#include "NIDAQmx.h"

#include "edtinc.h"

#include <fcntl.h>

#include <malloc.h>

#include <sys\types.h>

#include <sys\stat.h>

#include <time.h>

#include <omp.h>

#include <stdio.h>

#include <commctrl.h>

#include <sm.h>

#include "smDirectX.h"

#include "smfiles.h"

#include "smsecure.h"

#define DAVINCIBITFLIP

#define O\_MODE (O\_BINARY | O\_CREAT | O\_TRUNC | O\_SEQUENTIAL | O\_RDWR)

#define S\_MODE S\_IWRITE

extern PdvDev \*pdv\_pt[MAXCHANNELS], \*pdv\_p, \*pcd\_p;

extern signed short int \*image\_data, \*single\_img\_data;

extern int config\_num\_col, config\_num\_row, live\_factor, shutter\_wait, stim\_delay, curConfig, multi\_pix\_read, multi\_pix\_av, fd\_no\_deinterleave\_flag;

extern HWND main\_window\_handle;

extern int bad\_row\_ar[10][5], bad\_col\_ar[10][5], bad\_pixel\_ar[300][9];

extern char config\_list[8][50];

extern char home\_dir[MAX\_PATH];

extern int SFC\_flag;

int32 CVICALLBACK DoneCallback(TaskHandle taskHandle, int32 status, void \*callbackData);

void stripAD\_data(unsigned short int \*image\_data, signed short int \*BNC\_data, char \*cameraType, int file\_height,

int frame\_width, int frame\_height, int loops, int cds);

//void SM\_convertDiskData(long num\_frames, char \*DISKfilename, int f\_handle, int handleBNC, char \*cameraType, short int numBNC\_chan, short int BNC\_ratio, int file\_width, int file\_height,

// int frame\_width, int frame\_height, int stripes, int layers, int factor, int quadFlag, int cds, int rotateFlag, int twoKFlag);

static void saveNI\_data(long num\_frames, char filename[MAX\_PATH], short int numBNC\_chan, short int BNC\_ratio, float64 \*NI\_data, int delay, int isTwoK);

static void saveBNC\_data(long num\_frames, char filename[MAX\_PATH], short int numBNC\_chan, short int BNC\_ratio, signed short int \*BNC\_data, char \*cameraType, int cam\_num\_col, int stripes);

static void saveBNC\_block(long num\_frames, int handle, long tt\_frames\_flag, short int numBNC\_chan, short int BNC\_ratio, signed short int \*BNC\_data, char \*cameraType, long lastDiskFrame, int cam\_num\_col, int stripes);

static void SM\_makeHeader(int images, int exposures, int rows, int columns, char \*camera, double exposure);

static char FitsHeader[36][80];

static char FitsEnder[36\*80];

int lut\_size;

unsigned int \*lut = (unsigned int \*)NULL;

static unsigned short int \*image\_buffer = (unsigned short int \*)NULL;

static int cdsFlag = FALSE, OffCenter\_crop = FALSE, numBNC\_chan = 4, FastDiskFlag; // , skip\_chan = 0;

static char fastD\_drive[10];

SYSTEMTIME acq\_end\_sysTime;

char multi\_file\_names[50][\_MAX\_FNAME];

#ifdef DAVINCIBITFLIP

#define SB(P,B) (((unsigned \_\_int64)1)<<(((int)(P-'A')\*16)+B+2))

//unsigned short int imageIn[maxLength];

//unsigned \_\_int64 imageOut[maxLength / 4];

static unsigned int \*bitlut = (unsigned int \*)NULL;

unsigned \_\_int64 bitLut[0x40000];

unsigned \_\_int64 bitPattern[4][16] = {

{ 0x00000000,0x00000000,SB('A',12),SB('B',13),SB('A', 0),SB('B',11),SB('A',11),SB('B',12),SB('B',10),SB('B', 9),SB('A',10),SB('B', 7),SB('B', 6),SB('B', 5),SB('A', 9),SB('B', 8) },

{ 0x00000000,0x00000000,SB('B', 3),SB('B', 2),SB('B', 1),SB('B', 0),SB('A', 8),SB('B', 4),SB('C', 6),SB('C', 5),SB('C', 4),SB('C',13),SB('C', 3),SB('C', 1),SB('C', 7),SB('C', 2) },

{ 0x00000000,0x00000000,SB('C', 0),SB('C',11),SB('D',12),SB('D',11),SB('D',13),SB('C',12),SB('D',10),SB('C', 9),SB('D', 8),SB('D', 7),SB('D', 6),SB('D', 5),SB('D', 9),SB('C',10) },

{ 0x00000000,0x00000000,SB('D', 3),SB('D', 4),SB('C', 8),SB('D', 0),SB('D', 2),SB('D', 1),SB('A', 3),SB('A',13),SB('A', 4),SB('A', 7),SB('A', 2),SB('A', 1),SB('A', 5),SB('A', 6) } };

unsigned int bitMask[16] = { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800,0x1000,0x2000,0x4000,0x8000 };

#endif

extern int AD\_flag, NI\_flag, MNC\_flag, Bit16\_flag, pcdFlag, num\_trials;

int NI\_add\_frames = 0, shutter\_add\_frames;

//for monochromator

extern int ratio\_type, frame\_per\_ratio;

extern double MNC\_in1\_vol, MNC\_in2\_vol, MNC\_out1\_vol, MNC\_out2\_vol, MNC\_w1\_vol, MNC\_w2\_vol;

static int horizontalBin, binFlag = 0, twoKFlag = FALSE, Falcon\_flag = false, superFrame = 1, loops = 1, image\_layers, image\_stripes; // , twoK\_start\_missed = 0;

static unsigned int frame\_width, frame\_height, frame\_length,image\_width, image\_height, image\_length, file\_length;

unsigned int file\_width, file\_height;

#define DAQmxErrChk(functionCall) if( DAQmxFailed(error=(functionCall)) ) goto Error; else

static int initAcquire(int numChannels, int cdsF, int layers, int stripes, char \*cameraname, int quadFlag, int rotateFlag, int bin, int factor, int superframe)

{

char \*p;

numBNC\_chan = 4;

if (bin <= 1) {

horizontalBin = 1;

binFlag = 0;

}

else {

horizontalBin = bin;

binFlag = 1;

}

superFrame = superframe;

if (NI\_flag) {

if (strstr(cameraname, "-NIU8"))

numBNC\_chan = 8;

else

numBNC\_chan = 4;

}

image\_layers = layers;

image\_stripes = stripes;

FastDiskFlag = 0;

Falcon\_flag = false;

if (strstr(cameraname, "DM64"))

Falcon\_flag = true;

else if (strstr(cameraname, "DM2K") || strstr(cameraname, "DM1K")) {

twoKFlag = TRUE;

if (numChannels > 2)

image\_stripes <<= 1;

image\_layers <<= 1;

if (strstr(cameraname, "-OCR"))

OffCenter\_crop = 1;

if (p = strstr(cameraname, "-FASTD")) {

FastDiskFlag = 1;

strcpy(fastD\_drive, (p + 7));

\*(fastD\_drive + 1) = '\0';

char tmp\_str[256];

strcpy(tmp\_str, fastD\_drive);

if (!existDir(strcat(tmp\_str, ":\\"))) {

MessageBox(main\_window\_handle, "The PCI-SSD drive was not specified properly. It should be -FASTD-H where H (or other letter) is the name of the drive", "message", MB\_OK);

return FALSE;

}

}

}

cdsFlag = cdsF;

if (pcdFlag != 1) {

file\_width = image\_width = (frame\_width = pdv\_get\_width(pdv\_pt[0]));

file\_height = image\_height = (frame\_height = pdv\_get\_height(pdv\_pt[0]));

}

else {

int config\_id = checkCfgLUT();

file\_width = image\_width = (frame\_width = get\_pcdWH(config\_id, 0));

file\_height = image\_height = (frame\_height = get\_pcdWH(config\_id, 1));

}

file\_length = (image\_length = (frame\_length = frame\_width \* frame\_height \* 2));

if (twoKFlag) {

image\_width \*= numChannels>>1;

image\_height \*= 2;

image\_length \*= 2\*(numChannels >> 1);

if (superFrame > 1) {

file\_height = (file\_height / factor + 1) / superFrame - 2;

file\_length = file\_width\*file\_height \* 2;

}

else if (factor > 1) {

file\_height = file\_height / factor;

file\_length = file\_width\*file\_height \* 2;

}

file\_height \*= 2;

if (cdsFlag) {

file\_width /= 2 / (numChannels >> 1);

file\_length \*= (numChannels >> 1);

}

else {

file\_width \*= (numChannels >> 1);

file\_length \*= 2 \* (numChannels >> 1);

}

}

else {

if (cdsFlag)

file\_width = file\_width / 2;

if (factor > 1)

file\_height = file\_height / factor;

file\_length = file\_width\*file\_height \* 2;

}

if (AD\_flag) {

if (strstr(cameraname, "DW"))

file\_height -= 1;

else

file\_height -= 2;

file\_length = file\_width\*file\_height \* 2;

}

// skip\_chan = 0;

return TRUE;

}

/\*

void checkCropping(int numChannels, int crop\_w)

{

if (numChannels > 2 && (crop\_w <= file\_width / 4)) {

image\_width /= 4;

file\_width /= 4;

image\_length /= 4;

skip\_chan = 4;

}

else if (crop\_w <= file\_width / 2) {

image\_width /= 2;

file\_width /= 2;

image\_length /= 2;

skip\_chan = 2;

}

file\_length = file\_width \* file\_height \* 2;

}\*/

int initLive(int numChannels, int cdsF, int images, int layers, int stripes, char \*cameraname, int quadFlag, int rotateFlag, int bad\_pix\_index)

{

char badPixFname[\_MAX\_FNAME];

FILE \*fp\_bad;

unsigned int image\_height\_sv;

if (!initAcquire(numChannels, cdsF, layers, stripes, cameraname, quadFlag, rotateFlag, 1, 1, 1))

return FALSE;

loops = 1;

if (image\_buffer != NULL)

\_aligned\_free(image\_buffer);

image\_buffer = (unsigned short \*)\_aligned\_malloc(file\_length\*superFrame, 2);

if (lut != NULL)

\_aligned\_free(lut);

lut\_size = file\_length \* superFrame \* sizeof(lut[0]);

lut = (unsigned int \*)\_aligned\_malloc(lut\_size, 2);

image\_height\_sv = image\_height;

if (AD\_flag) {

if (strstr(cameraname, "DW"))

image\_height -= 1;

else

image\_height -= 2;

}

makeLookUpTable(lut, image\_width, image\_height, file\_width, file\_height, image\_stripes, image\_layers, superFrame, quadFlag, cdsFlag, rotateFlag, twoKFlag);

image\_height = image\_height\_sv;

if (pcdFlag != 1) {

for (int i = 0; i < numChannels; ++i)

pdv\_setsize(pdv\_pt[i], config\_num\_col, config\_num\_row\*live\_factor);

}

// correct bad rows and cols for DM2K or bad pixels for 128X - read map

if (bad\_pix\_index) {

unsigned int row = 0, tmp, bad\_x, bad\_y;

int xx[] = { 0, -1, -1, -1, 0, 0, 1, 1, 1 };

int yy[] = { 0, -1, 0, 1, -1, 1, -1, 0, 1 };

char buf[256], tempStr[100];

memset(bad\_pixel\_ar, 0, sizeof(bad\_pixel\_ar));

sprintf(badPixFname, ".\\SMSYSDAT\\bad\_pixels%d.txt", bad\_pix\_index);

if (fp\_bad = fopen(badPixFname, "r")) {

if (!fgets(buf, 255, fp\_bad))

goto exit\_pix;

sscanf(buf, "%s", tempStr);

while (strcmp(tempStr, "END")) {

if (row > sizeof(bad\_pixel\_ar) / sizeof(bad\_pixel\_ar[0]))

break;

tmp = atoi(tempStr) - 1;

if ((tmp / file\_width >= (file\_width - file\_height) / 2 + file\_height) || (tmp / file\_width < (file\_width - file\_height) / 2))

goto OUT\_PIX;

bad\_pixel\_ar[row][0] = (tmp / file\_width - (file\_width - file\_height) / 2)\*file\_width + (tmp % file\_width) + 1;

bad\_x = ((bad\_pixel\_ar[row][0] - 1) % file\_width);

bad\_y = (bad\_pixel\_ar[row][0] - 1) / file\_width;

bad\_x = min(max(0, bad\_x), file\_width - 1);

bad\_y = min(max(0, bad\_y), file\_height - 1);

int mm, l;

for (mm = 1; mm <= 8; mm++) {

if ((bad\_x + xx[mm] < 0) || (bad\_x + xx[mm] >= file\_width) || (bad\_y + yy[mm] < 0) || (bad\_y + yy[mm] >= file\_height))

goto BAD\_PIX;

for (l = 0; l < sizeof(bad\_pixel\_ar) / sizeof(bad\_pixel\_ar[0]); l++) {

if (bad\_pixel\_ar[l][mm] == 0)

break;

if (file\_width\*(bad\_y + yy[mm]) + bad\_x + xx[mm] + 1 == bad\_pixel\_ar[l][0])

goto BAD\_PIX;

}

bad\_pixel\_ar[row][mm] = file\_width \* (bad\_y + yy[mm]) + bad\_x + xx[mm] + 1;

BAD\_PIX:

mm = mm;

}

row++;

OUT\_PIX:

if (!fgets(buf, 255, fp\_bad))

goto exit\_pix;

sscanf(buf, "%s", tempStr);

}

exit\_pix:

fclose(fp\_bad);

for (unsigned int i = 0; i < row; i++) {

for (int mm = 1; mm <= 8; mm++) {

for (unsigned int l = 0; l < row; l++) {

if (bad\_pixel\_ar[i][mm] == bad\_pixel\_ar[l][0]) {

bad\_pixel\_ar[i][mm] = 0;

break;

}

}

}

}

}

}

else if (twoKFlag) {//check for bad\_row, bad\_col files

char tempStr[256];

char buf[256];

int line;

char \*badLineFmt = "%d %d %d %d %d\n";

sprintf(badPixFname, ".\\SMSYSDAT\\bad\_rows.txt");

if (fp\_bad = fopen(badPixFname, "r")) {

if (!fgets(buf, 255, fp\_bad))

goto exit\_row;

line = 0;

if (!fgets(buf, 255, fp\_bad))

goto exit\_row;

sscanf(buf, "%s", tempStr);

while (strcmp(tempStr, "END")) {

sscanf(buf, badLineFmt, &bad\_row\_ar[line][0], &bad\_row\_ar[line][1], &bad\_row\_ar[line][2], &bad\_row\_ar[line][3], &bad\_row\_ar[line][4]);

if (!fgets(buf, 255, fp\_bad))

goto exit\_row;

sscanf(buf, "%s", tempStr);

line++;

}

exit\_row:

fclose(fp\_bad);

}

sprintf(badPixFname, ".\\SMSYSDAT\\bad\_cols.txt");

if (fp\_bad = fopen(badPixFname, "r")) {

if (!fgets(buf, 255, fp\_bad))

goto exit\_col;

line = 0;

if (!fgets(buf, 255, fp\_bad))

goto exit\_col;

sscanf(buf, "%s", tempStr);

while (strcmp(tempStr, "END")) {

sscanf(buf, badLineFmt, &bad\_col\_ar[line][0], &bad\_col\_ar[line][1], &bad\_col\_ar[line][2], &bad\_col\_ar[line][3], &bad\_col\_ar[line][4]);

if (!fgets(buf, 255, fp\_bad))

goto exit\_col;

sscanf(buf, "%s", tempStr);

line++;

}

exit\_col:

fclose(fp\_bad);

}

}

return TRUE;

}

void multiPixReadAv(signed short \*img)

{

if (multi\_pix\_av) {

int img\_jump = file\_width / multi\_pix\_read;

signed short \*scr\_pt = img;

signed short \*dst\_pt = img;

long pix\_sum;

for (unsigned int l = 0; l < file\_height; l++) {

for (int n = 0; n < img\_jump; n++) {

pix\_sum = 0;

for (int m = 0; m < multi\_pix\_read; m++)

pix\_sum += \*(scr\_pt + img\_jump \* m);

\*dst\_pt = (signed short)(pix\_sum / multi\_pix\_read);

scr\_pt++;

dst\_pt++;

}

scr\_pt += img\_jump \* (multi\_pix\_read - 1);

dst\_pt += img\_jump \* (multi\_pix\_read - 1);

}

}

}

int SM\_take\_Live(signed short \*live\_img, int numChannels, int cdsF, int images, int layers, int stripes, int factor, char \*cameraname, int file\_img\_w, int hbin, int quadFlag, int rotateFlag,

int bad\_pix\_index, int ndrFlag, int num\_NDRsub\_frames, int segmented)

{

int i;

u\_char \*data\_ptr[MAXCHANNELS] = { NULL, NULL, NULL, NULL };

u\_char \*image\_ptr;

int times[MAXCHANNELS][2];

signed short int \*BNC\_data = (signed short int \*)NULL;

signed short int \*live\_img\_ptr;

char buf[200];

initAcquire(numChannels, cdsF, layers, stripes, cameraname, quadFlag, rotateFlag, hbin, factor, 1);

if (ndrFlag)

loops = max(2, (num\_NDRsub\_frames+1) / factor + (((num\_NDRsub\_frames+1) % factor )>0));

else

loops = (images + cdsFlag ) / (factor \* superFrame) + (((images + cdsFlag) % (factor \* superFrame))>0);

if (AD\_flag)

BNC\_data = (signed short \*)\_aligned\_malloc(loops\*image\_width \* 2 \* sizeof(BNC\_data[0]), 2);

if (!file\_img\_w)

file\_img\_w = file\_width;

for (i = 0; i < numChannels; ++i)

data\_ptr[i] = (u\_char \*)live\_img + i\*frame\_length / factor;

if (pcdFlag) {

int bufsize = image\_length; // image\_length;

int numbufs = 4;

PdvDev \*pdv\_p\_sv = pdv\_p;

// if (pcdFlag == 2)

pdv\_p = pcd\_p;

edt\_reg\_write(pdv\_p, SSD16\_CHEN, 0x0);

edt\_flush\_channel(pdv\_p, 0);

if (edt\_configure\_ring\_buffers(pdv\_p, bufsize, numbufs, EDT\_READ, NULL) != -1)

{

edt\_start\_buffers(pdv\_p, numbufs);

edt\_reg\_write(pdv\_p, SSD16\_CHEN, 0x0001);

image\_ptr = edt\_wait\_for\_buffers(pdv\_p, 1);

memcpy(data\_ptr[0], image\_ptr, frame\_length);

data\_ptr[0] += image\_length;

}

else

MessageBox(main\_window\_handle, "edt\_configure\_ring\_buffers failed", "message", MB\_OK);

pdv\_p = pdv\_p\_sv;

}

else {

for (i = 0; i < numChannels; ++i) {

pdv\_flush\_fifo(pdv\_pt[i]);

pdv\_multibuf(pdv\_pt[i], 4); //has to be 4

}

if (numChannels > 1 && omp\_get\_num\_procs() >= numChannels) {

if (ndrFlag) {

int mismatchFlag;

for (int n = 0; n < 20; n++) {

omp\_set\_num\_threads(numChannels);

#pragma omp parallel private(image\_ptr, buf)

{

int thread\_id;

thread\_id = omp\_get\_thread\_num();

pdv\_start\_images(pdv\_pt[thread\_id], 4);

image\_ptr = pdv\_wait\_image\_timed\_raw(pdv\_pt[thread\_id], (unsigned int \*)&times[thread\_id][0], 1);

pdv\_start\_image(pdv\_pt[thread\_id]);

for (int i = 0; i < factor; ++i) {

memcpy(data\_ptr[thread\_id], image\_ptr, frame\_length / factor);

image\_ptr += frame\_length / factor;

data\_ptr[thread\_id] += image\_length / factor;

}

mismatchFlag = 1;

if ((numChannels < 4 && (abs(times[1][1] - times[0][1]) < 100000)) || ((abs(times[1][1] - times[0][1]) < 100000) && (abs(times[2][1] - times[0][1]) < 100000) && (abs(times[3][1] - times[0][1]) < 100000))) {

for (int j = 1; j < loops; j++) {

image\_ptr = pdv\_wait\_image\_raw(pdv\_pt[thread\_id]);

pdv\_start\_image(pdv\_pt[thread\_id]);

}

for (int i = 0; i < factor; ++i) {

memcpy(data\_ptr[thread\_id], image\_ptr, frame\_length / factor);

image\_ptr += frame\_length / factor;

data\_ptr[thread\_id] += image\_length / factor;

}

mismatchFlag = 0;

}

}

if (!mismatchFlag) {

int min\_ndr = 0;

unsigned int single\_img\_len = image\_length / (2 \* factor);

long ndr\_jump = single\_img\_len \* (factor + num\_NDRsub\_frames % factor);

live\_img\_ptr = (signed short int \*)live\_img;

for (int j = 0; j < (int)(single\_img\_len); live\_img\_ptr++, j++) {

\*live\_img\_ptr = \*(live\_img\_ptr + ndr\_jump) - \*live\_img\_ptr;

if (min\_ndr > \*live\_img\_ptr)

min\_ndr = \*live\_img\_ptr;

}

if (min\_ndr > -200)

break;

}

for (i = 0; i < numChannels; ++i)

data\_ptr[i] = (u\_char \*)live\_img + i \* frame\_length / factor;

for (i = 0; i < numChannels; ++i) {

pdv\_flush\_fifo(pdv\_pt[i]);

pdv\_multibuf(pdv\_pt[i], 4); //has to be 4

}

// sprintf(buf, "n: %d, Timging Diff: %d %d %d\n", n, (times[1][1] - times[0][1]) / 1000, (times[2][1] - times[0][1]) / 1000, (times[3][1] - times[0][1]) / 1000);

// OutputDebugString(buf);

}

}

else {

for (int m = 0; m < 20; m++) {

omp\_set\_num\_threads(numChannels);

#pragma omp parallel private(image\_ptr, buf)

{

int thread\_id;

thread\_id = omp\_get\_thread\_num();

pdv\_start\_images(pdv\_pt[thread\_id], loops + (factor > 1 ? 1 : 0));

for (int j = 0; j < loops; j++) {

image\_ptr = pdv\_wait\_image\_timed\_raw(pdv\_pt[thread\_id], (unsigned int \*)&times[thread\_id][0], 1);

// image\_ptr = pdv\_wait\_image\_raw(pdv\_pt[thread\_id]);

for (int i = 0; i < factor; ++i) {

memcpy(data\_ptr[thread\_id], image\_ptr, frame\_length / factor);

image\_ptr += frame\_length / factor;

data\_ptr[thread\_id] += image\_length / factor;

}

}

}

if (numChannels < 4 && (abs(times[1][1] - times[0][1]) < 100000))

break;

else if ((abs(times[1][1] - times[0][1]) < 100000) && (abs(times[2][1] - times[0][1]) < 100000) && (abs(times[3][1] - times[0][1]) < 100000))

break;

// sprintf(buf, "\*\*\*\* %d: %d %d %d\n", m, (times[1][1] - times[0][1]) / 1000, (times[2][1] - times[0][1]) / 1000, (times[3][1] - times[0][1]) / 1000);

// OutputDebugString(buf);

for (int i = 0; i < numChannels; ++i)

data\_ptr[i] = (u\_char \*)live\_img + i \* frame\_length / factor;

if (factor > 1) {

for (i = 0; i < numChannels; ++i) {

pdv\_flush\_fifo(pdv\_pt[i]);

pdv\_multibuf(pdv\_pt[i], 4); //has to be 4

}

}

}

}

}

else {

pdv\_start\_images(pdv\_pt[0], loops + (factor > 1 ? 1 : 0));

for (int j = 0; j < loops; j++) {

image\_ptr = pdv\_wait\_image\_raw(pdv\_pt[0]);

memcpy(data\_ptr[0], image\_ptr, frame\_length);

data\_ptr[0] += image\_length;

}

if (Falcon\_flag) {

unsigned \_\_int64 \*long\_pt = (unsigned \_\_int64 \*)live\_img;

unsigned short int \*short\_pt0, \*short\_pt1, \*short\_pt2, \*short\_pt3;

short\_pt0 = (unsigned short int \*)long\_pt;

short\_pt1 = short\_pt0 + 1;

short\_pt2 = short\_pt1 + 1;

short\_pt3 = short\_pt2 + 1;

for (i = 0; i < (signed)frame\_length >> 2; i++) {

\*long\_pt++ = (bitLut[\*short\_pt0] | bitLut[(\*short\_pt1) | 0x10000L] | bitLut[(\*short\_pt2) | 0x20000L] | bitLut[(\*short\_pt3) | 0x30000L]) & 0x3fff3fff3fff3fffL;

short\_pt0 += 4;

short\_pt1 += 4;

short\_pt2 += 4;

short\_pt3 += 4;

}

}

if (ndrFlag > 1) {

memcpy(live\_img, live\_img + file\_length \* (ndrFlag - 1)/2, file\_length);

}

}

}

if (AD\_flag) {

if (factor > 1) {

frame\_height /= factor;

factor = 1;

loops = images;

}

stripAD\_data((unsigned short \*)live\_img, BNC\_data, cameraname, file\_height, frame\_width, frame\_height, loops, cdsFlag);

image\_height = file\_height;

image\_length = image\_width\*image\_height \* 2;

}

if (factor > 1) {

image\_height = file\_height;

image\_length = image\_width\*image\_height \* 2;

factor = 1;

loops = (images + cdsFlag) / superFrame + (((images + cdsFlag) % superFrame)>0);

}

if (Bit16\_flag) {

unsigned short \*img\_ptr = (unsigned short \*)live\_img;

for (i = 0; i < (int)(image\_length\*loops>>1); i++)

\*img\_ptr++ = (\*img\_ptr >> 2);

}

if (cdsFlag)

subtractCDS((unsigned short \*)live\_img, loops, image\_width, image\_height, image\_length, factor, quadFlag, twoKFlag, numChannels);

if (cdsFlag)

--loops;

data\_ptr[0] = (u\_char \*)live\_img;

unsigned int file\_length\_sv = file\_length;

for (int j = 0, k = 0; j < loops; ++j, k += superFrame) {

file\_length = file\_length\_sv;

frame\_deInterleave(superFrame\*file\_length >> 1, lut, (unsigned short \*)data\_ptr[0], image\_buffer);

frameRepair(image\_buffer, bad\_pix\_index, twoKFlag, image\_width, file\_width, file\_height, horizontalBin, superFrame, file\_img\_w, binFlag, segmented);

file\_length = file\_height \* min(file\_img\_w, (signed int)file\_width) \* 2 / horizontalBin;

multiPixReadAv((signed short \*)image\_buffer);

memcpy(data\_ptr[0], (void \*)image\_buffer, (k <= images ? superFrame : images % superFrame)\*file\_length);

data\_ptr[0] += image\_length;

}

if (AD\_flag) {

if (BNC\_data != NULL)

\_aligned\_free(BNC\_data);

BNC\_data = NULL;

}

return TRUE;

}

extern int runPHT\_flag;

int SM\_take\_Dark(signed short \*dk\_img, int images, int numChannels, int cdsF, int layers, int stripes, double frame\_interval, char \*cameraname, int file\_img\_w, int hbin, int quadFlag, int rotateFlag,

int bad\_pix\_index, int ndrFlag, int num\_NDRsub\_frames, int segmented)

{

int i;

u\_char \*data\_ptr[MAXCHANNELS] = { NULL, NULL, NULL, NULL };

u\_char \*image\_ptr;

signed short \*dark\_imgs, \*dk\_ptr;

int times[MAXCHANNELS][2];

signed short int \*BNC\_data = (signed short int \*)NULL;

signed short int \*dk\_img\_ptr;

char buf[200];

long dataLength;

int factor;

superFrame = 1;

// images = 16;

if (numChannels > 1)

factor = 10 \* (int)(1024 / config\_num\_row);

else

factor = 20; // (int)max(20, 20 \* (1 + cdsF) / frame\_interval);

if (pcdFlag != 1) {

for (i = 0; i < numChannels; ++i) {

pdv\_setsize(pdv\_pt[i], config\_num\_col, config\_num\_row\*factor);

if (!pcdFlag) {

pdv\_flush\_fifo(pdv\_pt[i]);

pdv\_multibuf(pdv\_pt[i], 4); //has to be 4

}

}

}

initAcquire(numChannels, cdsF, layers, stripes, cameraname, quadFlag, rotateFlag, hbin, factor, 1);

loops = (images + cdsFlag) / (factor \* superFrame) + (((images + cdsFlag) % (factor \* superFrame))>0);

if (AD\_flag)

BNC\_data = (signed short \*)\_aligned\_malloc(loops\*image\_width \* 2 \* sizeof(BNC\_data[0]), 2);

if (!file\_img\_w)

file\_img\_w = file\_width;

dataLength = (unsigned long long)(loops + 1)\*(unsigned long long)image\_length;

dark\_imgs = (signed short \*)\_aligned\_malloc(dataLength, 2);

for (i = 0; i < numChannels; ++i)

data\_ptr[i] = (u\_char \*)dark\_imgs + i \* frame\_length / factor;

if (numChannels > 1 && omp\_get\_num\_procs() >= numChannels) {

if (ndrFlag) {

int mismatchFlag;

for (int n = 0; n < 20; n++) {

omp\_set\_num\_threads(numChannels);

#pragma omp parallel private(image\_ptr, buf)

{

int thread\_id;

thread\_id = omp\_get\_thread\_num();

pdv\_start\_images(pdv\_pt[thread\_id], 4);

image\_ptr = pdv\_wait\_image\_timed\_raw(pdv\_pt[thread\_id], (unsigned int \*)&times[thread\_id][0], 1);

pdv\_start\_image(pdv\_pt[thread\_id]);

for (int i = 0; i < factor; ++i) {

memcpy(data\_ptr[thread\_id], image\_ptr, frame\_length / factor);

image\_ptr += frame\_length / factor;

data\_ptr[thread\_id] += image\_length / factor;

}

mismatchFlag = 1;

if ((numChannels < 4 && (abs(times[1][1] - times[0][1]) < 100000)) || ((abs(times[1][1] - times[0][1]) < 100000) && (abs(times[2][1] - times[0][1]) < 100000) && (abs(times[3][1] - times[0][1]) < 100000))) {

for (int j = 1; j < loops; j++) {

image\_ptr = pdv\_wait\_image\_raw(pdv\_pt[thread\_id]);

pdv\_start\_image(pdv\_pt[thread\_id]);

}

for (int i = 0; i < factor; ++i) {

memcpy(data\_ptr[thread\_id], image\_ptr, frame\_length / factor);

image\_ptr += frame\_length / factor;

data\_ptr[thread\_id] += image\_length / factor;

}

mismatchFlag = 0;

}

}

if (!mismatchFlag) {

int min\_ndr = 0;

unsigned int single\_img\_len = image\_length / (2 \* factor);

long ndr\_jump = single\_img\_len \* (factor + num\_NDRsub\_frames % factor);

dk\_img\_ptr = (signed short int \*)dark\_imgs;

for (int j = 0; j < (int)(single\_img\_len); dk\_img\_ptr++, j++) {

\*dk\_img\_ptr = \*(dk\_img\_ptr + ndr\_jump) - \*dk\_img\_ptr;

if (min\_ndr > \*dk\_img\_ptr)

min\_ndr = \*dk\_img\_ptr;

}

if (min\_ndr > -200)

break;

}

for (i = 0; i < numChannels; ++i)

data\_ptr[i] = (u\_char \*)dark\_imgs + i \* frame\_length / factor;

for (i = 0; i < numChannels; ++i) {

pdv\_flush\_fifo(pdv\_pt[i]);

pdv\_multibuf(pdv\_pt[i], 4); //has to be 4

}

// sprintf(buf, "n: %d, Timging Diff: %d %d %d\n", n, (times[1][1] - times[0][1]) / 1000, (times[2][1] - times[0][1]) / 1000, (times[3][1] - times[0][1]) / 1000);

// OutputDebugString(buf);

}

}

else {

for (int m = 0; m < 20; m++) {

omp\_set\_num\_threads(numChannels);

#pragma omp parallel private(image\_ptr, buf)

{

int thread\_id;

thread\_id = omp\_get\_thread\_num();

pdv\_start\_images(pdv\_pt[thread\_id], loops + (factor>1 ? 1 : 0));

for (int j = 0; j < loops; j++) {

image\_ptr = pdv\_wait\_image\_timed\_raw(pdv\_pt[thread\_id], (unsigned int \*)&times[thread\_id][0], 1);

// image\_ptr = pdv\_wait\_image\_raw(pdv\_pt[thread\_id]);

for (int i = 0; i < factor; ++i) {

memcpy(data\_ptr[thread\_id], image\_ptr, frame\_length / factor);

image\_ptr += frame\_length / factor;

data\_ptr[thread\_id] += image\_length / factor;

}

}

}

if (numChannels < 4 && (abs(times[1][1] - times[0][1]) < 100000))

break;

else if ((abs(times[1][1] - times[0][1]) < 100000) && (abs(times[2][1] - times[0][1]) < 100000) && (abs(times[3][1] - times[0][1]) < 100000))

break;

// sprintf(buf, "\*\*\*\* %d: %d %d %d\n", m, (times[1][1] - times[0][1]) / 1000, (times[2][1] - times[0][1]) / 1000, (times[3][1] - times[0][1]) / 1000);

// OutputDebugString(buf);

for (int i = 0; i < numChannels; ++i)

data\_ptr[i] = (u\_char \*)dark\_imgs + i \* frame\_length / factor;

if (factor > 1) {

for (i = 0; i < numChannels; ++i) {

pdv\_flush\_fifo(pdv\_pt[i]);

pdv\_multibuf(pdv\_pt[i], 4); //has to be 4

}

}

}

}

}

else {

pdv\_start\_images(pdv\_pt[0], loops + (factor>1 ? 1 : 0));

for (int j = 0; j < loops; j++) {

image\_ptr = pdv\_wait\_image\_raw(pdv\_pt[0]);

memcpy(data\_ptr[0], image\_ptr, frame\_length);

data\_ptr[0] += image\_length;

}

}

if (AD\_flag) {

if (factor > 1) {

frame\_height /= factor;

factor = 1;

loops = images;

}

stripAD\_data((unsigned short \*)dark\_imgs, BNC\_data, cameraname, file\_height, frame\_width, frame\_height, loops, cdsFlag);

image\_height = file\_height;

image\_length = image\_width \* image\_height \* 2;

}

if (factor > 1) {

image\_height = file\_height;

image\_length = image\_width \* image\_height \* 2;

factor = 1;

loops = (images + cdsFlag) / superFrame + (((images + cdsFlag) % superFrame)>0);

}

if (Bit16\_flag) {

unsigned short \*img\_ptr = (unsigned short \*)dark\_imgs;

for (i = 0; i < (int)(image\_length\*loops>>1); i++)

\*img\_ptr++ = (\*img\_ptr >> 2);

}

if (cdsFlag)

subtractCDS((unsigned short \*)dark\_imgs, loops, image\_width, image\_height, image\_length, factor, quadFlag, twoKFlag, numChannels);

if (cdsFlag)

--loops;

data\_ptr[0] = (u\_char \*)dark\_imgs;

unsigned int file\_length\_sv = file\_length;

for (int j = 0, k = 0; j < loops; ++j, k += superFrame) {

file\_length = file\_length\_sv;

frame\_deInterleave(superFrame\*file\_length >> 1, lut, (unsigned short \*)data\_ptr[0], image\_buffer);

frameRepair(image\_buffer, bad\_pix\_index, twoKFlag, image\_width, file\_width, file\_height, horizontalBin, superFrame, file\_img\_w, binFlag, segmented);

file\_length = file\_height \* min(file\_img\_w, (signed int)file\_width) \* 2 / horizontalBin;

multiPixReadAv((signed short \*)image\_buffer);

memcpy(data\_ptr[0], (void \*)image\_buffer, (k <= images ? superFrame : images % superFrame)\*file\_length);

data\_ptr[0] += image\_length;

}

if (!runPHT\_flag) {

long this\_sum;

dk\_ptr = dk\_img;

for (unsigned int m = 0; m < file\_length >> 1; m++, dk\_ptr++) {

dk\_img\_ptr = dark\_imgs + m;

this\_sum = (long)\*dk\_img\_ptr;

for (int j = 1; j < loops; ++j) {

dk\_img\_ptr += image\_length >> 1;

this\_sum += \*dk\_img\_ptr;

}

\*dk\_ptr = (signed short)(this\_sum/loops);

}

}

else

memcpy(dk\_img, dark\_imgs, file\_length\*images);

if (pcdFlag != 1) {

for (int i = 0; i < numChannels; ++i)

pdv\_setsize(pdv\_pt[i], config\_num\_col, config\_num\_row\*live\_factor);

}

if (AD\_flag) {

if (BNC\_data != NULL)

\_aligned\_free(BNC\_data);

BNC\_data = NULL;

}

if (dark\_imgs != NULL)

\_aligned\_free(dark\_imgs);

return TRUE;

}

static HWND acq\_hwndPB = NULL;

static int num\_steps = 30;

void createAcqProgBar()

{

// making a progress bar

RECT rcClient; // Client area of parent window.

int cyVScroll = 50; // Height of scroll bar arrow.

GetClientRect(main\_window\_handle, &rcClient);

acq\_hwndPB = CreateWindowEx(0, PROGRESS\_CLASS, (LPTSTR)NULL,

WS\_CHILD | WS\_VISIBLE | WS\_CAPTION, rcClient.left + rcClient.right / 6,

rcClient.top + (rcClient.bottom - cyVScroll) \* 2 / 5,

rcClient.right / 4, cyVScroll,

main\_window\_handle, NULL, NULL, NULL);

SendMessage(acq\_hwndPB, PBM\_SETRANGE, 0, MAKELPARAM(0, num\_steps+1));

SendMessage(acq\_hwndPB, PBM\_SETSTEP, (WPARAM)1, 0);

SendMessage(acq\_hwndPB, PBM\_STEPIT, 0, 0);

}

// this is only for slow frame rate - streaming to disk for McCormik

static int Ddisk\_flag = 0;

int SM\_take\_disk(int numChannels, char \*file\_name, int cdsF, int images, int layers, int stripes, double frame\_interval, int factor, char \*cameraname, int file\_img\_w, int hbin, int BNC\_ratio,

int ExtTriggerFlag, int preTrigger\_frames, int quadFlag, int rotateFlag, int bad\_pix\_index, int segmented)

{

int i, j;

int numFiles = 1;

int overrun[MAXCHANNELS], overruns[MAXCHANNELS] = { 0,0,0,0 }, timeouts[MAXCHANNELS] = { 0,0,0,0 };// , last\_timeouts = 0, recovering\_timeout = FALSE;

int useFrameFormat = FALSE;

char frameFormat[\_MAX\_FNAME] = "";

char tsmfname[MAXCHANNELS][\_MAX\_FNAME] = { "","","","" };

int tsmhandle[MAXCHANNELS] = { NULL, NULL, NULL, NULL };

int debug = FALSE;

u\_char \*data\_ptr[MAXCHANNELS];

u\_char \*image\_ptr, \*data\_pt;

signed short \*tmp\_frame = NULL;

unsigned long long dataLength, fileDataLength;

unsigned long dataLeft;

int numExposures = 1;

char edt\_devname[128] = "";

char errorBuf[256] = "";

long triggerPos = 0;

int times[MAXCHANNELS][2];

char buf[200];

int start\_frames = 64;

int esc\_hit = 0;

loops = 1;

superFrame = 1;

strcpy(tsmfname[0], file\_name);

if (strlen(tsmfname[0])) {

if ((tsmhandle[0] = \_open(tsmfname[0], O\_MODE, S\_MODE)) == -1) {

sprintf(errorBuf, "Cannot open %s\n", tsmfname[0]);

debug = TRUE;

goto cleanUp;

}

}

else

return 0;

if (pcdFlag != 1) {

for (i = 0; i < numChannels; ++i) {

pdv\_setsize(pdv\_pt[i], config\_num\_col, config\_num\_row\*factor);

if (!pcdFlag) {

pdv\_flush\_fifo(pdv\_pt[i]);

pdv\_multibuf(pdv\_pt[i], 4); //has to be 4

}

}

}

initAcquire(numChannels, cdsF, layers, stripes, cameraname, quadFlag, rotateFlag, hbin, factor, 1);

if (NI\_flag)

NI\_add\_frames = 2;

else

NI\_add\_frames = 0;

if (!NI\_flag && !AD\_flag) {

char BNCname[\_MAX\_PATH];

char \*p;

strcpy(BNCname, file\_name);

p = strstr(BNCname, ".tsm");

\*p = '\0';

strcat(BNCname, ".tbn");

DeleteFile(BNCname);

}

loops = (images + cdsFlag + NI\_add\_frames) / (factor \* superFrame) + (((images + cdsFlag + NI\_add\_frames) % (factor \* superFrame)) > 0);

if (!file\_img\_w)

file\_img\_w = file\_width;

if (twoKFlag && ((file\_img\_w != file\_width) || (horizontalBin > 1)))

SM\_makeHeader(images, numExposures, file\_height, min(file\_img\_w, (signed int)file\_width) / horizontalBin, cameraname, (double)frame\_interval / 1000.0);

else

SM\_makeHeader(images, numExposures, file\_height, file\_width, cameraname, (double)frame\_interval / 1000.0);

if (AD\_flag)

++loops;

dataLength = 2 \* (unsigned long long)image\_length;

fileDataLength = (unsigned long long)(images + 1)\*(unsigned long long)file\_length\*superFrame;

dataLeft = sizeof(FitsHeader) - (fileDataLength) % sizeof(FitsHeader);

memset(&FitsEnder, 0, min(sizeof(FitsEnder), dataLeft));

if (image\_data != NULL)

\_aligned\_free(image\_data);

if ((image\_data = (signed short \*)\_aligned\_malloc(dataLength, 2)) == NULL) {

// writeToDisk = TRUE;

strcpy(errorBuf, "Not enough memory for Data");

}

if (tmp\_frame != NULL)

\_aligned\_free(tmp\_frame);

tmp\_frame = (signed short \*)\_aligned\_malloc(image\_length / factor, 2);

if (twoKFlag)

for (i = 0; i < numChannels; ++i)

data\_ptr[i] = (u\_char \*)image\_data + i \* frame\_length / factor;

else

data\_ptr[0] = (u\_char \*)image\_data;

\_lseek(tsmhandle[0], 0L, SEEK\_SET);

\_write(tsmhandle[0], (char \*)FitsHeader, sizeof(FitsHeader));

createAcqProgBar();

int step\_cnt = 0;

double step\_size = ((double)num\_steps) / loops;

char str[256];

sprintf(str, "Acquired %d of %d images", 0, images);

SetWindowText(acq\_hwndPB, str);

unsigned int image\_height1 = image\_height / factor;

unsigned int image\_length1 = image\_length / factor;

int loops1 = factor;

long img\_cnt = 0;

Ddisk\_flag = 1;

for (i = 0; i < numChannels; ++i)

pdv\_start\_images(pdv\_pt[i], start\_frames);

if (numChannels > 1 && omp\_get\_num\_procs() >= numChannels) {

omp\_set\_num\_threads(numChannels);

for (j = 0; j < loops; j++) {

#pragma omp parallel private(image\_ptr)

{

int thread\_id, i;

thread\_id = omp\_get\_thread\_num();

image\_ptr = pdv\_wait\_image\_timed\_raw(pdv\_pt[thread\_id], (unsigned int \*)&times[thread\_id][0], 1);

// image\_ptr = pdv\_wait\_image\_raw(pdv\_pt[thread\_id]);

if ((overrun[thread\_id] = (edt\_reg\_read(pdv\_pt[thread\_id], PDV\_STAT) & PDV\_OVERRUN)))

++overruns[thread\_id];

for (i = 0; i < factor; ++i) {

memcpy(data\_ptr[thread\_id], image\_ptr, frame\_length / factor);

image\_ptr += frame\_length / factor;

data\_ptr[thread\_id] += image\_length / factor;

}

data\_ptr[thread\_id] -= image\_length;

if (!j)

data\_ptr[thread\_id] += image\_length / factor;

}

if (j < loops - start\_frames + 1)

for (i = 0; i < numChannels; ++i)

pdv\_start\_image(pdv\_pt[i]);

if (cdsFlag) {

data\_pt = (u\_char \*)image\_data + image\_length;

if (!j)

data\_pt -= image\_length / factor;

memcpy(tmp\_frame, data\_pt, image\_length / factor);

subtractCDS((unsigned short \*)image\_data, loops1, image\_width, image\_height1, image\_length1, 1, quadFlag, twoKFlag, numChannels);

}

data\_pt = (u\_char \*)image\_data;

unsigned int file\_length\_sv = file\_length;

for (int m = 0; m < loops1 - cdsFlag; ++m) {

file\_length = file\_length\_sv;

frame\_deInterleave(superFrame\*file\_length >> 1, lut, (unsigned short \*)data\_pt, image\_buffer);

frameRepair(image\_buffer, bad\_pix\_index, twoKFlag, image\_width, file\_width, file\_height, horizontalBin, superFrame, file\_img\_w, binFlag, segmented);

file\_length = file\_height \* min(file\_img\_w, (signed int)file\_width) \* 2 / horizontalBin;

\_write(tsmhandle[0], (void \*)image\_buffer, file\_length);

data\_pt += image\_length1;

img\_cnt++;

if (img\_cnt >= images)

break;

}

file\_length = file\_length\_sv;

if (cdsFlag) {

loops1 = factor + 1;

memcpy(image\_data, tmp\_frame, image\_length / factor);

}

sprintf(str, "Acquired %d of %d images", j\*factor, images);

SetWindowText(acq\_hwndPB, str);

while ((int)(j\*step\_size + 0.5) >= step\_cnt && step\_cnt < num\_steps) {

SendMessage(acq\_hwndPB, PBM\_STEPIT, 0, 0);

step\_cnt++;

}

if (KEY\_DOWN(VK\_ESCAPE)) {

esc\_hit = 1;

break;

}

// memcpy(single\_img\_data, image\_buffer, file\_length);

// getDispImage(0);

}

//test quadrant shift using time stamps

int quad\_shifts[4];

int min\_shift = 0, isShifted = 0;

quad\_shifts[0] = 0;

sprintf(buf, "%d %d %d\n", (times[1][1] - times[0][1]) / 1000, (times[2][1] - times[0][1]) / 1000, (times[3][1] - times[0][1]) / 1000);

OutputDebugString(buf);

}

else {

/\* for (j = 0; j < loops; j++) {

for (i = 0; i < numChannels; ++i) {

image\_ptr = pdv\_wait\_image\_raw(pdv\_pt[i]);

if ((overrun[i] = (edt\_reg\_read(pdv\_pt[i], PDV\_STAT) & PDV\_OVERRUN)))

++overruns[i];

memcpy(data\_ptr[i], image\_ptr, frame\_length);

data\_ptr[i] += image\_length;

}

}\*/

}

SM\_stopAD();

if (esc\_hit) {

images = img\_cnt;

if (twoKFlag && ((file\_img\_w != file\_width) || (horizontalBin > 1)))

SM\_makeHeader(images, numExposures, file\_height, min(file\_img\_w, (signed int)file\_width) / horizontalBin, cameraname, (double)frame\_interval / 1000.0);

else

SM\_makeHeader(images, numExposures, file\_height, file\_width, cameraname, (double)frame\_interval / 1000.0);

dataLength = 2 \* (unsigned long long)image\_length;

fileDataLength = (unsigned long long)(images + 1)\*(unsigned long long)file\_length\*superFrame;

dataLeft = sizeof(FitsHeader) - (fileDataLength) % sizeof(FitsHeader);

memset(&FitsEnder, 0, min(sizeof(FitsEnder), dataLeft));

}

memset(data\_ptr[0], 0, file\_length);

\_write(tsmhandle[0], (void \*)data\_ptr[0], file\_length);

\_write(tsmhandle[0], (char \*)FitsEnder, dataLeft);

if (esc\_hit) {

\_lseek(tsmhandle[0], 0L, SEEK\_SET);

\_write(tsmhandle[0], (char \*)FitsHeader, sizeof(FitsHeader));

}

for (i = 0; i < numChannels; ++i)

timeouts[i] = pdv\_timeouts(pdv\_pt[i]);

DestroyWindow(acq\_hwndPB);

acq\_hwndPB = NULL;

Ddisk\_flag = 0;

cleanUp:

if (debug)

MessageBox(main\_window\_handle, errorBuf, "message", MB\_OK);

if (pcdFlag != 1) {

for (int i = 0; i < numChannels; ++i)

pdv\_setsize(pdv\_pt[i], config\_num\_col, config\_num\_row\*live\_factor);

}

for (i = numFiles - 1; i >= 0; --i) {

if (tsmhandle[i]) {

\_close(tsmhandle[i]);

}

}

/\* if (AD\_flag) {

if (BNC\_data != NULL)

\_aligned\_free(BNC\_data);

BNC\_data = NULL;

}

else if (NI\_flag) {

if (NI\_data) {

if (NI\_data != NULL)

\_aligned\_free(NI\_data);

NI\_data = NULL;

}

}\*/

return 1;

}

static int loops\_sv = 0, NI\_add\_frames\_sv = 0, mulit\_file\_len\_sv = 0;

int SM\_take\_tb(int numChannels, char \*file\_name, int cdsF, int NDR\_flag, int images, int layers, int stripes, double frame\_interval, int factor, char \*cameraname, int file\_img\_w, int hbin, int BNC\_ratio,

int ExtTriggerFlag, int preTrigger\_frames, int darkFlag, int quadFlag, int rotateFlag, int bad\_pix\_index, int segmented)

{

int i, j, k;

int esc\_hit = 0;

int numFiles = 1;

int overrun[MAXCHANNELS], overruns[MAXCHANNELS] = { 0,0,0,0 }, timeouts[MAXCHANNELS] = { 0,0,0,0 };// , last\_timeouts = 0, recovering\_timeout = FALSE;

unsigned int \*dark\_image\_buffer = (unsigned int \*)NULL;

signed short int \*BNC\_data = (signed short int \*)NULL;

float64 \*NI\_data = (float64 \*)NULL;

int useFrameFormat = FALSE;

char frameFormat[\_MAX\_FNAME] = "";

char tsmfname[MAXCHANNELS][\_MAX\_FNAME] = { "","","","" };

int tsmhandle[MAXCHANNELS] = { NULL, NULL, NULL, NULL };

int writeToDisk = FALSE;

int debug = FALSE;

u\_char \*data\_ptr[MAXCHANNELS];

u\_char \*image\_ptr;

unsigned long long dataLength, fileDataLength;

unsigned long dataLeft;

int numExposures = 1;

char edt\_devname[128] = "";

char errorBuf[256] = "";

long triggerPos = 0;

int times[10][MAXCHANNELS][2];

char buf[200];

char disk\_tmp\_file[MAXCHANNELS][\_MAX\_FNAME] = { "", "", "","" };

int diskF\_handle[4];

int disk\_chunk\_size;

double stim\_train\_w, stim\_train\_intvl, stim\_train\_delay;

double stim\_train\_w2, stim\_train\_intvl2, stim\_train\_delay2;

double pacing\_train\_w2, pacing\_train\_intvl2, pacing\_train\_delay2;

double pacing\_train\_w3, pacing\_train\_intvl3, pacing\_train\_delay3;

int num\_stim1 = 0, num\_stim2 = 0, num\_stim3=0;

double ArtificialCell\_freq, ArtificialCell\_percent, ArtificialCell\_volt;

int ACell\_flag = 0;

HWND acq\_hwndPB\_disk;

// debug = TRUE;

loops = 1;

superFrame = 1;

strcpy(tsmfname[0], file\_name);

for (i = 0; i < MAXCHANNELS; ++i) {

if (strlen(tsmfname[i])) {

if ((tsmhandle[i] = \_open(tsmfname[i], O\_MODE, S\_MODE)) == -1) {

sprintf(errorBuf, "Cannot open %s\n", tsmfname[i]);

debug = TRUE;

goto cleanUp;

}

else {

\_close(tsmhandle[i]);

tsmhandle[i] = \_open(tsmfname[i], O\_MODE, S\_MODE);

\_lseek(tsmhandle[i], 0L, SEEK\_SET);

}

}

}

if (pcdFlag != 1) {

for (i = 0; i < numChannels; ++i) {

pdv\_setsize(pdv\_pt[i], config\_num\_col, config\_num\_row\*factor);

if (!pcdFlag) {

pdv\_flush\_fifo(pdv\_pt[i]);

pdv\_multibuf(pdv\_pt[i], 4); //has to be 4

}

}

}

initAcquire(numChannels, cdsF, layers, stripes, cameraname, quadFlag, rotateFlag, hbin, factor, 1);

if (NI\_flag) {

shutter\_add\_frames = (int)(shutter\_wait/frame\_interval);

NI\_add\_frames = 2 + shutter\_add\_frames;

char flagFname[MAX\_PATH];

FILE \*fp;

strcat(strcpy(flagFname, home\_dir), "\\SMSYSDAT\\pacing\_pattern.txt");

if (FOPEN\_S(fp, flagFname, "r")) {

if (fgets(buf, 255, fp))

sscanf(buf, "%d %lf %lf %lf%s", &num\_stim1, &stim\_train\_w, &stim\_train\_intvl, &stim\_train\_delay, buf);

if (fgets(buf, 255, fp))

sscanf(buf, "%d %lf %lf %lf%s", &num\_stim2, &pacing\_train\_w2, &pacing\_train\_intvl2, &pacing\_train\_delay2, buf);

if (fgets(buf, 255, fp))

sscanf(buf, "%d %lf %lf %lf%s", &num\_stim3, &pacing\_train\_w3, &pacing\_train\_intvl3, &pacing\_train\_delay3, buf);

fclose(fp);

}

else {

strcat(strcpy(flagFname, home\_dir), "\\SMSYSDAT\\stim\_pattern.txt");

if (FOPEN\_S(fp, flagFname, "r")) {

if (fgets(buf, 255, fp))

sscanf(buf, "%lf%s", &stim\_train\_w, buf);

if (fgets(buf, 255, fp))

sscanf(buf, "%lf%s", &stim\_train\_intvl, buf);

if (fgets(buf, 255, fp))

sscanf(buf, "%lf%s", &stim\_train\_delay, buf);

fclose(fp);

}

else

stim\_train\_w = 0.0;

}

strcat(strcpy(flagFname, home\_dir), "\\SMSYSDAT\\stim\_pattern2.txt");

if (FOPEN\_S(fp, flagFname, "r")) {

if (fgets(buf, 255, fp))

sscanf(buf, "%lf%s", &stim\_train\_w2, buf);

if (fgets(buf, 255, fp))

sscanf(buf, "%lf%s", &stim\_train\_intvl2, buf);

if (fgets(buf, 255, fp))

sscanf(buf, "%lf%s", &stim\_train\_delay2, buf);

fclose(fp);

}

else

stim\_train\_w2 = 0.0;

strcat(strcpy(flagFname, home\_dir), "\\SMSYSDAT\\artificial\_cell.txt");

if (FOPEN\_S(fp, flagFname, "r")) {

if (fgets(buf, 255, fp))

sscanf(buf, "%lf%s", &ArtificialCell\_freq, buf);

if (fgets(buf, 255, fp))

sscanf(buf, "%lf%s", &ArtificialCell\_percent, buf);

if (fgets(buf, 255, fp))

sscanf(buf, "%lf%s", &ArtificialCell\_volt, buf);

fclose(fp);

ACell\_flag = 1;

}

else

ACell\_flag = 0;

}

else {

NI\_add\_frames = 0;

shutter\_add\_frames = 0;

}

if (!NI\_flag && !AD\_flag) {

char BNCname[\_MAX\_PATH];

char \*p;

strcpy(BNCname, file\_name);

p = strstr(BNCname, ".tsm");

\*p = '\0';

strcat(BNCname, ".tbn");

DeleteFile(BNCname);

}

loops = (images + cdsFlag + NI\_add\_frames + 1) / (factor \* superFrame) + (((images + cdsFlag + NI\_add\_frames + 1) % (factor \* superFrame))>0);

if (!file\_img\_w)

file\_img\_w = file\_width;

int images\_header = images;

int fd\_file\_size\_cutoff = 5000;

if (FastDiskFlag && fd\_no\_deinterleave\_flag && images >= fd\_file\_size\_cutoff)

images\_header = 2000;

if (twoKFlag && ((file\_img\_w != file\_width) || (horizontalBin > 1)))

SM\_makeHeader(darkFlag? 1 : images\_header, numExposures, file\_height, min(file\_img\_w, (signed int)file\_width) / horizontalBin, cameraname, (double)frame\_interval / 1000.0);

else

SM\_makeHeader(darkFlag ? 1 : images\_header, numExposures, file\_height, file\_width, cameraname, (double)frame\_interval / 1000.0);

if (AD\_flag)

++loops;

dataLength = (unsigned long long)(loops + 1)\*(unsigned long long)image\_length;

fileDataLength = (unsigned long long)(images + 1)\*(unsigned long long)file\_length\*superFrame;

createAcqProgBar();

int step\_cnt = 0;

double step\_size = ((double)num\_steps) / loops;

char str[256];

sprintf(str, "Allocating memory");

SetWindowText(acq\_hwndPB, str);

if (!twoKFlag)

FastDiskFlag = 0;

if (image\_data != NULL)

\_aligned\_free(image\_data);

if (FastDiskFlag) {

for (i = 0; i < numChannels; i++) {

sprintf(disk\_tmp\_file[i], "%s:\\TB\_disk\_file\_ch%d.dat", fastD\_drive, i+1);

if ((diskF\_handle[i] = \_open(disk\_tmp\_file[i], O\_MODE, S\_MODE)) == -1) {

sprintf(errorBuf, "Cannot open %s\n", disk\_tmp\_file[i]);

debug = TRUE;

goto cleanUp;

}

}

if (fd\_no\_deinterleave\_flag && images >= fd\_file\_size\_cutoff) {

image\_data = (signed short \*)\_aligned\_malloc(image\_length\*(images\_header / factor + 1), 2);

}

else {

disk\_chunk\_size = factor \* 50;

image\_data = (signed short \*)\_aligned\_malloc(image\_length\*(disk\_chunk\_size / factor + 1), 2);

}

}

else if ((image\_data = (signed short \*)\_aligned\_malloc(dataLength, 2)) == NULL) {

writeToDisk = TRUE;

strcpy(errorBuf, "Not enough memory for Data - please reduce the number of frames to be acquired");

}

if (AD\_flag && !writeToDisk) {

if ((BNC\_data = (signed short \*)\_aligned\_malloc(loops\*image\_width\*factor \* 2 \* sizeof(BNC\_data[0]), 2)) == NULL) {

writeToDisk = TRUE;

strcpy(errorBuf, "Not enough memory for BNC\_data");

}

}

if (writeToDisk) {

debug = TRUE;

goto cleanUp;

}

sprintf(str, "Waiting for trigger");

SetWindowText(acq\_hwndPB, str);

dataLeft = sizeof(FitsHeader) - (fileDataLength) % sizeof(FitsHeader);

memset(&FitsEnder, 0, min(sizeof(FitsEnder),dataLeft));

if (twoKFlag)

for (i = 0; i < numChannels; ++i)

data\_ptr[i] = (u\_char \*)image\_data + i\*frame\_length/factor;

else

data\_ptr[0] = (u\_char \*)image\_data;

for (i = 0; i < numFiles; ++i)

\_write(tsmhandle[i], (char \*)FitsHeader, sizeof(FitsHeader));

for (i = 0; i < numChannels; ++i)

pdv\_start\_images(pdv\_pt[i], loops + 1);

if (ExtTriggerFlag == 1) {

sprintf(str, "Waiting for trigger");

SetWindowText(acq\_hwndPB, str);

}

if (ExtTriggerFlag && AD\_flag) {

long numAcquire = 0, numTriggeredAcq = 0, frame\_pos = 0, triggerPix\_inc;

signed short \*triggerPixel;

int triggered = FALSE;

if (strstr(cameraname, "DW"))

triggerPix\_inc = frame\_length /sizeof(signed short) - frame\_width + 6;

else

triggerPix\_inc = 6;

long timeout\_frame = min(200000, (long)(300000/frame\_interval));

int trigger\_val;

trigger\_val = 6500;

while (numAcquire < timeout\_frame && numTriggeredAcq < loops -preTrigger\_frames) {

image\_ptr = pdv\_wait\_image\_raw(pdv\_pt[0]);

if ((overrun[0] = (edt\_reg\_read(pdv\_pt[0], PDV\_STAT) & PDV\_OVERRUN)))

++overruns[0];

memcpy(data\_ptr[0], image\_ptr, frame\_length);

if (!triggered) {

pdv\_start\_image(pdv\_pt[0]);

triggerPixel = (signed short \*)data\_ptr[0];

if (\*(triggerPixel+triggerPix\_inc) > trigger\_val) {

triggered = TRUE;

numTriggeredAcq = 0;

}

}

else {

numTriggeredAcq++;

sprintf(str, "Acquired %d of %d images", numTriggeredAcq, images);

SetWindowText(acq\_hwndPB, str);

while ((int)(numTriggeredAcq\*step\_size + 0.5) >= step\_cnt && step\_cnt < num\_steps) {

SendMessage(acq\_hwndPB, PBM\_STEPIT, 0, 0);

step\_cnt++;

}

}

numAcquire++;

frame\_pos++;

if (frame\_pos >= loops) {

data\_ptr[0] = (u\_char \*)image\_data;

frame\_pos = 0;

}

else

data\_ptr[0] += image\_length;

}

data\_ptr[0] = (u\_char \*)image\_data;

triggerPos = frame\_pos;

}

else {

/\*\*\*\*\*for NI acquisition\*\*\*/

#ifdef \_USE\_NI

int error = 0;

TaskHandle taskHandle\_in = 0, taskHandle\_out = 0, taskHandle\_clk = 0, taskHandle\_AO = 0;

uInt32 \*output\_data = 0;

float64 \*AO\_data = NULL;

char errBuff[2048] = { '\0' };

int i, delay, output\_sample, resting\_v, trigger\_v, num\_ni\_out = 2;

float64 output\_rate, sample\_chan\_rate;

float64 \*NI\_pt = NULL;

long total\_read = 0, total\_written = 0, BNC\_num\_frames;

if (NI\_flag) {

delay = 0; // max(1, (int)(4.0 / frame\_interval));

// config outputs

sample\_chan\_rate = BNC\_ratio\*1005.0 / frame\_interval;

output\_rate = BNC\_ratio\*1000.0 / frame\_interval;

output\_sample = (images + NI\_add\_frames + 1)\*BNC\_ratio + delay + 2;

BNC\_num\_frames = output\_sample - delay - 2;

output\_data = (uInt32 \*)\_aligned\_malloc(output\_sample \* sizeof(output\_data[0]), 4);

NI\_data = (float64 \*)\_aligned\_malloc(BNC\_num\_frames\*numBNC\_chan\*BNC\_ratio \* sizeof(NI\_data[0]), 2);

NI\_pt = NI\_data;

if (NI\_flag == 1) {

resting\_v = 1;

trigger\_v = 0;

}

else {

resting\_v = 0;

trigger\_v = 1;

}

for (i = 0; i < delay; ++i)

output\_data[i] = resting\_v + 2;

if (BNC\_ratio > 1) {

int m;

for (; i < output\_sample / BNC\_ratio; ++i) {

for (m = 0; m < BNC\_ratio / 2; m++)

output\_data[i\*BNC\_ratio + m] = trigger\_v + 2;

for (; m < BNC\_ratio; m++)

output\_data[i\*BNC\_ratio + m] = resting\_v + 2;

}

for (m = (output\_sample / BNC\_ratio)\*BNC\_ratio; m < output\_sample; m++)

output\_data[m] = trigger\_v + 2;

output\_data[output\_sample - 1] = trigger\_v + 2;

}

else {

for (; i < output\_sample; ++i)

output\_data[i] = trigger\_v + 2;

}

if ((stim\_train\_w < 0.0000001) || (stim\_train\_w < frame\_interval/BNC\_ratio)) {

if (!MNC\_flag) {

int stim\_delay\_frame = (int)(BNC\_ratio \* stim\_delay / frame\_interval);

for (i = 0; i < BNC\_ratio \* 2; i++)

output\_data[delay + NI\_add\_frames\*BNC\_ratio + stim\_delay\_frame + i] += 4;

//add TTL output to P0.3 at the end of acqusition

for (i = output\_sample - BNC\_ratio \* 2; i < output\_sample; i++)

output\_data[i] += 8;

}

else {

int AO\_frames = images + NI\_add\_frames + 1;

if (ratio\_type) {

for (i = 0; i < frame\_per\_ratio\*BNC\_ratio; ++i)

output\_data[i] += 8;

for (; i < (AO\_frames- frame\_per\_ratio)\*BNC\_ratio; ++i)

output\_data[i] += 4;

for (; i < output\_sample; ++i)

output\_data[i] += 8;

}

else {

for (i = 0; i < AO\_frames / frame\_per\_ratio; ++i) {

for (int m = 0; m < frame\_per\_ratio\*BNC\_ratio; ++m) {

if (i % 2)

output\_data[i\*frame\_per\_ratio + m] += 8;

else

output\_data[i\*frame\_per\_ratio + m] += 4;

}

}

// for (i = (AO\_frames / frame\_per\_ratio - 1)\*frame\_per\_ratio; i < AO\_frames\*BNC\_ratio; ++i)

// output\_data[i] += 8;

}

}

}

else {

int stim\_train\_w\_frame = (int)(BNC\_ratio \* stim\_train\_w / frame\_interval);

int stim\_train\_intvl\_frame = (int)(BNC\_ratio \* stim\_train\_intvl / frame\_interval);

int stim\_train\_delay\_frame = (int)(BNC\_ratio \* stim\_train\_delay / frame\_interval);

int train\_stim\_pos = delay + NI\_add\_frames\* BNC\_ratio + stim\_train\_delay\_frame;

if (stim\_train\_intvl\_frame > 0) {

if (num\_stim1) {

for (j = 0; j < num\_stim1; j++) {

for (i = 0; i < stim\_train\_w\_frame; i++)

output\_data[train\_stim\_pos + i] += 4;

train\_stim\_pos += stim\_train\_intvl\_frame;

if (train\_stim\_pos + stim\_train\_w\_frame >= output\_sample)

break;

}

if (num\_stim2) {

stim\_train\_w\_frame = (int)(BNC\_ratio \* pacing\_train\_w2 / frame\_interval);

stim\_train\_intvl\_frame = (int)(BNC\_ratio \* pacing\_train\_intvl2 / frame\_interval);

stim\_train\_delay\_frame = (int)(BNC\_ratio \* pacing\_train\_delay2 / frame\_interval);

train\_stim\_pos = delay + NI\_add\_frames\* BNC\_ratio + stim\_train\_delay\_frame;

for (j = 0; j < num\_stim2; j++) {

for (i = 0; i < stim\_train\_w\_frame; i++)

if (output\_data[train\_stim\_pos + i] < 4)

output\_data[train\_stim\_pos + i] += 4;

train\_stim\_pos += stim\_train\_intvl\_frame;

if (train\_stim\_pos + stim\_train\_w\_frame >= output\_sample)

break;

}

}

if (num\_stim3) {

stim\_train\_w\_frame = (int)(BNC\_ratio \* pacing\_train\_w3 / frame\_interval);

stim\_train\_intvl\_frame = (int)(BNC\_ratio \* pacing\_train\_intvl3 / frame\_interval);

stim\_train\_delay\_frame = (int)(BNC\_ratio \* pacing\_train\_delay3 / frame\_interval);

train\_stim\_pos = delay + NI\_add\_frames\* BNC\_ratio + stim\_train\_delay\_frame;

for (j = 0; j < num\_stim3; j++) {

for (i = 0; i < stim\_train\_w\_frame; i++)

if (output\_data[train\_stim\_pos + i] < 4)

output\_data[train\_stim\_pos + i] += 4;

train\_stim\_pos += stim\_train\_intvl\_frame;

if (train\_stim\_pos + stim\_train\_w\_frame >= output\_sample)

break;

}

}

}

else {

while (train\_stim\_pos + stim\_train\_w\_frame < output\_sample) {

for (i = 0; i < stim\_train\_w\_frame; i++)

output\_data[train\_stim\_pos + i] += 4;

train\_stim\_pos += stim\_train\_intvl\_frame;

}

}

}

if ((stim\_train\_w2 >= frame\_interval / BNC\_ratio) && (stim\_train\_w2 > 0.0000001)) {

stim\_train\_w\_frame = (int)(BNC\_ratio \* stim\_train\_w2 / frame\_interval);

stim\_train\_intvl\_frame = (int)(BNC\_ratio \* stim\_train\_intvl2 / frame\_interval);

stim\_train\_delay\_frame = (int)(BNC\_ratio \* stim\_train\_delay2 / frame\_interval);

train\_stim\_pos = delay + NI\_add\_frames\* BNC\_ratio + stim\_train\_delay\_frame;

if (stim\_train\_intvl\_frame > 0) {

while (train\_stim\_pos + stim\_train\_w\_frame < output\_sample) {

for (i = 0; i < stim\_train\_w\_frame; i++)

output\_data[train\_stim\_pos + i] += 8;

train\_stim\_pos += stim\_train\_intvl\_frame;

}

}

}

}

// config clock channel M series don't have internal clock for output.

DAQmxErrChk(DAQmxCreateTask("", &taskHandle\_clk));

DAQmxErrChk(DAQmxCreateCOPulseChanTime(taskHandle\_clk, "Dev1/ctr0", "", DAQmx\_Val\_Seconds, DAQmx\_Val\_Low, 0.00, 0.50 / output\_rate, 0.50 / output\_rate));

DAQmxErrChk(DAQmxCfgImplicitTiming(taskHandle\_clk, DAQmx\_Val\_ContSamps, output\_sample));

DAQmxErrChk(DAQmxCreateTask("", &taskHandle\_out));

DAQmxErrChk(DAQmxCreateDOChan(taskHandle\_out, "Dev1/port0/line0:2", "", DAQmx\_Val\_ChanForAllLines));

DAQmxErrChk(DAQmxCfgSampClkTiming(taskHandle\_out, "/Dev1/PFI12", output\_rate, DAQmx\_Val\_Rising, DAQmx\_Val\_FiniteSamps, output\_sample)); //P

if (ExtTriggerFlag)

DAQmxErrChk(DAQmxCfgDigEdgeStartTrig(taskHandle\_clk, "/Dev1/PFI1", DAQmx\_Val\_Rising));

if (ACell\_flag) {

// config AO for artificial cell

int AO\_frames = images + NI\_add\_frames + 1;

int one\_cycle = (int)(1000 / (ArtificialCell\_freq\*frame\_interval));

AO\_data = (float64 \*)\_aligned\_malloc(AO\_frames \* sizeof(AO\_data[0]), 2);

for (i = 0; i < AO\_frames; i++)

AO\_data[i] = ArtificialCell\_volt;

for (int j = 0; j < AO\_frames / one\_cycle; ++j) {

for (i = (int)(one\_cycle\*0.9); i < one\_cycle; ++i)

AO\_data[j\* one\_cycle + i] = ArtificialCell\_volt\*(1+ ArtificialCell\_percent/100);

}

DAQmxErrChk(DAQmxCreateTask("", &taskHandle\_AO));

DAQmxErrChk(DAQmxCreateAOVoltageChan(taskHandle\_AO, "Dev1/ao0", "", 0.0, 5.0, DAQmx\_Val\_Volts, NULL)); // for 6341 - 2 analog lines

DAQmxErrChk(DAQmxCfgSampClkTiming(taskHandle\_AO, "/Dev1/PFI0", sample\_chan\_rate, DAQmx\_Val\_Rising, DAQmx\_Val\_FiniteSamps, AO\_frames));

}

else if (MNC\_flag) {

// config AO for monochromator

int AO\_frames = images + NI\_add\_frames + 1;

int wavelen\_jump;

if (MNC\_flag == 2) {

AO\_data = (float64 \*)\_aligned\_malloc(AO\_frames \* sizeof(AO\_data[0]) \* 2, 2);

wavelen\_jump = 1;

}

else {

AO\_data = (float64 \*)\_aligned\_malloc(AO\_frames \* sizeof(AO\_data[0]) \* 3, 2);

wavelen\_jump = 2;

}

if (ratio\_type) {

for (i = 0; i < frame\_per\_ratio; ++i) {

AO\_data[i] = MNC\_in1\_vol;

AO\_data[i + AO\_frames] = MNC\_out1\_vol;

AO\_data[i + AO\_frames \* wavelen\_jump] = MNC\_w1\_vol;

}

for (; i < AO\_frames - frame\_per\_ratio; ++i) {

AO\_data[i] = MNC\_in2\_vol;

AO\_data[i + AO\_frames] = MNC\_out2\_vol;

AO\_data[i + AO\_frames \* wavelen\_jump] = MNC\_w2\_vol;

}

for (; i < AO\_frames; ++i) {

AO\_data[i] = MNC\_in1\_vol;

AO\_data[i + AO\_frames] = MNC\_out1\_vol;

AO\_data[i + AO\_frames \* wavelen\_jump] = MNC\_w1\_vol;

}

}

else {

for (i = 0; i < AO\_frames / frame\_per\_ratio; ++i) {

for (int m = 0; m < frame\_per\_ratio; ++m) {

if (i % 2) {

AO\_data[i\*frame\_per\_ratio + m] = MNC\_in2\_vol;

AO\_data[i\*frame\_per\_ratio + m + AO\_frames] = MNC\_out2\_vol;

AO\_data[i\*frame\_per\_ratio + m + AO\_frames \* wavelen\_jump] = MNC\_w2\_vol;

}

else {

AO\_data[i\*frame\_per\_ratio + m] = MNC\_in1\_vol;

AO\_data[i\*frame\_per\_ratio + m + AO\_frames] = MNC\_out1\_vol;

AO\_data[i\*frame\_per\_ratio + m + AO\_frames \* wavelen\_jump] = MNC\_w1\_vol;

}

}

}

for (i = (AO\_frames / frame\_per\_ratio - 1)\*frame\_per\_ratio; i < AO\_frames; ++i) {

AO\_data[i] = MNC\_in2\_vol;

AO\_data[i + AO\_frames] = MNC\_out2\_vol;

AO\_data[i + AO\_frames \* wavelen\_jump] = MNC\_w2\_vol;

}

}

DAQmxErrChk(DAQmxCreateTask("", &taskHandle\_AO));

if (MNC\_flag == 2)

DAQmxErrChk(DAQmxCreateAOVoltageChan(taskHandle\_AO, "Dev1/ao0:1", "", 0.0, 5.0, DAQmx\_Val\_Volts, NULL)); // for 6341 - 2 analog lines

else

DAQmxErrChk(DAQmxCreateAOVoltageChan(taskHandle\_AO, "Dev1/ao0:2", "", 0.0, 5.0, DAQmx\_Val\_Volts, NULL));

DAQmxErrChk(DAQmxCfgSampClkTiming(taskHandle\_AO, "/Dev1/PFI0", sample\_chan\_rate, DAQmx\_Val\_Rising, DAQmx\_Val\_FiniteSamps, AO\_frames));

}

// config inputs and trigger

DAQmxErrChk(DAQmxCreateTask("", &taskHandle\_in));

if (numBNC\_chan == 8)

DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle\_in, "Dev1/ai0:7", "", DAQmx\_Val\_RSE, -10.0, 10.0, DAQmx\_Val\_Volts, NULL));

else

DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle\_in, "Dev1/ai0:3", "", DAQmx\_Val\_RSE, -10.0, 10.0, DAQmx\_Val\_Volts, NULL));

if (BNC\_ratio > 1) {

DAQmxErrChk(DAQmxCfgSampClkTiming(taskHandle\_in, "/Dev1/PFI12", sample\_chan\_rate, DAQmx\_Val\_Falling, DAQmx\_Val\_FiniteSamps, BNC\_num\_frames)); //frame-by-frame clock trigger

DAQmxErrChk(DAQmxCfgDigEdgeStartTrig(taskHandle\_in, "/Dev1/PFI0", DAQmx\_Val\_Rising));

}

else

DAQmxErrChk(DAQmxCfgSampClkTiming(taskHandle\_in, "/Dev1/PFI0", sample\_chan\_rate, DAQmx\_Val\_Rising, DAQmx\_Val\_FiniteSamps, BNC\_num\_frames)); //frame-by-frame clock trigger

if (NI\_flag != 1)

DAQmxErrChk(DAQmxCfgDigEdgeStartTrig(taskHandle\_in, "/Dev1/PFI2", DAQmx\_Val\_Rising));

DAQmxErrChk(DAQmxRegisterDoneEvent(taskHandle\_clk, 0, DoneCallback, NULL));

DAQmxErrChk(DAQmxWriteDigitalU32(taskHandle\_out, output\_sample, 0, 10.0, DAQmx\_Val\_GroupByChannel, output\_data, &total\_written, NULL));

if (MNC\_flag || ACell\_flag)

DAQmxErrChk(DAQmxWriteAnalogF64(taskHandle\_AO, images + 1, 0, 10.0, DAQmx\_Val\_GroupByChannel, AO\_data, NULL, NULL));

DAQmxErrChk(DAQmxStartTask(taskHandle\_in));

if (MNC\_flag || ACell\_flag)

DAQmxErrChk(DAQmxStartTask(taskHandle\_AO));

DAQmxErrChk(DAQmxStartTask(taskHandle\_out));

DAQmxErrChk(DAQmxStartTask(taskHandle\_clk));

if (debug)

fprintf(stdout, "\n\noutput\_sample: %d, total\_written :%d, %lf, %lf, %d\n\n", output\_sample, total\_written, sample\_chan\_rate, output\_rate, BNC\_ratio);

}

#endif

/\*\*\*\*End of NI\*\*\*\*/

if (SFC\_flag)

startSFC\_scan(-1);

// twoK\_start\_missed = 0;

if (pcdFlag) {

int bufsize = image\_length;

int numbufs = 4;

PdvDev \*pdv\_p\_sv = pdv\_p;

// if (pcdFlag == 2)

pdv\_p = pcd\_p;

edt\_reg\_write(pdv\_p, SSD16\_CHEN, 0x0);

edt\_flush\_channel(pdv\_p, 0);

if (edt\_configure\_ring\_buffers(pdv\_p, bufsize, numbufs, EDT\_READ, NULL) != -1)

{

edt\_start\_buffers(pdv\_p, loops);

edt\_reg\_write(pdv\_p, SSD16\_CHEN, 0x0001);

for (j = 0; (j < loops || loops == 0); j++)

{

image\_ptr = edt\_wait\_for\_buffers(pdv\_p, 1);

if (FastDiskFlag)

\_write(diskF\_handle[0], (void \*)image\_ptr, frame\_length);

else {

memcpy(data\_ptr[0], image\_ptr, frame\_length);

data\_ptr[0] += image\_length;

}

sprintf(str, "Acquired %d of %d images", j\*factor, images);

SetWindowText(acq\_hwndPB, str);

while ((int)(j\*step\_size + 0.5) >= step\_cnt && step\_cnt < num\_steps) {

SendMessage(acq\_hwndPB, PBM\_STEPIT, 0, 0);

step\_cnt++;

}

}

}

else

MessageBox(main\_window\_handle, "edt\_configure\_ring\_buffers failed", "message", MB\_OK);

pdv\_p = pdv\_p\_sv;

}

else if (numChannels > 1 && omp\_get\_num\_procs() >= numChannels) {

for (int m = 0; m < 10; m++) {

for (int n = 0; n < numChannels; n++) {

image\_ptr = pdv\_wait\_image\_timed\_raw(pdv\_pt[n], (unsigned int \*)&times[0][n][0], 1);

if (FastDiskFlag)

\_write(diskF\_handle[n], (void \*)image\_ptr, frame\_length);

else {

for (i = 0; i < factor; ++i) {

memcpy(data\_ptr[n], image\_ptr, frame\_length / factor);

image\_ptr += frame\_length / factor;

data\_ptr[n] += image\_length / factor;

}

}

}

break;

}

omp\_set\_num\_threads(numChannels);

for (j = 1; j < loops; j++) {

#pragma omp parallel private(image\_ptr)

{

int thread\_id, i;

thread\_id = omp\_get\_thread\_num();

// image\_ptr = pdv\_wait\_image\_timed\_raw(pdv\_pt[thread\_id], (unsigned int \*)&times[0][thread\_id][0], 1);

image\_ptr = pdv\_wait\_image\_raw(pdv\_pt[thread\_id]);

if ((overrun[thread\_id] = (edt\_reg\_read(pdv\_pt[thread\_id], PDV\_STAT) & PDV\_OVERRUN)))

++overruns[thread\_id];

if (FastDiskFlag)

\_write(diskF\_handle[thread\_id], (void \*)image\_ptr, frame\_length);

else {

for (i = 0; i < factor; ++i) {

memcpy(data\_ptr[thread\_id], image\_ptr, frame\_length / factor);

image\_ptr += frame\_length / factor;

data\_ptr[thread\_id] += image\_length / factor;

}

}

}

/\*\*\*\*\*for NI acquisition\*\*\*/

#ifdef \_USE\_NI

if (NI\_flag && !darkFlag) {

long read;

if (j == loops - 1)

DAQmxReadAnalogF64(taskHandle\_in, factor\*BNC\_ratio\*(j + 1) - total\_read, 5.0, DAQmx\_Val\_GroupByScanNumber, NI\_pt, (factor\*BNC\_ratio\*(j + 1) - total\_read + 1)\*numBNC\_chan, &read, NULL);

else

DAQmxReadAnalogF64(taskHandle\_in, factor\*BNC\_ratio\*(j + 1) - total\_read, 0.0, DAQmx\_Val\_GroupByScanNumber, NI\_pt, (factor\*BNC\_ratio\*(j + 1) - total\_read)\*numBNC\_chan, &read, NULL);

NI\_pt += read\*numBNC\_chan;

total\_read += read;

}

#endif

/\*\*\*\*End of NI\*\*\*\*/

sprintf(str, "Acquired %d of %d images", j\*factor, images);

SetWindowText(acq\_hwndPB, str);

while ((int)(j\*step\_size + 0.5) >= step\_cnt && step\_cnt < num\_steps) {

SendMessage(acq\_hwndPB, PBM\_STEPIT, 0, 0);

step\_cnt++;

}

if (KEY\_DOWN(VK\_ESCAPE)) {

esc\_hit = 1;

goto exitAcq;

}

}

}

else {

for (j = 0; j < loops; j++) {

for (i = 0; i < numChannels; ++i) {

image\_ptr = pdv\_wait\_image\_raw(pdv\_pt[i]);

if ((overrun[i] = (edt\_reg\_read(pdv\_pt[i], PDV\_STAT) & PDV\_OVERRUN)))

++overruns[i];

memcpy(data\_ptr[i], image\_ptr, frame\_length);

data\_ptr[i] += image\_length;

}

if (Falcon\_flag) {

unsigned \_\_int64 \*long\_pt = (unsigned \_\_int64 \*)(data\_ptr[0] - image\_length);

unsigned short int \*short\_pt0, \*short\_pt1, \*short\_pt2, \*short\_pt3;

short\_pt0 = (unsigned short int \*)long\_pt;

short\_pt1 = short\_pt0 + 1;

short\_pt2 = short\_pt1 + 1;

short\_pt3 = short\_pt2 + 1;

for (i = 0; i < (signed)frame\_length >> 2; i++) {

\*long\_pt++ = (bitLut[\*short\_pt0] | bitLut[(\*short\_pt1) | 0x10000L] | bitLut[(\*short\_pt2) | 0x20000L] | bitLut[(\*short\_pt3) | 0x30000L]) & 0x3fff3fff3fff3fffL;

short\_pt0 += 4;

short\_pt1 += 4;

short\_pt2 += 4;

short\_pt3 += 4;

}

}

/\*\*\*\*\*for NI acquisition\*\*\*/

#ifdef \_USE\_NI

if (NI\_flag && !darkFlag) {

long read;

if (j == loops - 1)

DAQmxReadAnalogF64(taskHandle\_in, factor\*BNC\_ratio\*(j + 1) - total\_read, 5.0, DAQmx\_Val\_GroupByScanNumber, NI\_pt, (factor\*BNC\_ratio\*(j + 1) - total\_read + 1)\*numBNC\_chan, &read, NULL);

else

DAQmxReadAnalogF64(taskHandle\_in, factor\*BNC\_ratio\*(j + 1) - total\_read, 0.0, DAQmx\_Val\_GroupByScanNumber, NI\_pt, (factor\*BNC\_ratio\*(j + 1) - total\_read)\*numBNC\_chan, &read, NULL);

NI\_pt += read\*numBNC\_chan;

total\_read += read;

}

#endif

/\*\*\*\*End of NI\*\*\*\*/

sprintf(str, "Acquired %d of %d images", j\*factor, images);

SetWindowText(acq\_hwndPB, str);

while ((int)(j\*step\_size + 0.5) >= step\_cnt && step\_cnt < num\_steps) {

SendMessage(acq\_hwndPB, PBM\_STEPIT, 0, 0);

step\_cnt++;

}

}

}

exitAcq:

GetSystemTime(&acq\_end\_sysTime);

for (i = 0; i < numChannels; ++i)

timeouts[i] = pdv\_timeouts(pdv\_pt[i]);

/\*\*\*\*\*for NI acquisition\*\*\*/

#ifdef \_USE\_NI

Error:

if (NI\_flag && !darkFlag) {

if (DAQmxFailed(error)) {

debug = TRUE;

DAQmxGetExtendedErrorInfo(errBuff, 2048);

}

if (taskHandle\_clk) {

DAQmxStopTask(taskHandle\_clk);

DAQmxClearTask(taskHandle\_clk);

}

if (taskHandle\_AO) {

DAQmxStopTask(taskHandle\_AO);

DAQmxClearTask(taskHandle\_AO);

}

if (taskHandle\_out != 0) {

DAQmxStopTask(taskHandle\_out);

DAQmxClearTask(taskHandle\_out);

}

if (taskHandle\_in != 0) {

DAQmxStopTask(taskHandle\_in);

DAQmxClearTask(taskHandle\_in);

}

if (DAQmxFailed(error)) {

char errStr[256];

sprintf(errStr, "National Instruments Device Error: %s", errBuff);

MessageBox(main\_window\_handle, errStr, "message", MB\_OK);

}

if (output\_data)

\_aligned\_free(output\_data);

if (AO\_data)

\_aligned\_free(AO\_data);

NI\_add\_frames -= shutter\_add\_frames;

saveNI\_data(images, tsmfname[0], numBNC\_chan, BNC\_ratio, NI\_data, delay, twoKFlag);

}

#endif

/\*\*\*\*End of NI\*\*\*\*/

}

SM\_stopAD();

if (esc\_hit)

goto cleanUp;

if (AD\_flag) {

if (factor > 1) {

frame\_height /= factor;

factor = 1;

loops = images + 1;

}

stripAD\_data((unsigned short \*)image\_data, BNC\_data, cameraname, file\_height, frame\_width, frame\_height, loops, cdsFlag);

--loops;

saveBNC\_data(loops, tsmfname[0], numBNC\_chan, BNC\_ratio, BNC\_data, cameraname, frame\_width, stripes);

image\_height = file\_height;

image\_length = image\_width\*image\_height\*2;

}

if (factor > 1) {

image\_height = image\_height / factor;

image\_length = image\_width\*image\_height \* 2;

factor = 1;

loops = (images + cdsFlag + NI\_add\_frames) / superFrame + (((images + cdsFlag + NI\_add\_frames) % superFrame)>0);

// if (twoK\_start\_missed)

// loops++;

}

if (Bit16\_flag) {

unsigned short \*img\_ptr = (unsigned short \*)image\_data;

for (i = 0; i < (int)(image\_length\*loops>>1); i++)

\*img\_ptr++ = (\*img\_ptr >> 2);

}

if (FastDiskFlag) {

int stt\_byte;

int read\_len;

unsigned int file\_length\_sv;

acq\_hwndPB\_disk = acq\_hwndPB;

acq\_hwndPB = NULL;

SendMessage(acq\_hwndPB\_disk, PBM\_SETPOS, (WPARAM)0, 0);

step\_cnt = 0;

step\_size = ((double)num\_steps) / loops;

long this\_img;

signed short \*last\_frame = (signed short \*)\_aligned\_malloc(image\_length, 2);

if (NI\_flag && shutter\_add\_frames)

stt\_byte = shutter\_add\_frames \* image\_length/numChannels;

else

stt\_byte = 0;

disk\_chunk\_size = min(disk\_chunk\_size, loops);

data\_ptr[0] = (u\_char \*)image\_data;

read\_len = image\_length / numChannels;

for (i = 0; i < numChannels; i++) {

\_close(diskF\_handle[i]);

diskF\_handle[i] = \_open(disk\_tmp\_file[i], O\_BINARY | O\_RDONLY);

\_lseeki64(diskF\_handle[i], stt\_byte, SEEK\_SET);

// if (cdsFlag) {

\_read(diskF\_handle[i], (u\_char \*)data\_ptr[0], read\_len);

data\_ptr[0] += image\_length / numChannels;

// }

}

this\_img = 1;

file\_length\_sv = file\_length;

if (fd\_no\_deinterleave\_flag && images >= fd\_file\_size\_cutoff) {

step\_size = ((double)num\_steps) / images\_header;

for (k = 0; k < images\_header / 2; k++) {

for (i = 0; i < numChannels; i++) {

\_read(diskF\_handle[i], (u\_char \*)data\_ptr[0], read\_len);

data\_ptr[0] += image\_length / numChannels;

}

}

for (i = 0; i < numChannels; i++)

\_lseeki64(diskF\_handle[i], stt\_byte + read\_len\*(images - images\_header / 2), SEEK\_SET);

for (k = 0; k < images\_header / 2 + 1; k++) {

for (i = 0; i < numChannels; i++) {

\_read(diskF\_handle[i], (u\_char \*)data\_ptr[0], read\_len);

data\_ptr[0] += image\_length / numChannels;

}

}

if (cdsFlag)

subtractCDS((unsigned short \*)image\_data, images\_header + 1, image\_width, image\_height, image\_length, factor, quadFlag, twoKFlag, numChannels);

data\_ptr[0] = (u\_char \*)image\_data;

for (k = 0; k < images\_header; k++) {

file\_length = file\_length\_sv;

frame\_deInterleave(superFrame\*file\_length >> 1, lut, (unsigned short \*)data\_ptr[0], image\_buffer);

frameRepair(image\_buffer, bad\_pix\_index, twoKFlag, image\_width, file\_width, file\_height, horizontalBin, superFrame, file\_img\_w, binFlag, segmented);

file\_length = file\_height \* min(file\_img\_w, (signed int)file\_width) \* 2 / horizontalBin;

if (ExtTriggerFlag && AD\_flag && triggerPos) {

if (!k)

\_lseek(tsmhandle[0], 2880L + (loops - triggerPos - 1)\*file\_length, SEEK\_SET);

else if (k == triggerPos)

\_lseek(tsmhandle[0], 2880L, SEEK\_SET);

}

\_write(tsmhandle[0], (void \*)image\_buffer, file\_length);

data\_ptr[0] += image\_length;

sprintf(str, "Deinterleaving %d of %d images", this\_img, images\_header);

SetWindowText(acq\_hwndPB\_disk, str);

while ((int)(this\_img\*step\_size + 0.5) >= step\_cnt && step\_cnt < num\_steps) {

SendMessage(acq\_hwndPB\_disk, PBM\_STEPIT, 0, 0);

step\_cnt++;

}

this\_img++;

}

memset(data\_ptr[0], 0, file\_length);

dataLeft = sizeof(FitsHeader) - ((images\_header+1)\*file\_length) % sizeof(FitsHeader);

\_lseeki64(tsmhandle[0], 2880L + (unsigned long long)images\_header\*(unsigned long long)file\_length, SEEK\_SET);

\_write(tsmhandle[0], (void \*)data\_ptr[0], file\_length);

\_write(tsmhandle[0], (char \*)FitsEnder, dataLeft);

char new\_diskF\_name[\_MAX\_PATH], new\_diskF\_name2[\_MAX\_PATH];

char \*newP;

char mssg[256];

for (i = 0; i < numChannels; i++) {

\_close(diskF\_handle[i]);

strcpy(new\_diskF\_name, file\_name);

while (newP = strstr(new\_diskF\_name, "\\"))

strcpy(new\_diskF\_name, newP + 1);

if (newP = strstr(new\_diskF\_name, ".tsm")) {

\*newP = '\0';

strcpy(new\_diskF\_name2, new\_diskF\_name);

sprintf(new\_diskF\_name, "%s:\\%s\_disk\_file\_ch%d.dat", fastD\_drive, new\_diskF\_name2, i+1);

if ((diskF\_handle[i] = \_open(new\_diskF\_name, O\_BINARY | O\_RDONLY)) != -1) {

\_close(diskF\_handle[i]);

sprintf(mssg, "%s already exists. Yes to overwrite it, No to keep the name %s, which you need to delete or rename before next acqusition", new\_diskF\_name, disk\_tmp\_file[i]);

if (MessageBox(main\_window\_handle,

(LPCTSTR) mssg,

(LPCTSTR) "Quit Turbo-SM",

MB\_YESNO |

MB\_ICONQUESTION |

MB\_SYSTEMMODAL) == IDYES) {

DeleteFile(new\_diskF\_name);

rename(disk\_tmp\_file[i], new\_diskF\_name);

}

else

strcpy(new\_diskF\_name, disk\_tmp\_file[i]);

}

else

rename(disk\_tmp\_file[i], new\_diskF\_name);

if (!i) {

if ((diskF\_handle[i] = \_open(new\_diskF\_name, O\_APPEND | O\_BINARY | O\_SEQUENTIAL | O\_RDWR, S\_MODE)) != -1) {

\_lseeki64(diskF\_handle[i], 0, SEEK\_END);

\_write(diskF\_handle[i], &images, sizeof(int));

\_write(diskF\_handle[i], &image\_width, sizeof(unsigned int));

\_write(diskF\_handle[i], &image\_height, sizeof(unsigned int));

\_write(diskF\_handle[i], &image\_length, sizeof(unsigned int));

\_write(diskF\_handle[i], &file\_width, sizeof(unsigned int));

\_write(diskF\_handle[i], &file\_height, sizeof(unsigned int));

\_write(diskF\_handle[i], &image\_stripes, sizeof(int));

\_write(diskF\_handle[i], &image\_layers, sizeof(int));

\_write(diskF\_handle[i], &stt\_byte, sizeof(int));

\_write(diskF\_handle[i], &cdsFlag, sizeof(int));

\_write(diskF\_handle[i], &numChannels, sizeof(int));

\_write(diskF\_handle[i], &horizontalBin, sizeof(int));

\_write(diskF\_handle[i], &file\_img\_w, sizeof(int));

\_write(diskF\_handle[i], &binFlag, sizeof(int));

\_write(diskF\_handle[i], &segmented, sizeof(int));

\_close(diskF\_handle[i]);

}

}

}

}

}

else {

for (j = 0; j < loops / disk\_chunk\_size; ++j) {

for (k = 0; k < disk\_chunk\_size; k++) {

for (i = 0; i < numChannels; i++) {

\_read(diskF\_handle[i], (u\_char \*)data\_ptr[0], read\_len);

data\_ptr[0] += image\_length / numChannels;

}

}

if (cdsFlag) {

memcpy(last\_frame, (data\_ptr[0] - image\_length), image\_length);

subtractCDS((unsigned short \*)image\_data, disk\_chunk\_size + 1, image\_width, image\_height, image\_length, factor, quadFlag, twoKFlag, numChannels);

}

data\_ptr[0] = (u\_char \*)image\_data;

for (k = 0; k < disk\_chunk\_size; k++) {

file\_length = file\_length\_sv;

frame\_deInterleave(superFrame\*file\_length >> 1, lut, (unsigned short \*)data\_ptr[0], image\_buffer);

frameRepair(image\_buffer, bad\_pix\_index, twoKFlag, image\_width, file\_width, file\_height, horizontalBin, superFrame, file\_img\_w, binFlag, segmented);

file\_length = file\_height \* min(file\_img\_w, (signed int)file\_width) \* 2 / horizontalBin;

if (ExtTriggerFlag && AD\_flag && triggerPos) {

if (!k)

\_lseek(tsmhandle[0], 2880L + (loops - triggerPos - 1)\*file\_length, SEEK\_SET);

else if (k == triggerPos)

\_lseek(tsmhandle[0], 2880L, SEEK\_SET);

}

\_write(tsmhandle[0], (void \*)image\_buffer, file\_length);

data\_ptr[0] += image\_length;

sprintf(str, "Deinterleaving %d of %d images", this\_img, images);

SetWindowText(acq\_hwndPB\_disk, str);

while ((int)(this\_img\*step\_size + 0.5) >= step\_cnt && step\_cnt < num\_steps) {

SendMessage(acq\_hwndPB\_disk, PBM\_STEPIT, 0, 0);

step\_cnt++;

}

this\_img++;

}

data\_ptr[0] = (u\_char \*)image\_data;

if (cdsFlag) {

memcpy(data\_ptr[0], last\_frame, image\_length);

data\_ptr[0] += image\_length;

}

}

if (loops % disk\_chunk\_size) {

for (k = 0; k < (loops % disk\_chunk\_size); k++) {

for (i = 0; i < numChannels; i++) {

\_read(diskF\_handle[i], (u\_char \*)data\_ptr[0], read\_len);

data\_ptr[0] += image\_length / numChannels;

}

}

if (cdsFlag) {

memcpy(last\_frame, (data\_ptr[0] - image\_length), image\_length);

subtractCDS((unsigned short \*)image\_data, (loops % disk\_chunk\_size) + 1, image\_width, image\_height, image\_length, factor, quadFlag, twoKFlag, numChannels);

}

data\_ptr[0] = (u\_char \*)image\_data;

for (k = 0; k < (loops % disk\_chunk\_size); k++) {

file\_length = file\_length\_sv;

frame\_deInterleave(superFrame\*file\_length >> 1, lut, (unsigned short \*)data\_ptr[0], image\_buffer);

frameRepair(image\_buffer, bad\_pix\_index, twoKFlag, image\_width, file\_width, file\_height, horizontalBin, superFrame, file\_img\_w, binFlag, segmented);

file\_length = file\_height \* min(file\_img\_w, (signed int)file\_width) \* 2 / horizontalBin;

if (ExtTriggerFlag && AD\_flag && triggerPos) {

if (!k)

\_lseek(tsmhandle[0], 2880L + (loops - triggerPos - 1)\*file\_length, SEEK\_SET);

else if (k == triggerPos)

\_lseek(tsmhandle[0], 2880L, SEEK\_SET);

}

\_write(tsmhandle[0], (void \*)image\_buffer, file\_length);

data\_ptr[0] += image\_length;

sprintf(str, "Deinterleaving %d of %d images", this\_img, images);

SetWindowText(acq\_hwndPB\_disk, str);

while ((int)(this\_img\*step\_size + 0.5) >= step\_cnt && step\_cnt < num\_steps) {

SendMessage(acq\_hwndPB\_disk, PBM\_STEPIT, 0, 0);

step\_cnt++;

}

this\_img++;

}

}

memset(data\_ptr[0], 0, file\_length);

\_lseeki64(tsmhandle[0], 2880L + (unsigned long long)images\*(unsigned long long)file\_length, SEEK\_SET);

\_write(tsmhandle[0], (void \*)data\_ptr[0], file\_length);

\_write(tsmhandle[0], (char \*)FitsEnder, dataLeft);

for (i = 0; i < numChannels; i++) {

\_close(diskF\_handle[i]);

DeleteFile(disk\_tmp\_file[i]);

}

}

DestroyWindow(acq\_hwndPB\_disk);

acq\_hwndPB\_disk = NULL;

goto cleanUp;

}

if (NI\_flag && shutter\_add\_frames) {

signed short \*scr\_ptr, \*dst\_ptr;

scr\_ptr = image\_data + shutter\_add\_frames \* image\_length / 2 ;

dst\_ptr = image\_data;

for (i = 0; i < loops; i++, scr\_ptr += (image\_length>>1), dst\_ptr += (image\_length >> 1))

memcpy(dst\_ptr, scr\_ptr, image\_length);

}

if (!fd\_no\_deinterleave\_flag || (num\_trials == 1)) {

if (cdsFlag) {

subtractCDS((unsigned short \*)image\_data, loops, image\_width, image\_height, image\_length, factor, quadFlag, twoKFlag, numChannels);

--loops;

}

loops -= NI\_add\_frames;

SendMessage(acq\_hwndPB, PBM\_SETPOS, (WPARAM)0, 0);

step\_cnt = 0;

step\_size = ((double)num\_steps) / loops;

unsigned int file\_length\_sv = file\_length;

for (i = 0; i < numFiles; ++i) {

data\_ptr[i] = (u\_char \*)image\_data;

if (NI\_flag && NI\_add\_frames)

data\_ptr[i] += image\_length \* (long)NI\_add\_frames;

for (j = 0, k = 0; j < loops; ++j, k += superFrame) {

file\_length = file\_length\_sv;

frame\_deInterleave(superFrame\*file\_length >> 1, lut, (unsigned short \*)data\_ptr[i], image\_buffer);

frameRepair(image\_buffer, bad\_pix\_index, twoKFlag, image\_width, file\_width, file\_height, horizontalBin, superFrame, file\_img\_w, binFlag, segmented);

file\_length = file\_height \* min(file\_img\_w, (signed int)file\_width) \* 2 / horizontalBin;

if (ExtTriggerFlag && AD\_flag && triggerPos) {

if (!j)

\_lseek(tsmhandle[i], 2880L + (loops - triggerPos - 1)\*file\_length, SEEK\_SET);

else if (j == triggerPos)

\_lseek(tsmhandle[i], 2880L, SEEK\_SET);

}

multiPixReadAv((signed short \*)image\_buffer);

if (!darkFlag || !j)

\_write(tsmhandle[i], (void \*)image\_buffer, (k <= images ? superFrame : images % superFrame)\*file\_length);

data\_ptr[i] += image\_length;

sprintf(str, "Deinterleaving %d of %d images", j, images);

SetWindowText(acq\_hwndPB, str);

while ((int)(j\*step\_size + 0.5) >= step\_cnt && step\_cnt < num\_steps) {

SendMessage(acq\_hwndPB, PBM\_STEPIT, 0, 0);

step\_cnt++;

}

}

}

memset(data\_ptr[0], 0, file\_length);

for (i = 0; i < numFiles; ++i) {

\_write(tsmhandle[i], (void \*)data\_ptr[0], file\_length);

\_write(tsmhandle[i], (char \*)FitsEnder, dataLeft);

}

}

else {

char tmp\_file\_name[MAX\_PATH];

char \*p;

int tmptsm\_handle;

loops\_sv = loops;

NI\_add\_frames\_sv = NI\_add\_frames;

mulit\_file\_len\_sv = file\_length;

strcpy(tmp\_file\_name, tsmfname[0]);

if (p = strstr(tmp\_file\_name, ".tsm")) {

\*p = '\0';

strcat(tmp\_file\_name, ".tmp");

if ((tmptsm\_handle = \_open(tmp\_file\_name, O\_MODE, S\_MODE)) != -1) {

\_write(tmptsm\_handle, (void \*)image\_data, image\_length\*loops);

\_close(tmptsm\_handle);

}

}

}

DestroyWindow(acq\_hwndPB);

acq\_hwndPB = NULL;

cleanUp:

if (debug)

MessageBox(main\_window\_handle, errorBuf, "message", MB\_OK);

if (pcdFlag != 1) {

for (int i = 0; i < numChannels; ++i)

pdv\_setsize(pdv\_pt[i], config\_num\_col, config\_num\_row\*live\_factor);

}

for (i = numFiles - 1; i >= 0; --i) {

if (tsmhandle[i]) {

\_close(tsmhandle[i]);

}

}

if (dark\_image\_buffer != NULL) {

\_aligned\_free(dark\_image\_buffer);

dark\_image\_buffer = NULL;

}

if (AD\_flag) {

if (BNC\_data != NULL)

\_aligned\_free(BNC\_data);

BNC\_data = NULL;

}

else if (NI\_flag && !darkFlag) {

if (NI\_data) {

if (NI\_data != NULL)

\_aligned\_free(NI\_data);

NI\_data = NULL;

}

}

if (esc\_hit) {

if (FastDiskFlag) {

for (i = 0; i < numChannels; i++) {

\_close(diskF\_handle[i]);

DeleteFile(disk\_tmp\_file[i]);

}

}

DestroyWindow(acq\_hwndPB);

acq\_hwndPB = NULL;

return FALSE;

}

return TRUE;

}

void rearrange\_File(int images, int file\_id, int numChannels, int quadFlag, int file\_img\_w, int segmented)

{

int j, k;

char str[256], tmp\_file\_name[MAX\_PATH];

u\_char \*data\_ptr;

char \*p;

int tmp\_handle, tsm\_handle;

unsigned long dataLeft;

int step\_cnt = 0;

double step\_size = ((double)num\_steps) / loops;

createAcqProgBar();

unsigned int file\_length\_sv = mulit\_file\_len\_sv;

SendMessage(acq\_hwndPB, PBM\_SETPOS, (WPARAM)0, 0);

loops = loops\_sv;

NI\_add\_frames = NI\_add\_frames\_sv;

if ((tsm\_handle = \_open(multi\_file\_names[file\_id], O\_APPEND | O\_BINARY | O\_SEQUENTIAL | O\_RDWR, S\_MODE)) == -1) {

MessageBox(main\_window\_handle, "Can't find the \*.tsm file.", "message", MB\_OK);

return;

}

strcpy(tmp\_file\_name, multi\_file\_names[file\_id]);

if (p = strstr(tmp\_file\_name, ".tsm")) {

\*p = '\0';

strcat(tmp\_file\_name, ".tmp");

if ((tmp\_handle = \_open(tmp\_file\_name, O\_BINARY | O\_RDONLY)) == -1) {

MessageBox(main\_window\_handle, "Can't find the \*.tsm file.", "message", MB\_OK);

return;

}

}

\_read(tmp\_handle, (void \*)image\_data, image\_length\*loops);

if (cdsFlag) {

subtractCDS((unsigned short \*)image\_data, loops, image\_width, image\_height, image\_length, 1, quadFlag, twoKFlag, numChannels);

--loops;

}

loops -= NI\_add\_frames;

data\_ptr = (u\_char \*)image\_data;

if (NI\_flag && NI\_add\_frames)

data\_ptr += image\_length \* (long)NI\_add\_frames;

\_lseek(tsm\_handle, 2880L, SEEK\_SET);

for (j = 0, k = 0; j < loops; ++j, k += superFrame) {

file\_length = file\_length\_sv;

frame\_deInterleave(superFrame\*file\_length >> 1, lut, (unsigned short \*)data\_ptr, image\_buffer);

frameRepair(image\_buffer, 0, twoKFlag, image\_width, file\_width, file\_height, horizontalBin, superFrame, file\_img\_w, binFlag, segmented);

file\_length = file\_height \* min(file\_img\_w, (signed int)file\_width) \* 2 / horizontalBin;

multiPixReadAv((signed short \*)image\_buffer);

\_write(tsm\_handle, (void \*)image\_buffer, (k <= images ? superFrame : images % superFrame)\*file\_length);

data\_ptr += image\_length;

sprintf(str, "Deinterleaving %d of %d images of %s", j, images, multi\_file\_names[file\_id]);

SetWindowText(acq\_hwndPB, str);

while ((int)(j\*step\_size + 0.5) >= step\_cnt && step\_cnt < num\_steps) {

SendMessage(acq\_hwndPB, PBM\_STEPIT, 0, 0);

step\_cnt++;

}

}

memset(data\_ptr, 0, file\_length);

\_write(tsm\_handle, (void \*)data\_ptr, file\_length);

dataLeft = sizeof(FitsHeader) - ((images + 1)\*file\_length) % sizeof(FitsHeader);

\_write(tsm\_handle, (char \*)FitsEnder, dataLeft);

\_close(tmp\_handle);

DeleteFile(tmp\_file\_name);

\_close(tsm\_handle);

DestroyWindow(acq\_hwndPB);

acq\_hwndPB = NULL;

}

void

stripAD\_data(unsigned short int \*image\_data, signed short int \*BNC\_data, char \*cameraType, int file\_height, int frame\_width, int frame\_height, int loops, int cds)

{

long dst\_inc, src\_inc, BNC\_frame\_len, BNC\_dst\_inc;

int i;

unsigned short int \*sptr, \*dptr;

signed short int \*BNC\_sptr, \*BNC\_dptr;

dptr = (unsigned short \*)image\_data;

sptr = (unsigned short \*)image\_data;

BNC\_dptr = (signed short \*)BNC\_data;

BNC\_sptr = (signed short \*)image\_data;

dst\_inc = frame\_width\*file\_height;

src\_inc = frame\_width\*frame\_height;

BNC\_frame\_len = frame\_width\*2;

BNC\_dst\_inc = frame\_width;

if (strstr(cameraType, "DW")) {

BNC\_sptr += dst\_inc;

if (cds)

BNC\_sptr += src\_inc;

}

else { // \*\*\*CCD39 and CCD67\*\*\*

sptr += frame\_width \* 2;

BNC\_dst\_inc = frame\_width\*2;

BNC\_frame\_len = frame\_width\*4;

}

for (i = 0; i < loops; i++) {

memcpy((void \*)BNC\_dptr, (void \*)BNC\_sptr, BNC\_frame\_len);

BNC\_dptr += BNC\_dst\_inc;

BNC\_sptr += src\_inc;

memcpy((void \*)dptr, (void \*)sptr, frame\_width\*file\_height \* 2);

dptr += dst\_inc;

sptr += src\_inc;

}

}

void frameRepair(unsigned short int \*image\_buffer, int bad\_pix\_index, int twoKFlag, unsigned int image\_width, unsigned int file\_width, unsigned int file\_height, int horizontal\_bin, int superFrame, int file\_img\_w, int binFlag, int segmented)

{

if (bad\_pix\_index && bad\_pixel\_ar[0][0]) {

//fudge bad pixels

int i, tmp;

signed short \*bad\_pt, \*nb\_pt;

for (i = 0; i < sizeof(bad\_pixel\_ar) / sizeof(bad\_pixel\_ar[0]); i++) {

if (bad\_pixel\_ar[i][0]) {

bad\_pt = (signed short \*)(image\_buffer + bad\_pixel\_ar[i][0] - 1);

int nn = 0;

tmp = 0;

for (int mm = 1; mm <= 8; mm++) {

if (bad\_pixel\_ar[i][mm] > 0) {

nb\_pt = (signed short \*)(image\_buffer + bad\_pixel\_ar[i][mm] - 1);

tmp = tmp + \*nb\_pt;

nn++;

}

}

if (nn > 0)

\*bad\_pt = tmp / nn;

}

else

break;

}

}

else if (twoKFlag) {

//fix bad rows and bad cols

signed short \*bad\_pt, \*nb\_pt1, \*nb\_pt2;

int ii;

if (bad\_row\_ar[0][0] && file\_width <= 2048) {

ii = 0;

while (bad\_row\_ar[ii][0]) {

bad\_pt = (signed short \*)(image\_buffer + bad\_row\_ar[ii][0] \* file\_width);

nb\_pt1 = (signed short \*)(image\_buffer + bad\_row\_ar[ii][3] \* file\_width);

nb\_pt2 = (signed short \*)(image\_buffer + bad\_row\_ar[ii][4] \* file\_width);

if (bad\_row\_ar[ii][3] == bad\_row\_ar[ii][4])

memcpy(bad\_pt, nb\_pt1, file\_width \* 2);

else {

for (unsigned int kk = 0; kk < file\_width; kk++) {

\*bad\_pt = (\*nb\_pt1 + \*nb\_pt2) / 2;

bad\_pt++;

nb\_pt1++;

nb\_pt2++;

}

}

ii++;

}

}

if (bad\_col\_ar[0][0] && file\_width <= 2048) {

ii = 0;

int y1, y2;

while (bad\_col\_ar[ii][0]) {

if (bad\_col\_ar[ii][1] == 0) {

y1 = 0;

y2 = file\_height / 2 - 1;

}

else {

y1 = file\_height / 2;

y2 = file\_height - 1;

}

bad\_pt = (signed short \*)image\_buffer + y1\*file\_width + bad\_col\_ar[ii][0];

nb\_pt1 = (signed short \*)image\_buffer + y1\*file\_width + bad\_col\_ar[ii][3];

nb\_pt2 = (signed short \*)image\_buffer + y1\*file\_width + bad\_col\_ar[ii][4];

for (int kk = y1; kk <= y2; kk++) {

if (bad\_row\_ar[ii][3] == bad\_row\_ar[ii][4])

\*bad\_pt = \*nb\_pt1;

else

\*bad\_pt = (\*nb\_pt1 + \*nb\_pt2) / 2;

bad\_pt += file\_width;

nb\_pt1 += file\_width;

nb\_pt2 += file\_width;

}

ii++;

}

}

if (file\_img\_w != file\_width) {

unsigned short int \*sptr, \*dptr;

if (segmented)

sptr = image\_buffer + file\_width / 2;

else {

if (OffCenter\_crop)

sptr = image\_buffer + file\_width - file\_img\_w;

else

sptr = image\_buffer + (file\_width - file\_img\_w) / 2;

}

dptr = image\_buffer;

int l, m;

for (m = 0; m < (signed int)file\_height; m++) {

for (l = 0; l < file\_img\_w; l++)

\*dptr++ = \*sptr++;

sptr += file\_width - file\_img\_w;

}

}

if (horizontal\_bin > 1) {

unsigned short int \*sptr, \*dptr;

sptr = image\_buffer;

dptr = image\_buffer;

int l, m, kk;

unsigned long bin\_val;

for (m = 0; m < (signed int)file\_height; m++) {

for (l = 0; l < min(file\_img\_w, (signed int)file\_width) / horizontal\_bin; l++) {

bin\_val = 0;

for (kk = 0; kk < horizontal\_bin; kk++)

bin\_val += (unsigned long)\*sptr++;

\*dptr++ = (unsigned short)(bin\_val / horizontal\_bin);

}

}

}

}

}

void saveNI\_data(long num\_frames, char filename[MAX\_PATH], short int numBNC\_chan, short int BNC\_ratio, float64 \*NI\_data, int delay, int isTwoK)

{

// write BNC data

char BNCname[MAX\_PATH];

char \*p;

int handle, i, m;

short neg\_numChan;

float64 \*NI\_one\_chan\_data = NULL, \*NI\_dpt, \*NI\_spt;

strcpy(BNCname, filename);

p = strstr(BNCname, ".tsm");

\*p = '\0';

strcat(BNCname, ".tbn");

if ((handle = \_open(BNCname, O\_MODE, S\_MODE)) == -1)

return;

NI\_one\_chan\_data = (float64 \*)\_aligned\_malloc(num\_frames\*BNC\_ratio\*sizeof(float64), 2);

neg\_numChan = -numBNC\_chan;

\_write(handle, (void \*)&neg\_numChan, sizeof(neg\_numChan));

\_write(handle, (void \*)&BNC\_ratio, sizeof(BNC\_ratio));

for (i = 0; i < numBNC\_chan; i++) {

NI\_dpt = NI\_one\_chan\_data;

if (cdsFlag) {

if (BNC\_ratio)

NI\_spt = NI\_data + numBNC\_chan\*(NI\_add\_frames + BNC\_ratio) + i;

else

NI\_spt = NI\_data + numBNC\_chan\*NI\_add\_frames + i;

}

else {

if (BNC\_ratio > 2)

NI\_spt = NI\_data + numBNC\_chan \* (max(0, (NI\_add\_frames - 1)) + BNC\_ratio) + i;

else

NI\_spt = NI\_data + numBNC\_chan \* max(0, (NI\_add\_frames - 1)) + i;

}

NI\_spt += shutter\_add\_frames \* numBNC\_chan\*BNC\_ratio;

for (m = 0; m < num\_frames\*BNC\_ratio; m++) {

\*NI\_dpt++ = \*NI\_spt;

NI\_spt += numBNC\_chan;

}

\_write(handle, NI\_one\_chan\_data, num\_frames\*BNC\_ratio\*sizeof(float64));

}

if (NI\_one\_chan\_data)

\_aligned\_free(NI\_one\_chan\_data);

\_commit(handle);

\_close(handle);

}

void saveBNC\_data(long num\_frames, char filename[MAX\_PATH], short int numBNC\_chan, short int BNC\_ratio, signed short int \*BNC\_data, char \*cameraType, int cam\_num\_col, int stripes)

{

// write BNC data

char BNCname[MAX\_PATH];

char \*p;

int handle;

strcpy(BNCname, filename);

p = strstr(BNCname, ".tsm");

\*p = '\0';

strcat(BNCname, ".tbn");

if ((handle = \_open(BNCname, O\_MODE, S\_MODE)) == -1)

return;

\_write(handle, (void \*)&numBNC\_chan, sizeof(numBNC\_chan));

\_write(handle, (void \*)&BNC\_ratio, sizeof(BNC\_ratio));

saveBNC\_block(num\_frames, handle, 0, numBNC\_chan, BNC\_ratio, BNC\_data, cameraType, 0, cam\_num\_col, stripes);

\_commit(handle);

\_close(handle);

}

static unsigned long lastDiskBNC;

void saveBNC\_block(long num\_frames, int handle, long tt\_frames\_flag, short int numBNC\_chan, short int BNC\_ratio, signed short int \*BNC\_data, char \*cameraType, long lastDiskFrame, int cam\_num\_col, int stripes)

{

int k, i;

unsigned short \*BNC\_dptr0, \*BNC\_sptr, \*tmp\_ptr, \*chan\_ptr;

signed short int \*tmpBNC = (signed short int \*)\_aligned\_malloc(num\_frames\*BNC\_ratio\*numBNC\_chan\*sizeof(signed short int), 2);

if (tmpBNC == NULL)

return;

BNC\_sptr = (unsigned short \*)BNC\_data;

BNC\_dptr0 = (unsigned short \*)tmpBNC;

if (strstr(cameraType, "DW")) {

BNC\_sptr += cam\_num\_col;

for (i = 0; i < num\_frames; i++) {

for (k = 0; k < numBNC\_chan; k++) {

tmp\_ptr = BNC\_dptr0 + k\*num\_frames + i;

chan\_ptr = BNC\_sptr + k;

\*tmp\_ptr = \*chan\_ptr;

\*tmp\_ptr = (\*tmp\_ptr) << 2;

}

BNC\_sptr += cam\_num\_col;

}

}

else if (strstr(cameraType, "CCD67")) {

if (lastDiskFrame <= 1)

BNC\_sptr += cam\_num\_col\*2;

for (i = 0; i < num\_frames; i++) {

for (k = 0; k < numBNC\_chan; k++) {

tmp\_ptr = BNC\_dptr0 + k\*num\_frames\*BNC\_ratio + i\*BNC\_ratio;

for (int m = 0; m < BNC\_ratio; m++) {

if (m < 32)

chan\_ptr = BNC\_sptr + m\*8 + k;

else

chan\_ptr = BNC\_sptr + cam\_num\_col + (m-32)\*8 + k;

\*tmp\_ptr = \*chan\_ptr;

\*tmp\_ptr = (\*tmp\_ptr) << 2;

tmp\_ptr ++;

}

}

BNC\_sptr += cam\_num\_col\*2;

}

}

else {

BNC\_sptr += cam\_num\_col\*2;

for (i = 0; i < num\_frames; i++) {

for (k = 0; k < numBNC\_chan; k++) {

tmp\_ptr = BNC\_dptr0 + k\*num\_frames + i;

chan\_ptr = BNC\_sptr + k;

\*tmp\_ptr = \*chan\_ptr;

\*tmp\_ptr = (\*tmp\_ptr) << 2;

}

BNC\_sptr += cam\_num\_col\*2;

}

}

if (tt\_frames\_flag) {

unsigned long BNC\_Length = (unsigned long)num\_frames\*BNC\_ratio\*2;

signed short \*BNC\_ptr = (signed short \*)tmpBNC;

for (k = 0; k < numBNC\_chan; k++) {

\_lseek(handle,sizeof(BNC\_ratio)+sizeof(numBNC\_chan)+tt\_frames\_flag\*BNC\_ratio\*k\*2+max(lastDiskFrame-1, 0)\*BNC\_ratio\*2,SEEK\_SET);

\_write(handle,BNC\_ptr,BNC\_Length);

BNC\_ptr += num\_frames\*BNC\_ratio;

}

}

else {

unsigned long BNC\_Length = (unsigned long)num\_frames\*BNC\_ratio\*numBNC\_chan\*2;

memcpy(BNC\_data, tmpBNC, BNC\_Length);

\_write(handle, (signed short \*)BNC\_data, BNC\_Length);

}

\_aligned\_free(tmpBNC);

}

void SM\_makeHeader(int images, int numExposures, int rows, int columns, char \*camera, double exposure)

{

char buf[81];

int card = 0, year = 2013, month = 4, day = 21, hour = 10, minute = 20, second = 40;

memset(&FitsHeader, 0x20, sizeof(FitsHeader));

sprintf(buf, "SIMPLE = %20s", "T");

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "BITPIX = %20d", 16);

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "NAXIS = %20d", 2);

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "NAXIS1 = %20d", columns);

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "NAXIS2 = %20d", rows);

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "NAXIS3 = %20d", images);

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "DATE = %04d-%02d-%02dT%02d:%02d:%02d", year, month, day, hour, minute, second);

strncpy(FitsHeader[card++], buf, strlen(buf));

strcpy(buf, "ORIGIN = SciMeasure Analytical Systems, Inc.");

strncpy(FitsHeader[card++], buf, strlen(buf));

strcpy(buf, "CREATOR = Turbo-SM");

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "INSTRUME= %s, %s", camera, config\_list[curConfig]);

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "EXPOSURE= %20.10lf", exposure\*numExposures);

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "SATURATE= %20d", 16383);

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "DATAMAX = %20.0lf", 16383.0);

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "DATAMIN = %20.0lf", 0.0);

strncpy(FitsHeader[card++], buf, strlen(buf));

sprintf(buf, "END");

strncpy(FitsHeader[card++], buf, strlen(buf));

}

void

subtractCDS(unsigned short int \*image\_data, int loops, unsigned int width, unsigned int height, unsigned int length, int factor, int QUAD, int TWOK, int num\_pdvChan)

{

int l, m, n, row, col, doublecols = 0, rows = 0, cols = 0;

unsigned short int \*new\_data, \*old\_data, \*reset\_data;

int CDS\_add = 256;

int step\_cnt, total\_cds\_loops;

double step\_size;

char str[256];

total\_cds\_loops = loops;

if (acq\_hwndPB && !Ddisk\_flag) {

SendMessage(acq\_hwndPB, PBM\_SETPOS, (WPARAM)0, 0);

step\_cnt = 0;

step\_size = ((double)num\_steps) / total\_cds\_loops;

}

if (Falcon\_flag) {

rows = height / 2;

cols = width;// \*2;

new\_data = image\_data;

reset\_data = image\_data + cols;

old\_data = image\_data + length / 2;

}

else if (TWOK) {

if (factor == 1)

rows = height \* (num\_pdvChan>>1);

else

rows = (height + 2) / (2 \* factor) - 2;

cols = width / num\_pdvChan;

doublecols = cols \* 2;

new\_data = image\_data;

reset\_data = image\_data + cols;

if (factor == 1)

old\_data = image\_data + length / 2;

else

old\_data = image\_data + (rows + 2)\*doublecols;

}

else if (QUAD) {

rows = height / 2;

cols = width;

new\_data = image\_data;

reset\_data = image\_data + width;

old\_data = image\_data + length / 2;

}

else {

rows = height;

cols = width / 2;

new\_data = image\_data;

reset\_data = image\_data + cols;

old\_data = image\_data + length / 2;

}

if (factor == 1) {

for (--loops, m = 0; loops > 0; --loops, ++m) {

for (row = 0; row < rows; ++row) {

for (col = cols; col; --col)

\*new\_data++ = CDS\_add + \*old\_data++ - \*reset\_data++;

new\_data += cols;

reset\_data += cols;

old\_data += cols;

}

if (acq\_hwndPB && !Ddisk\_flag) {

sprintf(str, "CDS Subtraction: %d of %d images", m, total\_cds\_loops);

SetWindowText(acq\_hwndPB, str);

while ((int)(m\*step\_size + 0.5) >= step\_cnt && step\_cnt < num\_steps) {

SendMessage(acq\_hwndPB, PBM\_STEPIT, 0, 0);

step\_cnt++;

}

}

}

}

else {

for (m = 1; loops; --loops, m += factor) {

for (n = 4; n; --n) {

for (l = factor - 1; l; --l, ++m) {

for (row = rows; row; --row) {

for (col = cols; col; --col)

\*new\_data++ = CDS\_add + \*old\_data++ - \*reset\_data++;

new\_data += cols;

old\_data += cols;

reset\_data += cols;

}

new\_data += doublecols \* 2;

old\_data += doublecols \* 2;

reset\_data += doublecols \* 2;

}

if (loops > 0) {

old\_data += width \* 3 \* height / 4 - doublecols;

for (row = rows; row; --row) {

for (col = cols; col; --col)

\*new\_data++ = CDS\_add + \*old\_data++ - \*reset\_data++;

new\_data += cols;

old\_data += cols;

reset\_data += cols;

}

old\_data -= width \* 3 \* height / 4 - doublecols;

new\_data += doublecols;

old\_data += doublecols;

reset\_data += doublecols;

++m;

m -= factor;

}

}

}

}

}

extern int image\_chan\_order[MAXOFFSETS];

int

makeLookUpTable(unsigned int \*Lut, int image\_width, int image\_height, int file\_width, int file\_height, int stripes, int layers, int factor, int QUAD, int CDS, int ROTATE, int TWOK)

{

// int SEGS, file\_offset, image\_offset, frame\_length, image\_length, file\_length;

int SEGS, image\_length, file\_length;

// int frame, segment, row, rowIndex, destIndex, rows, cols, swid, dwid, srcIndex;

int segment, row, rowIndex, destIndex, rows, cols, swid, dwid, srcIndex;

static int twokchannel[] = { 3, 2, 0, 1, 7, 6, 4, 5, 9, 8, 10, 11, 13, 12, 14, 15 };

static int onekchannel[] = { 3, 2, 0, 1, 5, 4, 6, 7 };

int \*channelOrder = NULL;

int num\_pdvChan;

SEGS = stripes\*layers;

image\_length = image\_width\*image\_height;

file\_length = file\_width\*file\_height;

rows = file\_height/layers;

if (TWOK) {

if (stripes == 8) {

channelOrder = twokchannel;

num\_pdvChan = 4;

}

else {

channelOrder = onekchannel;

num\_pdvChan = 2;

}

}

swid = image\_width / stripes;

dwid = file\_width / stripes;

#ifdef DAVINCIBITFLIP

int tab, bit;

unsigned int val;

unsigned \_\_int64 entry, \*vptr;

for (tab = 0, vptr = bitLut; tab < 4; ++tab)

for (val = 0; val < 0x10000; ++val, ++vptr) {

entry = 0;

for (bit = 0; bit < 16; ++bit)

if (val&bitMask[bit])

entry |= bitPattern[tab][bit];

\*vptr = entry;

}

#endif

if (Falcon\_flag) {

omp\_set\_num\_threads(omp\_get\_num\_procs());

#pragma omp parallel

{

int segment;

#pragma omp parallel for private(row,cols,segment,rowIndex,srcIndex,destIndex)

for (segment = 0; segment < stripes; ++segment) {

srcIndex = (segment == 0);

for (row = 0, rowIndex = 0; row < rows; ++row, rowIndex += file\_width) {

for (destIndex = rowIndex + dwid \* segment, cols = dwid; cols > 0; --cols, ++destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

if (CDS)

srcIndex += dwid \* SEGS;

}

}

#pragma omp parallel for private(row,cols,segment,rowIndex,srcIndex,destIndex)

for (segment = stripes; segment < SEGS; ++segment) {

srcIndex = segment;

for (row = 0, rowIndex = file\_length - file\_width; row < rows; ++row, rowIndex -= file\_width) {

for (destIndex = rowIndex + dwid \* (segment - SEGS / 2), cols = dwid; cols > 0; --cols, ++destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

if (CDS)

srcIndex += dwid \* SEGS;

}

}

}

}

else if (TWOK) {

omp\_set\_num\_threads(omp\_get\_num\_procs());

#pragma omp parallel

{

int segment;

#pragma omp parallel for private(row,cols,segment,rowIndex,srcIndex,destIndex)

for (segment = 0; segment < stripes; ++segment) {

srcIndex = channelOrder[segment] % 4;

if (num\_pdvChan == 4)

srcIndex += (image\_length / 4)\*(channelOrder[segment] / 4);

for (row = 0, rowIndex = 0; row < rows; ++row, rowIndex += file\_width) {

for (destIndex = rowIndex + dwid\*segment, cols = dwid; cols > 0; --cols, ++destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS / num\_pdvChan;

}

if (CDS)

srcIndex += dwid\*SEGS / num\_pdvChan;

}

}

#pragma omp parallel for private(row,cols,segment,rowIndex,srcIndex,destIndex)

for (segment = stripes; segment < SEGS; ++segment) {

srcIndex = channelOrder[segment] % 4;

if (num\_pdvChan == 4)

srcIndex += (image\_length / 4)\*(channelOrder[segment] / 4);

else

srcIndex += (image\_length / 2);

for (row = 0, rowIndex = file\_length - file\_width; row < rows; ++row, rowIndex -= file\_width) {

for (destIndex = rowIndex + dwid\*(segment - SEGS / 2), cols = dwid; cols > 0; --cols, ++destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS / num\_pdvChan;

}

if (CDS)

srcIndex += dwid\*SEGS / num\_pdvChan;

}

}

}

}

else {

if (CDS) {

swid = swid / 2;

image\_width = image\_width / 2;

image\_length = image\_length / 2;

}

if (layers == 2) {

// TOP LAYER

if (QUAD) {

for (segment = 0; segment < SEGS; segment += 4) {

srcIndex = (channelOrder == NULL) ? segment : channelOrder[segment];

for (row = 0, rowIndex = 0; row < rows; ++row, rowIndex += image\_width) {

for (destIndex = rowIndex + (segment / 4) \* 2 \* swid, cols = swid; cols > 0; --cols, ++destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

if (CDS)

srcIndex += swid\*SEGS;

}

}

for (segment = 1; segment < SEGS; segment += 4) {

srcIndex = (channelOrder == NULL) ? segment : channelOrder[segment];

for (row = 0, rowIndex = 0; row < rows; ++row, rowIndex += image\_width) {

for (destIndex = rowIndex + (segment / 4 + 1) \* 2 \* swid - 1, cols = swid; cols > 0; --cols, --destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

if (CDS)

srcIndex += swid\*SEGS;

}

}

for (segment = 2; segment < SEGS; segment += 4) {

srcIndex = (channelOrder == NULL) ? segment : channelOrder[segment];

for (row = 0, rowIndex = image\_length - image\_width; row < rows; ++row, rowIndex -= image\_width) {

for (destIndex = rowIndex + (segment / 4) \* 2 \* swid, cols = swid; cols > 0; --cols, ++destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

if (CDS)

srcIndex += swid\*SEGS;

}

}

for (segment = 3; segment < SEGS; segment += 4) {

srcIndex = (channelOrder == NULL) ? segment : channelOrder[segment];

for (row = 0, rowIndex = image\_length - image\_width; row < rows; ++row, rowIndex -= image\_width) {

for (destIndex = rowIndex + ((segment / 4) + 1) \* 2 \* swid - 1, cols = swid; cols > 0; --cols, --destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

if (CDS)

srcIndex += swid\*SEGS;

}

}

}

else {

/\* for (segment = 1; segment < SEGS; segment += 2) {

srcIndex = (channelOrder == NULL) ? segment : channelOrder[segment];

for (row = rows - 1, rowIndex = (rows - 1)\*image\_width; row >= rows; --row, rowIndex -= image\_width) {

for (destIndex = rowIndex + (segment / 2)\*swid, cols = swid; cols > 0; --cols, ++destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

}

}\*/

/\* // CCID83 noise prog

int cam\_board = 4;

long board\_size = Height\*Width/2;

for (segment = 0; segment < SEGS/cam\_board; ++segment) {

srcIndex = segment\*cam\_board;

for (row = 0, rowIndex = 2L\*segment\*Height\*Width/SEGS; row < rows/layers; ++row,rowIndex += Width) {

for (destIndex = rowIndex, cols = swid; cols > 0; --cols, ++destIndex) {

Lut[destIndex] = srcIndex;

Lut[destIndex + cols\*2 - 1] = srcIndex + 3;

Lut[destIndex + board\_size] = srcIndex + 1;

Lut[destIndex + board\_size + cols\*2 - 1] = srcIndex + 2;

srcIndex += SEGS;

}

}

}\*/

// CCID79 && CCID66

// TOP LAYER

int mirror = 1, inc = -1;

rows = file\_height;

for (segment = 0; segment < SEGS / 2; segment++) {

srcIndex = image\_chan\_order[segment];

for (row = 0, rowIndex = 0; row < rows >> 1; ++row, rowIndex += image\_width) {

for (destIndex = rowIndex + (segment + mirror)\*swid - mirror, cols = swid; cols > 0; --cols, destIndex += inc) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

}

}

// BOTTOM LAYER

for (segment = SEGS / 2; segment < SEGS; segment++) {

srcIndex = image\_chan\_order[segment];

for (row = rows - 1, rowIndex = (rows - 2)\*image\_width; row >= rows >> 1; --row, rowIndex -= image\_width) {

for (destIndex = rowIndex + (segment + mirror)\*swid - mirror, cols = swid; cols > 0; --cols, destIndex += inc) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

}

}

if (ROTATE) {

int half\_length = image\_length/2;

unsigned int temp;

for (int i = 0, j = image\_length - 1; i < half\_length; ++i, --j) {

temp = Lut[i];

Lut[i] = Lut[j];

Lut[j] = temp;

}

}

if (multi\_pix\_read > 1) {

unsigned int \*lut\_tmp = (unsigned int \*)\_aligned\_malloc(file\_length \* sizeof(lut[0]), 2);

int img\_jump = image\_width / multi\_pix\_read;

srcIndex = 0;

destIndex = 0;

for (int n = 0; n < file\_height; n++) {

for (int m = 0; m < img\_jump; m++) {

lut\_tmp[destIndex] = lut[srcIndex];

srcIndex++;

for (int i = 1; i < multi\_pix\_read; i++) {

lut\_tmp[destIndex + img\_jump\*i] = lut[srcIndex];

srcIndex++;

}

destIndex++;

}

destIndex += img\_jump \* (multi\_pix\_read - 1);

}

memcpy(lut, lut\_tmp, file\_length \* sizeof(lut[0]));

\_aligned\_free(lut\_tmp);

}

}

}

else { // LAYERS != 2

if (SEGS == 20) {

if (ROTATE) { // rotate image

for (segment = 0; segment < SEGS; ++segment) {

srcIndex = (channelOrder == NULL) ? segment : channelOrder[segment];

for (row = 0, rowIndex = image\_height\*image\_width; row < rows; ++row, rowIndex -= image\_width) {

for (destIndex = rowIndex - segment\*swid - 1, cols = swid; cols > 0; --cols, --destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

}

}

}

else {

for (segment = 0; segment < SEGS; ++segment) {

srcIndex = (channelOrder == NULL) ? segment : channelOrder[segment];

for (row = 0, rowIndex = 0; row < rows; ++row, rowIndex += image\_width) {

for (destIndex = rowIndex + segment\*swid, cols = swid; cols > 0; --cols, ++destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

}

}

}

}

else {

if (ROTATE) { // rotate image

for (segment = 0; segment < SEGS; ++segment) {

srcIndex = 4 \* (segment % 4) + segment / 4; // DM0018 segment mapping for 16 channels

for (row = 0, rowIndex = image\_height\*image\_width; row < rows; ++row, rowIndex -= image\_width) {

for (destIndex = rowIndex - segment\*swid - 1, cols = swid; cols > 0; --cols, --destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

}

}

}

else {

srcIndex = 0;

destIndex = 0;

if (SEGS == 2) {

for (row = 0; row < rows; ++row) {

for (cols = swid; cols > 0; --cols, ++destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

srcIndex -= image\_width - 1;

destIndex += swid-1;

for (cols = swid; cols > 0; --cols, --destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

srcIndex -= SEGS - 1;

destIndex += swid+1;

}

}

else {

for (row = 0; row < rows; ++row) {

for (segment = 0; segment < SEGS; ++segment) {

for (cols = swid; cols > 0; --cols, ++destIndex) {

Lut[destIndex] = srcIndex;

srcIndex += SEGS;

}

srcIndex -= image\_width - 1;

}

srcIndex += image\_width\*(1 + CDS) - SEGS;

}

}

}

}

}

}

return 0;

}

void frame\_deInterleave(int length, unsigned \*lookuptable, unsigned short \*old\_images, unsigned short \*new\_images)

{

for (int i = 0; i < length; ++i)

\*(new\_images++) = old\_images[\*(lookuptable++)];

}

/\*\*\*\*\*for NI acquisition\*\*\*/

#ifdef \_USE\_NI

// add c:\Program Files (x86)\National Instruments\Shared\ExternalCompilerSupport\C\lib64\msvc\NIDAQmx.lib to Linker>>Input>>Additional Dependencies

int32 CVICALLBACK DoneCallback(TaskHandle taskHandle, int32 status, void \*callbackData)

{

int32 error = 0;

char errBuff[2048] = { '\0' };

// Check to see if an error stopped the task.

DAQmxErrChk(status);

Error:

if (DAQmxFailed(error)) {

DAQmxGetExtendedErrorInfo(errBuff, 2048);

DAQmxClearTask(taskHandle);

fprintf(stdout,"DAQmx Error: %s\n", errBuff);

}

return 0;

}

#endif

/\*\*\*end of NI\*\*\*/