: 16-bit RX Data (0x81)\n");
      printf("XBee: 16-bit Address: ");
      for (j=0;j<2;j++) {
        printf((j?":%02X":"%02X"),d[j]);
      }
      printf("\n");
      printf("XBee: RSSI: -%ddB\n",d[2]);
      if (d[3] & 0x02) printf("XBee: Options: Address Broadcast\n");
      if (d[3] & 0x03) printf("XBee: Options: PAN Broadcast\n");
#endif
      p->type = xbee_16bitData;

      p->sAddr64 = FALSE;
      p->dataPkt = TRUE;
      p->txStatusPkt = FALSE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = FALSE;
      p->IOPkt = FALSE;

      p->Addr16[0] = d[0];
      p->Addr16[1] = d[1];

      p->RSSI = d[2];

      p->status = d[3];

      p->datalen = i-3;
      for (;i>3;i--) p->data[i-4] = d[i];
    /* ########################################### */
    /* 64bit I/O recieve */
    } else if (t == 0x82) {
#ifdef DEBUG
      printf("XBee: Packet type: 64-bit RX I/O Data (0x82)\n");
      printf("XBee: 64-bit Address: ");
      for (j=0;j<8;j++) {
        printf((j?":%02X":"%02X"),d[j]);
      }
      printf("\n");
      printf("XBee: RSSI: -%ddB\n",d[8]);
      if (d[9] & 0x02) printf("XBee: Options: Address Broadcast\n");
      if (d[9] & 0x02) printf("XBee: Options: PAN Broadcast\n");
      printf("XBee: Samples: %d\n",d[10]);
#endif
      i = 13;

      for (o=d[10];o>0;o--) {
#ifdef DEBUG
        printf("XBee: --- Sample %3d -------------\n",o-d[10]+1);
#endif
        if (o<d[10]) {
          q = Xcalloc(sizeof(xbee_pkt));
          p->next = q;
          p = q;
          p->datalen = l;
        }

        p->type = xbee_64bitIO;

        p->sAddr64 = TRUE;
        p->dataPkt = FALSE;
        p->txStatusPkt = FALSE;
        p->modemStatusPkt = FALSE;
        p->remoteATPkt = FALSE;
        p->IOPkt = TRUE;

        p->Addr64[0] = d[0];
        p->Addr64[1] = d[1];
        p->Addr64[2] = d[2];
        p->Addr64[3] = d[3];
        p->Addr64[4] = d[4];
        p->Addr64[5] = d[5];
        p->Addr64[6] = d[6];
        p->Addr64[7] = d[7];

        p->RSSI = d[8];

        p->status = d[9];

        p->IOmask = (((d[11]<<8) | d[12]) & 0x7FFF);
        p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);

        i += (((d[11]&0x01)||(d[12]))?2:0);

        if (d[11]&0x02) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
        if (d[11]&0x04) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
        if (d[11]&0x08) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
        if (d[11]&0x10) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
        if (d[11]&0x20) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
        if (d[11]&0x40) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
#ifdef DEBUG
        if (p->IOmask & 0x0001) printf("XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
        if (p->IOmask & 0x0002) printf("XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
        if (p->IOmask & 0x0004) printf("XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
        if (p->IOmask & 0x0008) printf("XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
        if (p->IOmask & 0x0010) printf("XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
        if (p->IOmask & 0x0020) printf("XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
        if (p->IOmask & 0x0040) printf("XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
        if (p->IOmask & 0x0080) printf("XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));
        if (p->IOmask & 0x0100) printf("XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));
```

```c
        if (p->IOmask & 0x0200) printf("XBee: Analog  0: %.2fv\n",(3.3/1023)*p->IOanalog[0]);
        if (p->IOmask & 0x0400) printf("XBee: Analog  1: %.2fv\n",(3.3/1023)*p->IOanalog[1]);
        if (p->IOmask & 0x0800) printf("XBee: Analog  2: %.2fv\n",(3.3/1023)*p->IOanalog[2]);
        if (p->IOmask & 0x1000) printf("XBee: Analog  3: %.2fv\n",(3.3/1023)*p->IOanalog[3]);
        if (p->IOmask & 0x2000) printf("XBee: Analog  4: %.2fv\n",(3.3/1023)*p->IOanalog[4]);
        if (p->IOmask & 0x4000) printf("XBee: Analog  5: %.2fv\n",(3.3/1023)*p->IOanalog[5]);
#endif
      }
#ifdef DEBUG
      printf("XBee: ----------------------------\n");
#endif
    } else if (t == 0x83) {
    /* ########################################## */
    /* 16bit I/O recieve */
#ifdef DEBUG
      printf("XBee: Packet type: 16-bit RX I/O Data (0x83)\n");
      printf("XBee: 64-bit Address: ");
      for (j=0;j<2;j++) {
        printf((j?":%02X":"%02X"),d[j]);
      }
      printf("\n");
      printf("XBee: RSSI: -%ddB\n",d[2]);
      if (d[3] & 0x02) printf("XBee: Options: Address Broadcast\n");
      if (d[3] & 0x02) printf("XBee: Options: PAN Broadcast\n");
      printf("XBee: Samples: %d\n",d[4]);
#endif

      i = 7;

      for (o=d[4];o>0;o--) {
#ifdef DEBUG
        printf("XBee: --- Sample %3d -------------\n",o-d[4]+1);
#endif
        if (o<d[4]) {
          q = Xcalloc(sizeof(xbee_pkt));
          p->next = q;
          p = q;
          p->datalen = l;
        }

        p->type = xbee_16bitIO;

        p->sAddr64 = FALSE;
        p->dataPkt = FALSE;
        p->txStatusPkt = FALSE;
        p->modemStatusPkt = FALSE;
        p->remoteATPkt = FALSE;
        p->IOPkt = TRUE;

        p->Addr16[0] = d[0];
        p->Addr16[1] = d[1];

        p->RSSI = d[2];

        p->status = d[3];

        p->IOmask = (((d[5]<<8) | d[6]) & 0x7FFF);
        p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);

        i += (((d[5]&0x01)||(d[6]))?2:0);

        if (d[5]&0x02) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
        if (d[5]&0x04) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
        if (d[5]&0x08) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
        if (d[5]&0x10) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
        if (d[5]&0x20) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
        if (d[5]&0x40) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
#ifdef DEBUG
        if (p->IOmask & 0x0001) printf("XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
        if (p->IOmask & 0x0002) printf("XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
        if (p->IOmask & 0x0004) printf("XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
        if (p->IOmask & 0x0008) printf("XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
        if (p->IOmask & 0x0010) printf("XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
        if (p->IOmask & 0x0020) printf("XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
        if (p->IOmask & 0x0040) printf("XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
        if (p->IOmask & 0x0080) printf("XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));
        if (p->IOmask & 0x0100) printf("XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));
        if (p->IOmask & 0x0200) printf("XBee: Analog  0: %.2fv\n",(3.3/1023)*p->IOanalog[0]);
        if (p->IOmask & 0x0400) printf("XBee: Analog  1: %.2fv\n",(3.3/1023)*p->IOanalog[1]);
        if (p->IOmask & 0x0800) printf("XBee: Analog  2: %.2fv\n",(3.3/1023)*p->IOanalog[2]);
        if (p->IOmask & 0x1000) printf("XBee: Analog  3: %.2fv\n",(3.3/1023)*p->IOanalog[3]);
        if (p->IOmask & 0x2000) printf("XBee: Analog  4: %.2fv\n",(3.3/1023)*p->IOanalog[4]);
        if (p->IOmask & 0x4000) printf("XBee: Analog  5: %.2fv\n",(3.3/1023)*p->IOanalog[5]);
#endif
      }
```

```c
#ifdef DEBUG
      printf("XBee: ----------------------------\n");
#endif
    /* ########################################## */
    /* Unknown */
    } else {
#ifdef DEBUG
      printf("XBee: Packet type: Unknown (0x%02X)\n",t);
#endif
      p->type = xbee_unknown;
    }
    p->next = NULL;

    pthread_mutex_lock(&xbee.pktmutex);
    i = 1;
    if (!xbee.pktlist) {
      xbee.pktlist = po;
    } else {
      q = xbee.pktlist;
      while (q->next) {
        q = q->next;
        i++;
      }
      q->next = po;
    }

#ifdef DEBUG
    while (q && q->next) {
      q = q->next;
      i++;
    }
    printf("XBee: --========================--\n");
    printf("XBee: Packets: %d\n",i);
#endif

    po = p = q = NULL;
    pthread_mutex_unlock(&xbee.pktmutex);
  }
}

/* #################################################################
   xbee_getByte - INTERNAL
   waits for an escaped byte of data */
unsigned char xbee_getByte(void) {
  unsigned char c;

  ISREADY;

  c = xbee_getRawByte();
  if (c == 0x7D) c = xbee_getRawByte() ^ 0x20;

  return (c & 0xFF);
}

/* #################################################################
   xbee_getRawByte - INTERNAL
   waits for a raw byte of data */
unsigned char xbee_getRawByte(void) {
  unsigned char c;
  fd_set fds;

  ISREADY;

  FD_ZERO(&fds);
  FD_SET(xbee.ttyfd,&fds);

  if (select(xbee.ttyfd+1,&fds,NULL,NULL,NULL) == -1) {
    perror("xbee:xbee_listen():xbee_getByte()");
    exit(1);
  }

  do {
    if (read(xbee.ttyfd,&c,1) == 0) {
      usleep(10);
      continue;
    }
  } while (0);

  return (c & 0xFF);
}

/* #################################################################
   xbee_send_pkt - INTERNAL
   sends a complete packet of data */
void xbee_send_pkt(t_data *pkt) {
  ISREADY;

  /* write and flush the data */
  fwrite(pkt->data,pkt->length,1,xbee.tty);
  fflush(xbee.tty);

#ifdef DEBUG
  {
    int i;
    /* prints packet in hex byte-by-byte */
    printf("XBee: TX Packet - ");
    for (i=0;i<pkt->length;i++) {
      printf("0x%02X ",pkt->data[i]);
    }
    printf("\n");
  }
#endif

  xbee_destroy_pkt(pkt);
}

/* #################################################################
   xbee_make_pkt - INTERNAL
   adds delimiter field
   calculates length and checksum
   escapes bytes */
t_data *xbee_make_pkt(unsigned char *data, int length) {
  t_data *pkt;
  unsigned int l, i, o, t, x, m;
  char d = 0;

  ISREADY;

  /* check the data given isnt too long */
  if (length > 0xFFFF) return NULL;

  /* calculate the length of the whole packet */
  l = 3 + length + 1;

  /* prepare memory */
  pkt = Xmalloc(sizeof(t_data));

  /* put start byte on */
  pkt->data[0] = 0x7E;

  /* copy data into packet */
  for (t=0,i=0,o=1,m=1;i<=length;o++,m++) {
    if (i == length) {
      d = M8((0xFF - M8(t)));
    }
    else if (m == 1) d = M8(length >> 8);
    else if (m == 2) d = M8(length);
    else if (m > 2) d = data[i];
    x = 0;
    /* check for any escapes needed */
    if ((d == 0x11) || /* XON */
        (d == 0x13) || /* XOFF */
        (d == 0x7D) || /* Escape */
        (d == 0x7E)) { /* Frame Delimiter */
      l++;
      pkt->data[o++] = 0x7D;
      x = 1;
    }

    /* move data in */
    pkt->data[o] = (!x)?(d):(d^0x20);
    if (m > 2) {
      i++;
      t += d;
    }
  }

  /* remember the length */
  pkt->length = l;

  return pkt;
}

/* #################################################################
   xbee_destroy_pkt - INTERNAL
   free’s the packet memory */
void xbee_destroy_pkt(t_data *pkt) {

  ISREADY;

  /* free the stuff! */
  Xfree(pkt);
}
```