```
/* title:   comedi_server.c

** author:  jamie mazer
** created: Wed Jan  8 17:21:15 2003 mazer
** info:    shm interface to COMEDI devices
** history:
**
** Wed Jan  8 17:20:18 2003 mazer
**   - based on das16_server.c -- this is driver for the COMEDI
**     data acq. library/device-driver kit.  It’s GENERIC, designed
**     to work with the ISA & PCI versions of the DAS-1602 card.
**
** Sun Mar  9 13:34:54 2003 mazer
**   added support for din_changes[] to dig_in()
**
** Wed Nov  3 15:02:41 2004 mazer
**   added support for the DAS08 board (no 8255!!)
**
** Tue Apr  3 08:37:59 2007 mazer
**   cleaned up error messages for comedit to make it easier to track
**   down problems with non-DAS/ComputerBoards cards (like NI-6025E).
**
** Tue May  5 15:58:44 2009 mazer
**   joystick junk moved into separate JS device in das_common.c
**
** Thu Jul 22 12:06:45 2010 mazer
**   - looks like only das08 now is really pci-das08.. so pci-das08->das08
**   - moreover, looks like driver name changed, so for the best..
**   - also fixed some "signed" to "unsigned" that were probably always wrong..
**
******************************************************************************
**
**  Wed Sep 22 11:59:30 2010 mazer  -- from das_common.c
**
** title:   das_common.c
** author:  jamie mazer
** created: Mon Mar  4 16:41:26 2002 mazer
** info:    dasXX_server.c common functions
** history:
**
** Thu Apr  4 14:06:25 2002 mazer
**   - changed calls to setpriority to also bump scheduler priority up to
**      realtime (SCHED_RR)
**
** Fri Aug 23 16:53:54 2002 mazer
**   - Modified timestamp() to use the RDTSC for speed.  At
**     1 gHz, the 8byte (64bit) counter would overflow in:
**       (2^64) / (1e9) secs = 1.8e10s, or more than 500 years..
**     so, I’m assuming overflow is NOT a problem right now..
**
** Thu Dec 19 14:03:32 2002 mazer
**   added EYELINK_TEST mode
**
** Wed Apr 16 10:41:16 2003 mazer
**   added parsing of $XXEYELINK_OPTS to allow setting of eyelink
**   parameters in the pyperc config file...
**
** Sun Nov  6 10:06:36 2005 mazer
**   added $EYELINK_FILE to save native EDF file during run.
**
** Tue Jan 17 11:37:56 2006 mazer
**   - added $(CWD)/eyelink.ini file --> supplemental commands for the
**     eyelink
**   - made sure stderr messages all contain progname..
**
** Mon Jan 23 10:01:22 2006 mazer
**   Added handling of FIXWIN.vbias for vertical elongation of the
**   fixation window.
**
** Fri Mar 10 10:08:25 2006 mazer
**   Added stub support of a usb joystick or keypad. Right now the
**   device is detected and initialized, but nothing’s done yet
**   with the signals.
**
** Thu Apr 13 09:38:38 2006 mazer
**   merged stand-alone iscan_server code into the main event
**   loop for das_common, so all XXX_server’s will be able to
**   talk to the iscan without competition from a separate
**   process.
**
** Thu May 25 11:40:58 2006 mazer
**   changed z from int to float in mainloop() to avoid overflow
**   errors on (x*x)+(y*y) with ISCAN...
**
** Tue Nov 28 16:58:07 2006 mazer
**   added support for a ms-resolution alarm that sends interupts
**   the client/parent process
**
** Tue Apr  3 10:39:56 2007 mazer
**   added support for "-notracker" mode (for acutes)
**
** Fri Jun 15 15:09:05 2007 mazer
**   added arange (analog input range) for comedi drivers
**
** Thu Dec 18 11:39:36 2008 mazer
**   - moved eyelink and iscan specific code into separate files
**     that get included here:
**       - iscan.c
**       - eyelink.c
**   - reorganized the mainloop to sample each channel only once
**     and then usleep for a bit to reduce CPU load. original
**     behavior can be restored by #defining SPIN_SAMPLE (which
**     averages over the 1ms interval in a tight loop).
**
** Tue May  5 14:40:33 2009 mazer
**   - removed EYELINK_TEST mode completely..
**   - changed private XXxxx env vars to XX_xxxx
**
**************************************************************************
**
** Wed Sep 22 12:12:25 2010 mazer
**   - merged das_common.[ch], iscan.c & eyelink.c directly into this
**     file
**
*/

//#define SPIN_SAMPLE 1

#include <sys/types.h>
#include <sys/time.h>
#include <sys/errno.h>
#include <sys/resource.h>
```

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/mman.h>
#include <sys/io.h>
#include <signal.h>
#include <math.h>
#include <comedilib.h>

#include <sched.h>

#include <ezV24/ezV24.h>                /* for iscan serial I/O  */
#include <eyelink.h>            /* eyelink API  */
#include <eyetypes.h>           /* more eyelink API stuff */
#include <my_core_expt.h>       /* my_... works with both 32bit & 64bit libs */

#include "dacqinfo.h"
#include "sigs.h"
#include "psems.h"
#include "usbjs.h"
#include "debug.h"

static char *progname = NULL;
static DACQINFO *dacq_data = NULL;
static int mem_locked = 0;
static int dummymode = 0;

/* from das_common.h */
static unsigned long timestamp(int init);
static void perror2(char *s, char *file, int line);
static void mainloop(void);
static void iscan_halt(void);
static int semid;
static double arange;
static int usbjs_dev;
/* end das_common.h */

static int das08 = 0;           /* board is das08? (was pci-das08)*/

static char *comedi_devname = "/dev/comedi0";
static comedi_t *comedi_dev;    /* main handle to comedi lib */
static int analog_in = -1;      /* subdevice for analog input */

static int use8255;             /* 0 for ISA, 1 for PCI */
static int dig_io = -1;         /* combined digital I/O subdevice */
static int dig_i = -1;          /* digital IN only subdevice (ISA) */
static int dig_o = -1;          /* digital OUT only subdevice (ISA)*/
static int analog_range;

static char *_tmodes[] = {
  "ANALOG", "ISCAN", "EYELINK", "EYEJOY", "NONE"
};
#define ANALOG          0
#define ISCAN           1
#define EYELINK         2
```

```
#define EYEJOY          3
#define NONE            4

#define INSIDE          1
#define OUTSIDE         0

static int tracker_mode = ANALOG;
static int semid = -1;
static unsigned long long ticks_per_ms = 0;
static int swap_xy = 0;
static int usbjs_dev = -1;
static int iscan_x, iscan_y, iscan_p;
static double arange = 10.0;
static int eyelink_camera = -1;         /* eyelink handle */
static v24_port_t *iscan_port = NULL;   /* iscan handle */


// for cards with 8255 digital i/o (autodetectged), we have banks
// A,B and C and we want bank A (0-7) for input and B (8-15) for output.
#define BANK_A          0
#define BANK_B          8
#define PCI_NOWRITEMASK 0
#define PCI_READMASK    (1+2+4+8+16+32+64+128)
#define PCI_WRITEMASK   (1+2+4+8+16+32+64+128)<<BANK_B

// for the ISA cards, we have 4 bits of digital input and 4 of output
#define ISA_NOWRITEMASK 0
#define ISA_WRITEMASK   (1+2+4+8)

void iscan_halt()
{
  if (iscan_port) {
    v24ClosePort(iscan_port);
    fprintf(stderr, "%s: closed iscan_port\n", progname);
  }
}

void iscan_read()
{
  static unsigned char buf[25];
  static int bp = -1;
  static short *ibuf;
  static int lastc = -1;
  int c;

  // 2 bytes/param; 2 params/packet ==> 4 bytes/packet (XP, YP, XCR, YCR)
  int packet_length = 8;

  /* initialize the read buffer */
  if (bp < 0) {
    for (bp = 0; bp < sizeof(buf); bp++) {
      buf[bp] = 0;
    }
    bp = -1;
    ibuf = (short *)buf;
    iscan_x = 99999;
    iscan_y = 99999;
    iscan_p = 0;
  }

  if ((c = v24Getc(iscan_port)) < 0) {
    return;
  }
  if (c == 'D') {
    if (lastc == 'D') {
      lastc = -1;
      bp = 0;
    } else {
      lastc = c;
    }
    return;
  }
  if (bp >= 0) {
    buf[bp] = 0x00ff & c;
    if (bp == (packet_length - 1)) {
      if (ibuf[0] || ibuf[1] || ibuf[2] || ibuf[3]) {
        // currently packets should be:
        ///   <PUP_H1 PUP_V1 CR_H1 CR_V1>
        iscan_x = (ibuf[0] - ibuf[2] + 4096);
        iscan_y = (ibuf[1] - ibuf[3] + 4096);
        iscan_p = 1000;
        //fprintf(stderr, "  x=%d y=%d\n", iscan_x, iscan_y); fflush(stderr);
        return;
      } else {
        // out of range or no pupil lock
        iscan_x = 99999;
        iscan_y = 99999;
        iscan_p = 0;
        //fprintf(stderr, "* x=%d y=%d\n", iscan_x, iscan_y); fflush(stderr);
        return;
      }
    } else {
      if (++bp > (packet_length - 1)) {
        fprintf(stderr, "something bad happened.\n");
      }
    }
  }
}

void eyelink_init(char *ip_address)
{
  char *p, *q, *opts, buf[100];
  extern char *__progname;
  char *saved;
  FILE *fp;


  fprintf(stderr, "%s/eyelink_init: trying %s\n", progname, ip_address);

  saved = malloc(strlen(__progname) + 1);
  strcpy(saved, __progname);

  //begin_realtime_mode();
  set_eyelink_address(ip_address);

  if (open_eyelink_connection(0)) {
    fprintf(stderr, "\n%s/eyelink_init: can't open connection to tracker\n",
            progname);
    return;
  }
  set_offline_mode();
```

```c
  /* 16-apr-2003: step through the XX_EYELINK_OPTS env var (commands
   * separated by :'s) and set each command to the eyelink, while
   * echoing to the console..  This variable is setup by pype before
   * dacq_start() gets called..
   */
  opts = getenv("XX_EYELINK_OPTS");
  for (q = p = opts; *p; p++) {
    if (*p == ':') {
      *p = 0;
      eyecmd_printf(q);
      fprintf(stderr, "%s: eyelink_opt=<%s>\n", progname, q);
      *p = ':';
      q = p + 1;
    }
  }

  /* this should be "0" or "1", default to 1 */
  p = getenv("XX_EYELINK_CAMERA");
  if (p == NULL || sscanf(p, "%d", &eyelink_camera) != 1) {
    eyelink_camera = 1;
  }
  sprintf(buf, "eyelink_camera = %d", eyelink_camera);
  fprintf(stderr, "%s: %s\n", progname, buf);
  eyecmd_printf(buf);

  /* Tue Jan 17 11:37:48 2006 mazer
   * if file "eyelink.ini" exists in the current directory, send
   * it as a series of commands to the eyelink over the network.
   */

  if ((fp = fopen("eyelink.ini", "r")) != NULL) {
    while (fgets(buf, sizeof(buf), fp) != NULL) {
      if ((p = index(buf, '\n')) != NULL) {
        *p = 0;
      }
      fprintf(stderr, "%s: %s\n", progname, buf);
      eyecmd_printf(buf);
    }
  }

  // start recording & tell EL to send samples but
  // not events through link
  p = getenv("EYELINK_FILE");
  if (p != NULL) {
    open_data_file(p);
    if (start_recording(1,1,1,0)) {
      fprintf(stderr, "%s/eyelink_init: can't start recording\n", progname);
      return;
    }
    fprintf(stderr, "%s/eyelink_init: saving data to '%s'\n", progname, p);
  } else {
    if (start_recording(0,0,1,0)) {
      fprintf(stderr, "%s/eyelink_init: can't start recording\n", progname);
      return;
    }
  }

  if (eyelink_wait_for_block_start(10,1,0)==0) {
    fprintf(stderr, "%s/eyelink_init: can't get block start\n", progname);
    return;
  }
  fprintf(stderr, "%s/eyelink_init: connected ok\n", progname);
  tracker_mode = EYELINK;
}

void eyelink_halt()
{
  char *p;

  if (tracker_mode == EYELINK) {
    stop_recording();
    set_offline_mode();
    tracker_mode = ANALOG;

    p = getenv("EYELINK_FILE");
    if (p != NULL) {
      pump_delay(500);
      eyecmd_printf("close_data_file");
      fprintf(stderr, "%s/eyelink_halt: requesting ’%s’\n", progname, p);
      if (receive_data_file(p, p, 0) > 1) {
        fprintf(stderr, "%s/eyelink_halt: received.\n", progname);
      } else {
        fprintf(stderr, "%s/eyelink_halt: error receiving.\n", progname);
      }
    }
  }
}

int eyelink_read(float *x, float *y,  float *p,
                        unsigned int *t, int *new)
{
  static FSAMPLE sbuf;
  int e;

  if ((e = eyelink_newest_float_sample(&sbuf)) < 0) {
    return(0);
  } else {
    /* there are new data about eye positions */
    *t = (unsigned int) sbuf.time;
    *x = sbuf.px[eyelink_camera];               /* xpos, RIGHT/LEFT */
    *y = sbuf.py[eyelink_camera];               /* ypos, RIGHT/LEFT */
    *p = sbuf.pa[eyelink_camera];               /* pupil area, RIGHT/LEFT */
    *new = (e == 1);
    return(1);
  }
}

int comedi_init()
{
  const char *devname;
  comedi_range *r;
  int n;

  if (!(comedi_dev = comedi_open(comedi_devname))) {
    fprintf(stderr, "%s: can’t find comedi board.\n", progname);
    return(0);
  }
  devname = comedi_get_driver_name(comedi_dev);

  fprintf(stderr, "%s: found DAQ device board=<%s> driver=<%s>\n",
          progname, comedi_get_board_name(comedi_dev), devname);
  if (strncmp(devname, "das16", 5) == 0) {
    fprintf(stderr, "%s: 8255 disabled.\n", progname);
    use8255 = 0;
  } else if (strncmp(devname, "das08", 5) == 0) {
    use8255 = 0;
    das08 = 1;
    fprintf(stderr, "%s: 8255 disabled.\n", progname);
    fprintf(stderr, "%s: detected das08 -- will used delayed input\n", progname);
  } else {
    fprintf(stderr, "%s: 8255 enabled.\n", progname);
    use8255 = 1;
  }

  // find which comedi subdevices correspond the the facilities we need
  analog_in  = comedi_find_subdevice_by_type(comedi_dev,COMEDI_SUBD_AI,0);
  if (analog_in == -1) {
    comedi_perror("analog_in");
  } else {
    fprintf(stderr, "%s: analog input OK\n", progname);
  }

  n = comedi_get_n_channels(comedi_dev, analog_in);
  fprintf(stderr, "%s: %d analog inputs available.\n", progname, n);
  n = comedi_get_n_ranges(comedi_dev, analog_in, 0);
  fprintf(stderr, "%s: %d analog ranges available.\n", progname, n);
  if (n > 1) {
    // try to find the +/- 10V range.  the 4th parm means 'volts'.
    // BW: I THINK THIS ASSUMES ALL CHANNELS ARE THE SAME
    //analog_range = comedi_find_range(comedi_dev,analog_in,0,0,-10,10);
    analog_range = comedi_find_range(comedi_dev,analog_in,0,0,-arange,arange);
    if (analog_range == -1) {
      comedi_perror("analog_range");
    }
  } else {
    // DAS08 doesn't have programmable ranges -- use 0
    analog_range = 0;
  }
  r = comedi_get_range(comedi_dev, analog_in, 0, analog_range);
  fprintf(stderr, "%s: analog range (%.1f%s)-(%.1f%s)\n", progname,
          r->min, (r->unit==UNIT_volt) ? "V" : "??",
          r->max, (r->unit==UNIT_volt) ? "V" : "??");

  if (use8255) {
    dig_io = comedi_find_subdevice_by_type(comedi_dev,COMEDI_SUBD_DIO,0);
    if (dig_io == -1) {
      comedi_perror("dig_io");
    } else {
      fprintf(stderr, "%s: digitial IO OK\n", progname);
      dig_i = -1;
      dig_o = -1;
    }
  } else {
    dig_i  = comedi_find_subdevice_by_type(comedi_dev,COMEDI_SUBD_DI,0);
    if (dig_i == -1) {
      comedi_perror("dig_i");
    } else {
      fprintf(stderr, "%s: digitial input OK\n", progname);
    }
    dig_o = comedi_find_subdevice_by_type(comedi_dev,COMEDI_SUBD_DO,0);
    if (dig_o == -1) {
      comedi_perror("dig_o");
    } else {
      fprintf(stderr, "%s: digitial output OK\n", progname);
    }
  }

  if (use8255) {
    // configure digital I/O bank A as input, and bank B as output
    if (comedi_dio_config(comedi_dev,dig_io,BANK_A,COMEDI_INPUT) &&
        comedi_dio_config(comedi_dev,dig_io,BANK_B,COMEDI_OUTPUT)) {
      return(1);
    } else {
      return(0);
    }
  }
  return(1);
}


int ad_in(int chan)
{
  lsampl_t sample;
  int success;

  if (dummymode) {
    return(0);
  } else {
    // need to set aref correctly: either AREF_GROUND or AREF_COMMON
    if (das08) {
      // das08 is screwy -- needs time for multiplexer to settle:
      success = comedi_data_read_delayed(comedi_dev,analog_in,
                                         chan,analog_range,AREF_GROUND,
                                         &sample, 0);
      if (success < 0) {
        comedi_perror("comedi_data_read_delayed");
      }
    } else {
      success = comedi_data_read(comedi_dev,analog_in,
                                 chan,analog_range,AREF_GROUND,
                                 &sample);
      if (success < 0) {
        comedi_perror("comedi_data_read");
      }
    }
    // note: lsampl is an unsigned int; we are casting to int. it won't
    // matter for 12 bit cards
    return((int)sample);
  }
}

void dig_in()
{
  int i, success, last;
  unsigned int bits;

  if (dummymode) {
    // just lock these down -- polarities are
    // from the old taks -- hardcoded to work in NAF...
    LOCK(semid);
    dacq_data->din[0] = 0;      /* monkey bar NOT down */
    dacq_data->din[2] = 1;      /* user button 2 NOT down */
```

```c
    dacq_data->din[3] = 1;      /* user button 1 NOT down */
    UNLOCK(semid);
  } else {
    if (use8255) {
      success = comedi_dio_bitfield(comedi_dev,dig_io,PCI_NOWRITEMASK,&bits);
      bits = bits & PCI_READMASK;
    } else {
      success = comedi_dio_bitfield(comedi_dev,dig_i,ISA_NOWRITEMASK,&bits);
    }
    /* unpack inp word into the first 8 slots of the dacq struct’s din array */
    for (i = 0; i < 4; i++) {
      LOCK(semid);
      last = dacq_data->din[i];
      dacq_data->din[i] = ((bits & 1<<i) != 0);
      if (dacq_data->din[i] != last) {
        dacq_data->din_changes[i] += 1;
        if (dacq_data->din_intmask[i]) {
          dacq_data->int_class = INT_DIN;
          dacq_data->int_arg = i;
          kill(getppid(), SIGUSR1);
        }
      }
      UNLOCK(semid);
    }
  }
}

void dig_out()
{
  unsigned int bits = 0;
  int i, success;

  if (dummymode) {
    return;
  } else {
    for (i = 0; i < 8 && i < NDIGOUT; i++) {
      LOCK(semid);
      bits = bits | (dacq_data->dout[i] << i);
      UNLOCK(semid);
    }
    if (use8255) {
      bits = bits<<BANK_B;
      success = comedi_dio_bitfield(comedi_dev,dig_io,PCI_WRITEMASK,&bits);
    } else {
      success = comedi_dio_bitfield(comedi_dev,dig_o,ISA_WRITEMASK,&bits);
    }
  }
}

void mainloop_halt(void)
{
  fprintf(stderr, "%s: mainloop_halt()\n", progname);

  comedi_close(comedi_dev);

  if (dacq_data != NULL) {
    shmdt(dacq_data);
  }
  if (mem_locked) {
    if (munlockall() != 0) {
      perror2("munlockall", __FILE__, __LINE__);
    } else {
      mem_locked = 0;
    }
  }
}

int mainloop_init()
{
  int shmid;

  if (comedi_init()) {
    dummymode = 0;
  } else {
    fprintf(stderr, "%s: falling back to dummymode\n", progname);
    dummymode = 1;
  }
  fprintf(stderr, "%s: comedi initialized.\n", progname);
  if (dig_io >= 0) {
    fprintf(stderr, "%s: dig_io=subdev #%d\n", progname, dig_io);
  }
  if (dig_i >= 0) {
    fprintf(stderr, "%s: dig_i=subdev #%d\n", progname, dig_i);
  }
  if (dig_o >= 0) {
    fprintf(stderr, "%s: dig_o=subdev #%d\n", progname, dig_o);
  }
  if (analog_in >= 0) {
    fprintf(stderr, "%s: analog_in=subdev #%d\n", progname, analog_in);
  }

  if ((shmid = shmget((key_t)SHMKEY,
                      sizeof(DACQINFO), 0666 | IPC_CREAT)) < 0) {
    perror2("shmget", __FILE__, __LINE__);
    fprintf(stderr, "%s:init -- kernel compiled with SHM/IPC?\n", progname);
    exit(1);
  }

  if ((dacq_data = shmat(shmid, NULL, 0)) == NULL) {
    perror2("shmat", __FILE__, __LINE__);
    fprintf(stderr, "%s:init -- kernel compiled with SHM/IPC?\n", progname);
    exit(1);
  }

  if (mlockall(MCL_CURRENT) == 0) {
    mem_locked = 1;
  } else {
    perror2("mlockall", __FILE__, __LINE__);
    fprintf(stderr, "%s:init -- failed to lock memory\n", progname);
  }
  LOCK(semid);
  if (dacq_data->dacq_pri != 0) {
    if (nice(dacq_data->dacq_pri) == 0) {
      fprintf(stderr, "%s:init -- bumped priority %d\n",
              progname, dacq_data->dacq_pri);
    } else {
      perror2("nice", __FILE__, __LINE__);
      fprintf(stderr, "%s:init -- failed to change priority\n", progname);
    }
  }
  UNLOCK(semid);
  atexit(mainloop_halt);
  catch_signals(progname);

  /* ignore iscan exiting: */
  //signal(SIGCHLD, SIG_IGN);

  return(1);
}

/* from das_common.c */

void iscan_init(char *dev)
{
  if ((iscan_port = v24OpenPort(dev, V24_NO_DELAY | V24_NON_BLOCK)) == NULL) {
    fprintf(stderr, "%s: iscan_init can't open \"%s\"\n.", progname, dev);
    exit(1);
  }
  v24SetParameters(iscan_port, V24_B115200, V24_8BIT, V24_NONE);

  tracker_mode = ISCAN;
  fprintf(stderr, "%s: opened iscan_port (%s)\n", progname, dev);
}

double find_clockfreq() /* get clock frequency in Hz */
{
  FILE *fp;
  char buf[100];
  double mhz;

  if ((fp = fopen("/proc/cpuinfo", "r")) == NULL) {
    fprintf(stderr, "%s: can't open /proc/cpuinfo\n", progname);
    exit(1);
  }
  mhz = -1.0;
  while (fgets(buf, sizeof(buf), fp) != NULL) {
    if (sscanf(buf, "cpu MHz          : %lf", &mhz) == 1) {
      break;
    }
  }
  return(mhz * 1.0e6);
}

#if defined(__i386__)

// this macro doesn't quite work (under 64bit??)

#define RDTSC(x) __asm__ __volatile__ ( ".byte 0x0f,0x31"               \
                                        :"=a" (((unsigned long*)&x)[0]), \
                                         "=d" (((unsigned long*)&x)[1]))

unsigned long timestamp(int init)
{
  static unsigned long long timezero;
  unsigned long long now;

  RDTSC(now);                   /* get cycle counter from hardware TSC */
  if (init) {
    timezero = now;
    return(0);
  } else {
    /* use precalibrated ticks_per_ms to convert to real time.. */
    return((unsigned long)((now - timezero) / ticks_per_ms));
  }
}

#elif defined(__x86_64__)

/* need to use different method to access real time clock
** under 64bit kernel!
*/

unsigned long timestamp(int init)
{
  static unsigned long long timezero;
  unsigned long long now;
  unsigned a, d;

  asm("cpuid");
  asm volatile("rdtsc" : "=a" (a), "=d" (d));
  now = ((unsigned long long)a) | (((unsigned long long)d) << 32);

  if (init) {
    timezero = now;
    return(0);
  } else {
    /* use precalibrated ticks_per_ms to convert to real time.. */
    return((unsigned long)((now - timezero) / ticks_per_ms));
  }
}
#else
# error "real time clock not defined this arch"
#endif

void perror2(char *s, char *file, int line)
{
  char *p = (char *)malloc(strlen(progname)+strlen(s)+25);

  sprintf(p, "%s (file=%s, line=%d):%s", progname, file, line, s);
  perror(p);
  free(p);
}

void resched(int rt)
{
#ifdef ALLOW_RESCHED
  struct sched_param p;

  /* change scheduler priority from OTHER to RealTime/RR or vice versa */

  if (sched_getparam(0, &p) >= 0) {
    if (rt) {
      p.sched_priority = SCHED_RR;
      sched_setscheduler(0, SCHED_RR, &p);
    } else {
      p.sched_priority = SCHED_OTHER;
      sched_setscheduler(0, SCHED_OTHER, &p);
    }
  }
#endif
}

void mainloop(void)
```

```
{
  register int i, ii, lastpri, setpri;
  register float x, y, z, pa, tmp, calx, caly;
  float tx, ty, tp;
  unsigned long last_ts = 0, ts;
  unsigned int eyelink_t;
  int eyelink_new;
  int k;
  int jsbut, jsnum, jsval;
  unsigned long jstime;

  register float sx=0, sy=0;
  int si, sn, last;
  float sbx[MAXSMOOTH], sby[MAXSMOOTH];

  /*
   * calx/caly are the gain+offset adjusted eye position values
   * x/y are the raw values
   */
  calx = caly = x = y = pa = -1;
  y = x = 0.0;
  for (si = 0; si < MAXSMOOTH; si++) {
    sbx[si] = sby[si] = 0.0;
  }
  si = 0;

  errno = 0;
  LOCK(semid);
  k = dacq_data->dacq_pri;
  UNLOCK(semid);

  if (setpriority(PRIO_PROCESS, 0, k) == 0 && errno == 0) {
    fprintf(stderr, "%s: bumped priority %d\n", progname, k);
    lastpri = k;
    if (lastpri < 0) {
      resched(1);
    }
    setpri = 1;
  } else {
    fprintf(stderr, "%s: failed to change priority\n", progname);
    setpri = 0;
    lastpri = 0;
  }

  timestamp(1);                 /* initialize the timestamp to 0 */

  fprintf(stderr, "%s: tracker_mode=%s (%d)\n", progname,
          _tmodes[tracker_mode], tracker_mode);

  /* signal client we’re ready */
  LOCK(semid);
  dacq_data->das_ready = 1;
  fprintf(stderr, "%s: ready\n", progname);
  UNLOCK(semid);

  do {
    /* sample converters as fast as possible and accumulate
     * into temp buffer for averaging at the end of the sample
     * period (1ms).  This replaces spin-locking code.
     */
#ifdef SPIN_SAMPLE
```

```
    {
      int naccum = 0;
      long accum[NADC];

      do {
        /* sample the converters & acculumate values */
        for (i = 0; i < NADC; i++) {
          if (naccum == 0) {
            accum[i] = ad_in(i);
          } else {
            accum[i] += ad_in(i);
          }
        }
        naccum += 1;
        if (tracker_mode == ISCAN) {
          iscan_read();
        }
      } while (((ts = timestamp(0)) - last_ts) < 1);
      last_ts = ts;

      /* adjust for # of acculumated values */
      for (i = 0; i < NADC; i++) {
        LOCK(semid);
        dacq_data->adc[i] = (int)(accum[i] / naccum);
        UNLOCK(semid);
      }
    }
#else
    // usleep(500); --> looks like usleep actually sleeps for more
    // like 8ms minimum... so we’re back to spinning..

    /* and then wait for the 1ms interval to elapse */
    while ((timestamp(0) - last_ts) < 1) {
      ;
    }
    /* now quickly sample all the converters just once */
    for (i = 0; i < NADC; i++) {
      dacq_data->adc[i] = ad_in(i);
    }
    if (tracker_mode == ISCAN) {
      iscan_read();
    }
    ts = last_ts = timestamp(0);
#endif

    if (usbjs_dev > 0) {
      if (usbjs_query(usbjs_dev, &jsbut, &jsnum, &jsval, &jstime)) {
        if (jsbut) {
          /* button press: jsnum is button number, jsval is up/down */
          if (jsnum < NJOYBUT) {
            LOCK(semid);
            dacq_data->js[jsnum] = jsval;
            UNLOCK(semid);
          }
        } else if (jsbut == 0 && jsnum == 0) {
          /* x-axis motion, jsval indicates the current value */
          dacq_data->js_x = jsval;
        } else if (jsbut == 0 && jsnum == 1) {
          /* y-axis motion, jsval indicates the current value */
          dacq_data->js_y = jsval;
        }
      }
    }

    switch (tracker_mode)
      {
      case NONE:
        x = y = pa = 0;
        break;
      case ISCAN:
        x = iscan_x;
        y = iscan_y;
        pa = iscan_p;
        break;
      case EYELINK:
        if (eyelink_read(&tx, &ty, &tp, &eyelink_t, &eyelink_new) != 0) {
          x = tx;
          y = ty;
          pa = tp;
        }
        break;
      case EYEJOY:
        LOCK(semid);
        x = x + (dacq_data->js_x > 0 ? 1 : dacq_data->js_x < 0 ? -1 : 0)/100.0;
        y = y - (dacq_data->js_y > 0 ? 1 : dacq_data->js_y < 0 ? -1 : 0)/100.0;
        dacq_data->adc[0] = x;
        dacq_data->adc[1] = y;
        UNLOCK(semid);
        pa = -1;
        break;
      default:
        x = dacq_data->adc[0];
        y = dacq_data->adc[1];
        pa = -1;
        break;
      }

    if (swap_xy) {
      tmp = x; x = y; y = tmp;
    }

    /* smooth (if necessary) raw eye position trace */
    LOCK(semid);
    sn = dacq_data->eye_smooth;
    if (sn > MAXSMOOTH) {
      sn = MAXSMOOTH;
    }
    UNLOCK(semid);

    if (sn > 1) {
      /* remove old point, add new point to smoothing sum */
      sx = sx - sbx[si] + x;
      sy = sy - sby[si] + y;

      /* add new (unsmoothed data points) to smoothing buffer */
      sbx[si] = x;
      sby[si] = y;
      si = (si + 1) % sn;

      /* calc smoothed point */
      x = sx / sn;
      y = sy / sn;
```

```
}

/* convert from raw to pixel domain and save in eye_x/eye_y */
LOCK(semid);
calx = (dacq_data->eye_xgain * x) - dacq_data->eye_xoff;
caly = (dacq_data->eye_ygain * y) - dacq_data->eye_yoff;
dacq_data->eye_x = (int)((calx > 0) ? (calx+0.5) : (calx-0.5));
dacq_data->eye_y = (int)((caly > 0) ? (caly+0.5) : (caly-0.5));
dacq_data->eye_pa = pa;
UNLOCK(semid);

/* read digital input lines */
if (usbjs_dev >= 0) {
  /* if joystick device is available (even if it’s not
   * being used as an eye tracker, use buttons as digital inputs..
   */
  for (i = 0; i < 4; i++) {
    LOCK(semid);
    last = dacq_data->din[i];
    dacq_data->din[i] = dacq_data->js[i];
    if (dacq_data->din[i] != last) {
      dacq_data->din_changes[i] += 1;
      if (dacq_data->din_intmask[i]) {
        dacq_data->int_class = INT_DIN;
        dacq_data->int_arg = i;
        kill(getppid(), SIGUSR1);
      }
    }
    UNLOCK(semid);
  }
} else {
  /* otherwise, fall back to comedi DIO lines etc */
  dig_in();
}

/* set digital output lines, only if the strobe’s been set */
LOCK(semid);
k = dacq_data->dout_strobe;
UNLOCK(semid);
if (k) {
  dig_out();
  /* reset the strobe (as if it were a latch */
  LOCK(semid);
  dacq_data->dout_strobe = 0;
  UNLOCK(semid);
}

LOCK(semid);
dacq_data->timestamp = ts;
k = dacq_data->adbuf_on;

/* check alarm status */
if (dacq_data->alarm_time && ts < dacq_data->alarm_time) {
  /* alarm set and expired -- clean and send interupt to
   * client (ie, parent)
   */
  dacq_data->alarm_time = 0;
  dacq_data->int_class = INT_ALARM;
  dacq_data->int_arg = 0;
  kill(getppid(), SIGUSR1);
}
```

```
    UNLOCK(semid);


    /* Stash the data, if recording is on:
     *  adbuf_t,x,y <- calibrated eye signal
     *  adbuf_pa <- pupil area, if available (eyelink only)
     *  adbuf_c[01234] <- raw data streams; in eyelink test mode
     *    these are:
     *     c0 <- eyelink x
     *     c1 <- eyelink y
     *     c2 <- coil raw x
     *     c3 <- coil raw y
     *     c4 <- eyelink pupil area
     */
    if (k) {
      LOCK(semid);
      k = dacq_data->adbuf_ptr;
      dacq_data->adbuf_t[k] = ts;
      dacq_data->adbuf_x[k] = dacq_data->eye_x;
      dacq_data->adbuf_y[k] = dacq_data->eye_y;
      dacq_data->adbuf_pa[k] = dacq_data->eye_pa;

      /* the raw analog values are stuffed in, which
       * are usually raw x,y values off the coil, unless you’re
       * using them for something else (and have iscan/eyelink)
       */
      for (ii=0; ii < NADC; ii++) {
        dacq_data->adbufs[ii][k] = dacq_data->adc[ii];
      }
      /* Mon Jan 16 09:25:34 2006 mazer
       *  set up saving EDF-time to c4 channel for debugging
       dacq_data->adbuf_c4[k] = eyelink_t;
      */
      if (++dacq_data->adbuf_ptr > ADBUFLEN) {
        dacq_data->adbuf_overflow++;
        dacq_data->adbuf_ptr = 0;
      }
      UNLOCK(semid);
    }

    /* check fixwins for in/out events */
    for (i = 0; i < NFIXWIN; i++) {
      LOCK(semid);
      k = dacq_data->fixwin[i].active;
      UNLOCK(semid);
      if (k) {
        LOCK(semid);
        x = dacq_data->eye_x - dacq_data->fixwin[i].cx;
        y = (dacq_data->eye_y - dacq_data->fixwin[i].cy) /
          dacq_data->fixwin[i].vbias;
        UNLOCK(semid);

        z = (x * x) + (y * y);

        LOCK(semid);
        if (z < dacq_data->fixwin[i].rad2) {
          /*
           * eye is now INSIDE the fixation window -- stop counting
           * transient breaks
           */
          dacq_data->fixwin[i].state = INSIDE;
          dacq_data->fixwin[i].fcount = 0;
        } else {
          /*
           * eye is outside the fixation window, but could be shot noise..
           */
          if (dacq_data->fixwin[i].state == INSIDE) {
            /*
             * eye was inside last sample, so the break just happened
             * reset the break counter and start counting # samples
             * outside fixation window
             */
            dacq_data->fixwin[i].fcount = 1;
            dacq_data->fixwin[i].nout = 0;
          }
          dacq_data->fixwin[i].state = OUTSIDE;
          if (dacq_data->fixwin[i].fcount) {
            dacq_data->fixwin[i].nout += 1;
            if (dacq_data->fixwin[i].nout > dacq_data->fixbreak_tau) {
              /* number of samples the eye’s been out of the window
               * has exceeded the limit defined by fixbreak_tau, count
               * this as a real fixation break.
               */
              if (dacq_data->fixwin[i].broke == 0) {
                /* stash time if it’s the first break */
                dacq_data->fixwin[i].break_time =  dacq_data->timestamp;
              }
              dacq_data->fixwin[i].broke = 1;
              if (dacq_data->fixwin[i].genint) {
                /* send interupt to parent */
                dacq_data->int_class = INT_FIXWIN;
                dacq_data->int_arg = 0;
                dacq_data->fixwin[i].genint = 0;
                kill(getppid(), SIGUSR1);
                /* fprintf(stderr,"das: sent int, disabled\n"); */
              }
            }
          }
        }
        UNLOCK(semid);
      }
    }

    /* possibly bump up or down priority on the fly */
    LOCK(semid);
    k = dacq_data->dacq_pri;
    UNLOCK(semid);
    if (setpri && lastpri != k) {
      lastpri = k;
      errno = 0;
      if (setpriority(PRIO_PROCESS, 0, k) == -1 && errno) {
        /* disable future priority changes */
        setpri = 0;
      }
      if (lastpri < 0) {
        resched(1);
      }
    }
    LOCK(semid);
    k = dacq_data->terminate;
    UNLOCK(semid);
  } while (! k);
  fprintf(stderr, "%s: terminate signaled\n", progname);
  iscan_halt();
  eyelink_halt();
  if (usbjs_dev >= 0) {
    usbjs_close(usbjs_dev);
  }

  /* no longer ready */
  LOCK(semid);
  dacq_data->das_ready = 0;
  UNLOCK(semid);
}

int main(int ac, char **av)
{
  char *p;
  float mhz;

  p = rindex(av[0], '/');
  progname = p ? (p + 1) : av[0];

  ticks_per_ms = (unsigned long long)(0.5 +
                                      ((mhz = find_clockfreq()) / 1000.0));

  if ((semid = psem_init(SEMKEY)) < 0) {
    perror("psem_init");
    fprintf(stderr, "%s: can't init semaphore\n", progname);
    exit(1);
  }

  // get requested analog input range for comedi device (+- ARANGE volts)
  if ((p = getenv("XX_ARANGE")) != NULL) {
    double d;
    if (sscanf(p, "%lf", &d) == 1) {
      arange = d;
    }
  }

  mainloop_init();
  fprintf(stderr, "%s: initted\n", progname);

  if (av[1] && (strcmp(av[1], "-iscan") == 0)) {
    iscan_init(av[2]);
  } else if (av[1] && (strcmp(av[1], "-eyelink") == 0)) {
    eyelink_init(av[2]);
  } else if (av[1] && (strcmp(av[1], "-eyejoy") == 0)) {
    tracker_mode = EYEJOY;
  } else if (av[1] && (strcmp(av[1], "-notracker") == 0)) {
    tracker_mode = NONE;
    fprintf(stderr, "%s: no tracker mode\n", progname);
  }

  if (getenv("XX_SWAP_XY")) {
    /* this option is useful ONLY if the camera is rotated, like with
     * the original software release for the eyelink ELCL...
     */
    swap_xy = 1;
    fprintf(stderr, "%s: swapping X and Y\n", progname);
  }
  if ((p = getenv("XX_USBJS")) != NULL) {
    usbjs_dev = usbjs_init(p);
    if (usbjs_dev < 0) {
      fprintf(stderr, "%s: can’t open joystick %s\n", progname, p);
    } else {
      fprintf(stderr, "%s: joystick at %s configured\n", progname, p);
      LOCK(semid);
      dacq_data->js_enabled = 1;
      UNLOCK(semid);
    }
  }

  mainloop();
  fprintf(stderr, "%s: bye bye\n", progname);
  exit(0);
}

/* end das_common.c */
```