```
/*
 * camac.c
 *
 * CAMAC interface routines
 *
 * 19jul00 jms  short reads dont work in the sjy driver
 *              use csmad, and zero suppress in s/w
 * 28sep00 jms  use s/w pedestal subtraction as LRS 4300 cannot
 *              use pedestals > 255
 * 03Mar04 dm   Streamlining --  removed code dealing with ADC stuff.
 */

#include "rundaq.h"
/*#include <ieee_fun_types.h> */

/* Local Globals */


static int       event_num;              /* Event Number */
static int       spill_num;              /* Spill Number */
static int       cc73aext;               /* Posn of the 73A crate controller */
static int       lamext;                 /* Posn of the LAM LATCH module */
static int       tdcext;                 /* Posn of the first TDC */
static int       cntext;                 /* Posn of the FERA controller */
static int       adcext;                 /* Posn of the first ADC */
static int       scalerext;              /* Posn of the first scaler */
static int       adc2249ext;             /* Posn of 2249 ADC         */
static short int peds[NADC];             /* Pedestals from file */


#define ADCCSR1 0x2800  /* seq. read. zero supp */
#define ADCCSR2 0x3800  /* sequential readout, zero suppression, ped sub */

int init_camac()
{
        int          rc = 0;
        unsigned int trigger = 0;
        short int    data;
        int          qres=0;
        int          NADCSL1;
        int          i,j;
        int          tadcext;        /* used to load and test the pedestals */
        int          dum[4];         /* reading peds */
        float        fdum[2];        /* reading peds */
        char         filename[256];  /* for the peds filename */
        FILE         *fd;            /* for the peds filename */


        /* initialize globals */
        NADCSL1 = NADCSL - 1;
        rc = camac_globals();
        if (rc != 0)
        {
                printf("init_camac: error from camac_globals %d\n",rc);
                return(rc);
        }


        /* switch off triggers */
                set_trigger(0);

        /* now we need to inhibit ADC gates in order to check ADCs,
           therefore we will take the first trigger ...*/

        set_trigger(PHYS);


        rc = read_trigger(&trigger,&daq_control.stat_ptr->run_status);

        if (rc != 0)
        {
              printf("read_trigger failed rc=%d\n", rc);
```



```
                daq_control.stat_ptr->run_status = DAQ_ERROR;
                return(rc);
        }


            /* cdreg(&adc2249ext, BRANCH, CRATE, 1, 0);  12-channel ADC in the first
+ slot  <-- */
            /* clear ADC2249 no Q is expected */
            /* cssa(9, adc2249ext, &data, &qres); */

        return(0);


 }

 int camac_globals()
 {
        int     rc;

        /* Initialise static globals */

        /* initialize event and spill counter */
        event_num = 0;
        spill_num = 0;

        /* initialise CAMAC */
        rc = cdchn(BRANCH, 1, 0);
        if ((rc&1) != 1)
        {
                printf("camac_globals: error=%d from cdchn\n", rc);
                cmsg("initialise branch", rc, BRANCH);
                return(100+rc);
        }

        /* Set up 73A crate controller */
        cdreg(&cc73aext, BRANCH, CRATE, 24, 0);
        printf("controller: 0x%x\n",cc73aext);

        /* Set up LAM module */
        cdreg(&lamext, BRANCH, CRATE, 2, 0);
        printf("LAM: 0x%x\n",lamext);

        /* set up the scalers */
        cdreg(&scalerext, BRANCH, CRATE, NSLSCLR1, 0);
        printf("Escalers: 0x%x\n",scalerext);

        /* set up the TDC */
         cdreg(&tdcext, BRANCH, CRATE, NSLTDC1, 0);\
        printf("TDCs: 0x%x\n",scalerext);


        /* set up ADC2249 module */
        cdreg(&adc2249ext, BRANCH, CRATE, 1, 0);
        printf("ADC: 0x%x\n",adc2249ext);


        /* Clear all modules in CRATE and remove inhibits, LAMS, etc */
        ccctype(BRANCH,0,1);
        cccbyp(cc73aext,0);
        cccoff(cc73aext,0);
        ccci(cc73aext,0);
        cccc(cc73aext);
        cccz(cc73aext);
        ccci(cc73aext,0);

        return(0);

 }

 int read_trigger(
 unsigned int *trigger,
 unsigned int *run_status)
 {
```

```
        /*
         * Here we will poll for the next trigger
         * by reading the RFD02 latch with an F1.
         */

        int                poll_count;             /* how many polls before we got
+a trigger */
        int                qres;                   /* Q response */
        unsigned short int data;                /* from CAMAC */
        int                i;
        int                j =0;
        int                isleep[] = {25000000,15000000};
        struct timespec to_sleep;


        /* now we have two sleeps 25 and 15 ms */

        to_sleep.tv_sec = 0;

        data = 0;
        poll_count = 0;
        i = 0;

        clear_trigger();

        while((data&0xFF) == 0)

        {
             to_sleep.tv_nsec = isleep[i%2];

             i++;

             sleep_for_a_while(&to_sleep);
             cssa(0, cc73aext, &data, &qres);

             if(qres != 1)
             {
                 printf("No Q from RFD02 to get source of LAM\n");
             }

             if ((data&0xFF) != 0)
             {
                 /* fill trigger and return */
                 cssa(1, lamext, &data, &qres);

                 if(qres != 1)
                 {
                      printf("No Q from RFD02 to get LATCH register status\n");
                 }

                 if ( ((data&0xFF)) != 0 || ((data&0xFF) == 0))
                 {
                      *trigger = data & 0xFF;
                      break;
                 } else
                        {
                                /* clear LAM */

                                printf("Empty trigger latch after %d polls  LATCH D
+ATA 0x%x\n",poll_count,data);
                                fflush(stdout);
                                clear_trigger();
                        }
             }

             poll_count++;
             if (poll_count % 100 == 0)
             {

                   printf("No trigger seen after %d polls\n",poll_count);
                   printf("data_seen in loop: 0x%x\n", data);
                   fflush(stdout);
```



```
                   clear_trigger();
                   if (*run_status != DAQ_RUN)
                        {
                                *trigger = SPILL;
                                break;
                        }
             }
        }
        sleep_for_a_while(&to_sleep);
        return(0);
}


void clear_trigger(void)
{
  struct timespec to_sleep;
        /* Clears LAM and module */

        unsigned short  data=0;
        int             qres=0;


        /* Reset and Enable LAM */

        data = 0;
        cssa(28,lamext,&data,&qres);

        if (qres != 1)
        {
                printf("No Q from Enable LAM on RFD02\n");
                cmsg("Enable LAM",data, BRANCH);
        }
}


void set_trigger(
int     mask)                           /* trigger mask */
{
        /* Set trigger mask */
        unsigned int dummy_rc;
        unsigned short  data=0;
        int             qres=0;

        data = mask;
        dummy_rc = cssa(16,lamext,&data,&qres);

        if (qres != 1)
        {
                printf("No Q from RFD02 to set trigger mask=%d\n",mask);
                cmsg("trigger mask",data, BRANCH);
        }

        clear_trigger();
}


int read_event(
unsigned int    trigger,                        /* trigger type */
BUFF_DATA       *spill,                         /* buffer to put the data */
int             *scalers)                       /* place to copy the scalers */
{
        /*
         * reads camac, puts data into spill buffer
         *
         */
        int             i;                      /* loop counter */
        int             scaldata[NSCALER];      /* SCALER buffer    */
        unsigned short  adcdata[NADC+NADCSL];   /* ADC buffer */
        short           tadcdata[NADC+NADCSL];  /* ADC buffer */
        int             slots[NADCSL] = {10,11,12,13,14,15,16,17,18,19,20,21,22,23,
+1};
        int             asubs[NADCSL] = {16,16,16,16,16,16,16,16,16,16,16,16,16,16,
+12};
```

```
        int             nchan[NADC];            /* number of adcs per slot */
        BUFF_DATA       old_length;             /* old buffer length */
        unsigned short  tdcdata[NTDC];          /* TDC buffer */
        int             cblock[4];              /* CAMAC control block */
        int             nrem;                   /* remainder word */
        int             nwords;                 /* number of 32 bit words */
        unsigned short  data=0;
        int             qres=0;
        int             j,k;                    /* loop counters */
        int             sim_map[NADCSL] = {6,0,5,4,3,2,13,7,12,11,10,9,1,8};


        /* increment event number */
        event_num++;

        /* save a space for the event length */
        old_length = spill[0];
        spill[0]++;

        /* event marker */
        spill[spill[0]] = trigger;
        spill[0]++;

        /* event number */
        spill[spill[0]] = event_num;
        spill[0]++;

        /* Read and Clear scalers */
        memset(scaldata, 0, sizeof(scaldata));
        memset(cblock, 0, sizeof(cblock));
        cblock[0] = NSCALER;

        read_mod24_wait(2, BRANCH, CRATE, 3, 4, 12, &cblock[1],scaldata);


        if (cblock[1] != 48)
        /*if (cblock[1] != NSCALER)*/
        {
                printf("read_spill: Spill %d only read %d SCALERs",spill_num, cbloc
+k[1]);
        }

        for(i=0; i<NSCALER; i++)
        {
                spill[spill[0]] = scaldata[i];
                spill[0]++;
                scalers[i] = scaldata[i];
                }

        /* Read and Clear TDCs */
        memset(tdcdata, 0, sizeof(tdcdata));
        memset(cblock, 0, sizeof(cblock));
        cblock[0] = NTDC;

        /* save the number of chan */

        spill[spill[0]] = cblock[1];
        spill[0]++;

        /* Pack 2 16 bit words into a 32 bit word */

        nrem=cblock[1]%2;
        nwords=(cblock[1]+nrem)/2;

        /* add an end of readout marker */
        spill[spill[0]] = 0xBEEFDEAD;
        spill[0]++;


        /* Save the event length */
        spill[old_length] = spill[0] - old_length;

        if( spill[0] > MAX_BUFF_SIZE)
```



```
        {
                printf("read_event: need a bigger spill buffer\n");
                return(-1);
        }

        if(event_num%1000 == 0)
        {
                printf(" %d events                                          \r",eve
+nt_num);
                fflush(stdout);
        }

        return(0);
 }


 int read_spill(
 BUFF_DATA       *spill, int norm_scal, time_t time_buf)                        /*
+spill buffer */
 {
        /* Reads end of spill event, puts into spill buffer */

        unsigned long   old_length;             /* old buffer length */


        /* spill number */
        spill_num++;

        /* save room for the EOS length */
        old_length = spill[0];
        spill[0]++;

        /* spill marker */
        spill[spill[0]] = SPILL;
        spill[0]++;

        /* spill num */
        spill[spill[0]] = spill_num;
        spill[0]++;

        /* save the time and date stamp */
        spill[spill[0]] = time(0);
        spill[0]++;
        /* save norm_scaler */
        spill[spill[0]] = norm_scal;
        spill[0]++;
        /* save time buffer filling time */
        spill[spill[0]] = time_buf;
        spill[0]++;
        /* save the EOS event length */
        spill[old_length] = spill[0] - old_length;

        if( spill[0] > MAX_BUFF_SIZE)
        {
                printf("read_spill: need a bigger spill buffer\n");
                return(-1);
        }

        printf("\n\nEnd of Spill %d, Event Number=%d\n",spill_num,event_num);

        return(0);
 }
 int end_camac(run_num)
 {
        printf("camac_end: End of run %d, %d spills, %d events\n",
                run_num, spill_num, event_num);
        return(0);
 }
```