Radiometric Digital Acquisition Developers Kit
For
National Instruments Acquisition Module

(Document Number 700-00000063-R01)

10503 Timberwood Circle
Suite 120
Louisville, KY 40223
(502) 423-7225

Radiometric Digital Acquisition Developers Kit
Document Number 700-00000063
May 2006

DISCLAIMER

NOTICE OF PROPRIETARY RIGHTS

Lumitron
10503 Timberwood Circle
Suite 120
Louisville, KY 40223
(502) 423-7225
FAX (502) 423-7064

Table of Contents

1 Introduction

This reference manual has been written to assist the developer in creating custom digital acquisition solutions for Lumitron's line of cameras based on the 320A Embeddable Camera Electronics System hardware as it integrates with a National Instruments NI-1426 Camera Link module.

An overview of the system requirements and a detailed description of the interface are also provided.

This document has been written with the assumption that the developer is knowledgeable with the design of Microsoft Windows OS based applications.

2 General Requirements

Several components are required to begin developing digital acquisition software using Lumitron's interface libraries. They include software and hardware components from both Lumitron and National Instruments.

The following is a list of files that Lumitron provides with the ECS-320A Radiometric Developer's Kit (P/N: 600-00000030):

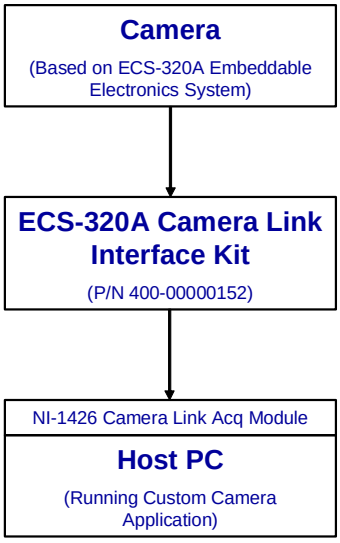
- LtronImgIR.dll Dynamic Link Library
- LtronImgIR.lib Library File
- LtronImgIR_Defines Header File
- LtronImgIR_Exports Header File
- LtronImgIR_Structs Header File
- LtronSptIR.dll Dynamic Link Library
- LtronSptIR.lib Library File
- LtronSptIR_Exports Header File
- LtronView.sln Microsoft Visual C++ .NET Project (complete solution)

The LtronView project is provided as a test and debugging application that can be used as a "quick start" reference application. The majority of the LtronImgIR library functions are utilized in this functional test application and portions of the code are also referenced in this manual.

In addition to the above files it will also be necessary to have National Instruments Image Acquisition software (NI-IMAQ) installed on the target platform. The NI software is required due to the implicit linking of the NI-IMAQ functions called in the LtronImgIR library. The current release of NI-IMAQ is 3.5.1 and is available through the installation of their Vision Acquisition 8.0 software (provided with NI module).

The target frame grabber module that this manual references is the NI-1426 Camera Link. Although it is available in 16 or 32 MB versions; the 16 MB version typically is sufficient.

The diagram to the right shows a typical connection using a Lumitron Camera Link based camera, Camera Link interface kit, and host computer with NI acquisition module.



3 LtronImgIr Library Function Reference

The functions listed in this section make up the interface for the LtronImgIR.dll library. Use of these functions will require access to the NI imaq.dll library.

3.1 lir_SetInterfaceID

Description:

Exported routine to set the National Instruments Interface ID filename. This file contains a link to a camera configuration file which contains configuration data for the acquisition module.

Function Declaration:

void LTRONIMGIR lir_SetInterfaceID(LPCSTR szIID, int camera_num)

Parameters:

LPCSTR szIID: filename of IID file (full path not needed).

int camera_num: associated camera index number (0 is default).

Returns:

None

Example:

```
// Set Interface ID for the NI Board
lir_SetInterfaceID("LTron_240 1426.iid", CAM_NUM_0);
```

3.2 lir_PowerUp

Description:

Exported routine to power up a camera.

NOTE: This function is currently not implemented.

Function Declaration:

void LTRONIMGIR lir_PowerUp(void)

Parameters:

None.

Returns:

None.

3.3 lir_PowerDown

Description:

Exported routine to power down a camera.

NOTE: This function is currently not implemented.

Function Declaration:

void LTRONIMGIR lir_PowerDown(void)

Parameters:

None.

Returns:

None.

3.4 lir_GetHeader

Description:

Exported routine to retrieve camera digital header information. This data is located on the first line of every frame transmitted. See Appendix A for further information about the data members of the header structure.

Function Declaration:

void LTRONIMGIR lir_GetHeader(ptr_img_Header header, int camera_num)

Parameters:

ptr_img_Header header: pointer to the structure to be filled.

int camera_num: associated camera index number (0 is default).

Returns:

None.

Example:

```
img_Header   sHdrInfo = {0};
// Get header data (calibration data)
lir_GetHeader(&sHdrInfo, 0);.
```

3.5 lir_PauseAcquisition

Description:

Exported routine to pause the acquisition thread.

Function Declaration:

void LTRONIMGIR lir_PauseAcquisition(int camera_num)

Parameters:

Int camera_num: index of camera to be paused (default is 0).

Returns:

None.

3.6 lir_ResumeAcquisition

Description:

Exported routine to resume the acquisition thread.

Function Declaration:

void LTRONIMGIR lir_ResumeAcquisition(int camera_num)

Parameters:

Int camera_num: index of camera to be paused (default is 0).

Returns:

None.

3.7 lir_CloseImageIR

Description:

Exported routine to stop entire acquisition process and close (release resources) associated with the current instance of the DLL.

Function Declaration:

int LTRONIMGIR lir_CloseImageIR(void)

Parameters:

None.

Returns:

Nonzero for success.

Example:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
…
switch (message)
{
case (WM_PAINT):
      …
case (WM_DESTROY):
      SaveSetup(szConfig);
      …
      lir_CloseImageIR();
      PostQuitMessage(0);
      break;
…
}
```

3.8 lir_CheckActive

Description:

Exported routine to check if the desired camera is active. The routine will attempt to initialize the acquisition parameters and perform a 'snap' to confirm the camera is available to begin continous acquisition (start polling).

Function Declaration:

bool LTRONIMGIR lir_CheckActive(int camera_num)

Parameters:

Int camera_num: index of camera to be checked (default is 0).

Returns:

True – if camera is active, false if not.

Example:

```
/***** First detect which cameras are active *****/

// Test if camera 0 is present
if (lir_CheckActive(CAM_NUM_0))
{
      // Add to mask
      nActCamDetect |= CAM_1_MASK;
      // Increment max counter
      nCamMax++;
}

// Test if camera 1 is present
if (lir_CheckActive(CAM_NUM_1))
{
      // Add to mask
      nActCamDetect |= CAM_2_MASK;
      // Increment max counter
      nCamMax++;
}
```

3.9 lir_StartCamera

Description:

Exported routine to 'start' continous acquisition on a desired camera. Use the lir_CheckActive routine first to confirm the camera is active.

Function Declaration:

bool LTRONIMGIR lir_StartCamera(int camera_num)

Parameters:

Int camera_num: index of camera to be checked (default is 0).

Returns:

True – if camera is polling (acquiring), false if not.

Example:

```
// If we have at least one camera
if (nCamMax > 0)
{
      // Now start each camera we need
      for (i = CAM_NUM_0; i < MAX_NUM_CAMERAS; i++)
      {
            // If camera is active
            if (nActCamDetect & (CAM_1_MASK << i))
            {
                  // Attempt to start camera
                  if (lir_StartCamera(i))
                  {
                        // If this is the first camera started
                        if (!bFirstCam)
                        {
                              // Set the main window camera index
                              nCamNum = i;
                              // Set flag
                              bFirstCam = true;
                        }
                  }
                  // Camera failed to start
                  else
                  {
                        // Clear the mask
                        nActCamDetect &= ~(CAM_1_MASK << i);
                        // Decrement counter
```

```
                        nCamMax--;
                    }
                }
            }
        }
```

3.10 lir_GetCameraInfo

Description:

Exported routine to retrieve the camera information structure. See Appendix B for further information about the data members of the camera information structure.

Function Declaration:

int LTRONIMGIR lir_GetCameraInfo(ptr_img_CameraInfo pIRinfo, int camera_num)

Parameters:

ptr_img_CameraInfo pIRinfo: pointer to structure to be filled.

int camera_num: associated camera index number.

Returns:

Non-zero if succesful.

Example:

```
switch (message)
{
case (WM_INITDIALOG):
      hDC = BeginPaint( hDlg, &ps );
      EndPaint( hDlg, &ps );
      lir_GetCameraInfo(&sCamInfo, nCamNum);

      if (nConv2Temp == UNITS_C)
      {
            SetDlgItemInt( hDlg, IDC_VIDFS, (int)F_TO_C(sCamInfo.VideoFull), TRUE);
            SetDlgItemInt( hDlg, IDC_VIDZS, (int)F_TO_C(sCamInfo.VideoZero), TRUE);
      }
      else
      {
            SetDlgItemInt( hDlg, IDC_VIDFS, (int)sCamInfo.VideoFull, TRUE);
            SetDlgItemInt( hDlg, IDC_VIDZS, (int)sCamInfo.VideoZero, TRUE);
      }

      …
      return (TRUE);

      …
}
```

3.11 lir_SetCameraInfo

Description:

Currently not used.

Function Declaration:

int LTRONIMGIR lir_SetCameraInfo(ptr_img_CameraInfo pIRinfo, int camera_num)

Parameters:

ptr_img_CameraInfo pIRinfo: pointer to structure to be filled.

int camera_num: associated camera index number.

Returns:

Non-zero if succesful.

3.12 lir_GetCamSerial

Description:

Exported routine to retrieve the desired camera's serial number.

Function Declaration:

UWord32 LTRONIMGIR lir_GetCamSerial(int camera_num)

Parameters:

Int camera_num: associated camera index number.

Returns:

Serial number.

3.13 lir_GetFrameFloat

Description:

Exported routine to retrieve a frame of data (floating point format) from the desired camera.

Function Declaration:

UWord32 LTRONIMGIR lir_GetFrameFloat(ptr_img_GetFrameInfo pIO)

(See Appendix C for further information on Image I/O data structure)

Parameters:

ptr_img_GetFrameInfo pIO - pointer to I/O structure containing info about the camera.

Returns:

Non-zero (last frame read) if succesful.

3.14 lir_GetFrameShort

Description:

Exported routine to retrieve a frame of data (short format) from the desired camera. Temperature data is scaled (x10) in the short format to retain some level of precision. So if retrieving temperature data each of the values will need to be divided by 10 to get the actual temperature. Ex: a return value of 421 (degrees Fahrenheit) represents a temperature of 42.1.

Function Declaration:

UWord32 LTRONIMGIR lir_GetFrameShort(ptr_img_GetFrameInfo pIO)

(See Appendix C for further information on Image I/O data structure)

Parameters:

ptr_img_GetFrameInfo pIO - pointer to I/O structure containing info about the camera.

Returns:

Non-zero (last frame read) if succesful.

Example:

```
while (!bExitProgram)
{
      if (bOnline)
      {
            // Init the data structure before passing
            // it to the image dll
            imgInfo.nCamNum = nCamNum;
            imgInfo.nUnits = nConv2Temp;
            imgInfo.dEmissivity = dEmissivity;
            imgInfo.dBkGndTempInF = dBkgnTempInF;

            if (relative)
            {
                  sptr2 = &imgBuf3[0][0];
                  dptr = &imgBuf2[0][0];
                  endpoint = dptr + FRAME_PIXELS;
                  nOffset = (short)((dPalLow+dPalHigh)*5);

                  // Assign buffer
                  imgInfo.pOutShort = &imgBuf3[0][0];
                  // Get the data
                  nFrame = lir_GetFrameShort(&imgInfo);

                  sptr1 = (short *)imgBuf3;
                  sptr2 = (short *)imgBuf1;

                  if (sptr1 && sptr2)
                  {
                        // Development use
                        QueryPerformanceCounter(&PerfCounter1);

                        while (dptr<endpoint)
                        {
                              *dptr++ = *sptr2++ - *sptr1++;
                        }
                        pImg = &imgBuf2[0][0];
                        nFrame++;
                  }
            }
            else
            {
                  nOffset = 0;

                  // Now start each camera we need
                  for (i = CAM_NUM_0; i < MAX_NUM_CAMERAS; i++)
                  {
                        // If camera is active
                        if (nActCamDetect & (CAM_1_MASK << i))
                        {
                              // Save the camera index number
                              imgInfo.nCamNum = i;

                              switch (i)
                              {
                              default:
                              case (CAM_NUM_0):
                                    // Assign buffer
                                    imgInfo.pOutShort = &imgBuf2[0][0];
                                    break;

                              case (CAM_NUM_1):
                                    // Assign buffer
                                    imgInfo.pOutShort = &imgBuf4[0][0];
                                    break;
                              }
```

```
                        // Get the data
                        nFrame = lir_GetFrameShort(&imgInfo);
                    }
                }

                // Development use
                QueryPerformanceCounter(&PerfCounter1);
            }

            if (nFrame != lastframe)
            {
                lastframe = nFrame;

                ProcessZones();

                // Development use
                QueryPerformanceCounter(&PerfCounter2);

                i64ProcTime.QuadPart = PerfCounter2.QuadPart - PerfCounter1.QuadPart;

                nDispComplete = 0;

                if (vcron)
                    SetEvent(hVCREvent);

                if (!bDispBusy)
                    SetEvent(hDispEvent);
            }
            else
            {
                Sleep(10);
            }
        }
        else
        {
            // Development use
            QueryPerformanceCounter(&PerfCounter1);

            ProcessZones();

            // Development use
            QueryPerformanceCounter(&PerfCounter2);

            i64ProcTime.QuadPart = PerfCounter2.QuadPart - PerfCounter1.QuadPart;

            if (vcron)
            {
                // waits for next frame
                WaitForSingleObject(hVCREvent, 100);
                ResetEvent(hVCREvent);
                nFrame++;

                if ((nFrame & nFrmSkip) == 0)
                    SetEvent(hDispEvent);
            }
            else
            {
                nFrame++;
                Sleep(100);
                SetEvent(hDispEvent);
            }
        }
    }
```

3.15 lir_GetTemp

Description:

Exported routine to retrieve the desired camera sensor temperature or battery power. See camera documentation to determine which ADC port conatins the desired temperature/volts.

Function Declaration:

double LTRONIMGIR lir_GetTemp(int sensor, int camera_num)

Parameters:

int sensor - desired enumeration index FPA_TEMP, SPT_TEMP, REMOTE_0_TEMP, REMOTE_1_TEMP, BATTERY_PWR, CAL_FLAG_TEMP.

int camera_num - associated camera index number (default is 0).

Returns:

Associated temperature or voltage.

Example:

```
while (!bExitProgram)
{
	if (bOnline)
	{
		dRemote0Temp = lir_GetTemp(REMOTE_0_TEMP, nCamNum);
		dRemote1Temp = lir_GetTemp(REMOTE_1_TEMP, nCamNum);

		dFpaTemp = lir_GetTemp(FPA_TEMP, nCamNum);
		dSptTemp = lir_GetTemp(SPT_TEMP, nCamNum);

		dBatteryPwr = lir_GetTemp(BATTERY_PWR, nCamNum);
	}

	Sleep(100);
}
```

4 LtronSptIr Library Function Reference

The functions listed in this section make up the interface for the LtronSptIR.dll library. Use of these functions does not require access to the NI imaq.dll library.

4.1 lir_Plank

Description:

Exported routine to compute the Plank value from the supplied terms.

Function Declaration:

double LTRONSPTIR lir_Plank(double x1, double x2, double PW, double TFahren)

Parameters:

double x1 - low pass wavelength.

double x2 - high pass wavelength.

double PW - photons or watts power factor.

double TFahren - temperature (degrees Fahrenheit).

Returns:

Computed Planks value.

4.2 lir_BuildConversionTableShort

Description:

Exported routine to build a temperature look up table (short format) from the information supplied so that the host application can convert stored ADC count data to temperature values.

Function Declaration:

void LTRONSPTIR lir_BuildConversionTableShort(ptr_img_ConversionInfo pConv, Word16* pLUT)

Parameters:

ptr_img_ConversionInfo pConv - pointer to structure containing data conversion information.

Word16* pLUT - pointer to a user supplied buffer that will be filled with temperature look up values.

(See Appendix D for further information on the conversion info structure)

Returns:

None

Example:

```
img_Header              sHdrInfo = {0};
img_ConversionInfo      sConvInfo = {0};

// Create LUT buffer
pMyLut = new Word16[LUT_SIZE];

pfMyLut = new float[LUT_SIZE];

// Get header data (calibration data)
lir_GetHeader(&sHdrInfo, 0);

// Get camera info to fill conversion data structure
```

```
if (sHdrInfo.sRadParams.highWave != 0.)
{
      sConvInfo.dLoWave = sHdrInfo.sRadParams.lowWave;
      sConvInfo.dHiWave = sHdrInfo.sRadParams.highWave;
      sConvInfo.dZSCounts = sHdrInfo.sRadParams.calZeroCount;
      sConvInfo.dZSTemp = sHdrInfo.sRadParams.calZeroTemp;
      sConvInfo.dFSCounts = sHdrInfo.sRadParams.calHotCount;
      sConvInfo.dFSTemp = sHdrInfo.sRadParams.calHotTemp;
}

// Call routine to build custom LUT
if (pMyLut)
{
      sConvInfo.dBkGndTemp = 77.;
      sConvInfo.dEmissivity = 1.0;
      sConvInfo.nUnits = UNITS_F;

      lir_BuildConversionTableShort(&sConvInfo, pMyLut);

      lir_BuildConversionTableFloat(&sConvInfo, pfMyLut);

      // Examine temp LUTs here
      dbgValA = pMyLut[3756];
      dbgValB = pfMyLut[3756];
};
```

4.3 lir_BuildConversionTableFloat

Description:

Exported routine to build a temperature look up table (float format) from the information supplied so that the host application can convert stored ADC count data to temperature values.

Function Declaration:

void LTRONSPTIR lir_BuildConversionTableShort(ptr_img_ConversionInfo pConv, Word16* pLUT)

Parameters:

ptr_img_ConversionInfo pConv - pointer to structure containing data conversion information.

Word16* pLUT - pointer to a user supplied buffer that will be filled with temperature look up values.

(See Appendix D for further information on the conversion info structure)

Returns:

None

Example:

```
img_Header              sHdrInfo = {0};
img_ConversionInfo      sConvInfo = {0};

// Create LUT buffer
pMyLut = new Word16[LUT_SIZE];

pfMyLut = new float[LUT_SIZE];

// Get header data (calibration data)
lir_GetHeader(&sHdrInfo, 0);

// Get camera info to fill conversion data structure
if (sHdrInfo.sRadParams.highWave != 0.)
{
      sConvInfo.dLoWave = sHdrInfo.sRadParams.lowWave;
      sConvInfo.dHiWave = sHdrInfo.sRadParams.highWave;
      sConvInfo.dZSCounts = sHdrInfo.sRadParams.calZeroCount;
```

```
        sConvInfo.dZSTemp = sHdrInfo.sRadParams.calZeroTemp;
        sConvInfo.dFSCounts = sHdrInfo.sRadParams.calHotCount;
        sConvInfo.dFSTemp = sHdrInfo.sRadParams.calHotTemp;
}

// Call routine to build custom LUT
if (pMyLut)
{
        sConvInfo.dBkGndTemp = 77.;
        sConvInfo.dEmissivity = 1.0;
        sConvInfo.nUnits = UNITS_F;

        lir_BuildConversionTableShort(&sConvInfo, pMyLut);

        lir_BuildConversionTableFloat(&sConvInfo, pfMyLut);

        // Examine temp LUTs here
        dbgValA = pMyLut[3756];
        dbgValB = pfMyLut[3756];
}
```

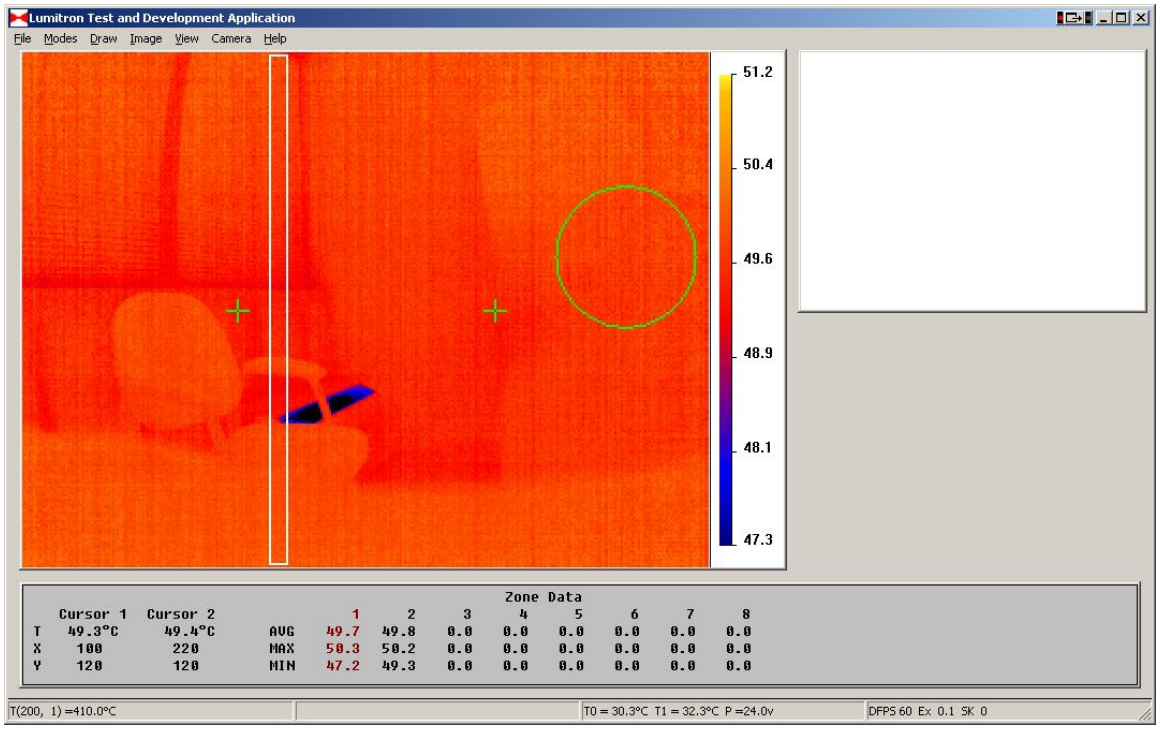
5 LtronView Sample Application

The sample application that is provided as part of P/N 600-00000030 was developed to test, debug, and optimize the interface libraries. LtronView is a multithreaded application that can acquire and display imagery from up to two cameras, while processing radiometric data from one of those cameras. This application provides just one of many solutions in which a host application might acquire and process image data.

NOTE: Some portions of the code in the LtronView application are not called or may be obsolete.

5.1 Application Boot

Upon boot - a main application interface is created. Inside the main window are two image display areas and an area for zone/cursor information. Next to the larger of the two image display windows is an area for mapping the active color palette to corresponding temperatures/ADC counts. The image below shows the application running with just a single camera attached.



Next the application will attempt to start acquiring data from two cameras. The configuration files are hard coded into the application and can be modified by changing the following lines:

```
// Set Interface ID for the NI Board
lir_SetInterfaceID("LTron_240 1426.iid", CAM_NUM_0);
lir_SetInterfaceID("LTron_240 1422.iid", CAM_NUM_1);
```

The application will work with only one camera installed.

After setting the camera files, the application will create a default palette, set up some program defaults, and then check to see if a configuration file exists. The application configuration file stores the zones, emissivity, temperature range, zoom factor, and other aspects of the application when it was previously opened.

The final stage of the boot process is to start the threads that handle the acquisition, display and data processing. Each of these threads performs a specific task and its core functions are either performed in a loop while the program is active and camera(s) are online or triggered by an application event.

5.2 Get Acquisition Data Thread

The 'GetAcqDataThread' routine is responsible for getting image data from the LtronImgIr library into the host application. It starts the camera acquisition process and then as necessary copies the data digital data from the library buffers (using lir_GetFrameShort) into its local memory for processing.

Below is a copy of the thread routine. Some portions have been removed to save space and readability – refer to source code for complete routine.

```
/*****************************************************************************/
UWord32 WINAPI GetAcqDataThread(LPVOID pDummy)
{
   img_GetFrameInfo  imgInfo = {0};

   LARGE_INTEGER     PerfCounter1;
   LARGE_INTEGER     PerfCounter2;

   DWORD             lastframe = 0;

   short             *sptr1,
                     *sptr2,
                     *dptr,
                     *endpoint;

   int               nActCamDetect = CAM_MASK_NONE,
                     i;

   bool              bFirstCam = false;

   SetThreadPriority(hGetDataThread, THREAD_PRIORITY_HIGHEST);

   // Development use
   QueryPerformanceCounter(&PerfCounter1);

   pImg = &imgBuf2[0][0];
   pImg = &imgBuf4[0][0];

   // Init baseline values to no cameras
   nCamMax = nCamNum = 0;

   /***** First detect which cameras are active *****/

   // Test if camera 0 is present
   if (lir_CheckActive(CAM_NUM_0))
   {
         // Add to mask
         nActCamDetect |= CAM_1_MASK;
         // Increment max counter
         nCamMax++;
   }

   // Test if camera 1 is present
   if (lir_CheckActive(CAM_NUM_1))
   {
         // Add to mask
         nActCamDetect |= CAM_2_MASK;
         // Increment max counter
         nCamMax++;
   }

   /***** Now we know which cameras are active *****/
```

```
// If we have at least one camera
if (nCamMax > 0)
{
      // Now start each camera we need
      for (i = CAM_NUM_0; i < MAX_NUM_CAMERAS; i++)
      {
            // If camera is active
            if (nActCamDetect & (CAM_1_MASK << i))
            {
                  // Attempt to start camera
                  if (lir_StartCamera(i))
                  {
                        // If this is the first camera started
                        if (!bFirstCam)
                        {
                              // Set the main window camera index
                              nCamNum = i;
                              // Set flag
                              bFirstCam = true;
                        }
                  }
                  // Camera failed to start
                  else
                  {
                        // Clear the mask
                        nActCamDetect &= ~(CAM_1_MASK << i);
                        // Decrement counter
                        nCamMax--;
                  }
            }
      }
}

// See if any cameras started
if (nCamMax > 0)
      bOnline = true;

if (bOnline)
{
      CheckMenuItem(GetMenu(hWndMain), IDM_ONLINE, MF_CHECKED);
}

while (!bExitProgram)
{
      if (bOnline)
      {
            // Init the data structure before passing
            // it to the image dll
            imgInfo.nCamNum = nCamNum;
            imgInfo.nUnits = nConv2Temp;
            imgInfo.dEmissivity = dEmissivity;
            imgInfo.dBkGndTempInF = dBkgnTempInF;

            if (relative)
            {
                  …
            }
            else
            {
                  nOffset = 0;

                  // Now start each camera we need
                  for (i = CAM_NUM_0; i < MAX_NUM_CAMERAS; i++)
                  {
                        // If camera is active
                        if (nActCamDetect & (CAM_1_MASK << i))
                        {
                              // Save the camera index number
```

```
                        imgInfo.nCamNum = i;

                        switch (i)
                        {
                        default:
                        case (CAM_NUM_0):
                              // Assign buffer
                              imgInfo.pOutShort = &imgBuf2[0][0];
                              break;

                        case (CAM_NUM_1):
                              // Assign buffer
                              imgInfo.pOutShort = &imgBuf4[0][0];
                              break;
                        }

                        // Get the data
                        nFrame = lir_GetFrameShort(&imgInfo);
                  }
            }

            // Development use
            QueryPerformanceCounter(&PerfCounter1);
      }

      if (nFrame != lastframe)
      {
            lastframe = nFrame;

            ProcessZones();

            // Development use
            QueryPerformanceCounter(&PerfCounter2);

            i64ProcTime.QuadPart = PerfCounter2.QuadPart - PerfCounter1.QuadPart;

            nDispComplete = 0;

            if (vcron)
                  SetEvent(hVCREvent);

            if (!bDispBusy)
                  SetEvent(hDispEvent);
      }
      else
      {
            Sleep(10);
      }
}
else
{
      // Development use
      QueryPerformanceCounter(&PerfCounter1);

      ProcessZones();

      // Development use
      QueryPerformanceCounter(&PerfCounter2);

      i64ProcTime.QuadPart = PerfCounter2.QuadPart - PerfCounter1.QuadPart;

      if (vcron)
      {
            …
      }
      else
      {
            nFrame++;
```

```
                    Sleep(100);
                    SetEvent(hDispEvent);
                }
            }
        }

        return (GET_DATA_THREAD_RETVAL);
    }
```

5.3 Display Acquisition Data Thread

The 'DisplayAcqDataThread' routine is responsible for displaying the image data that was copied by the acquisition thread. It is triggered by an event (hDispEvent) that is set in the 'GetAcqDataThread' routine when a new frame of data is ready. It also handles updating the zone objects (cursors, rectangles, etc.).

Below is a copy of the thread routine. Some portions have been removed to save space and readability – refer to source code for complete routine.

```
/*****************************************************************************/
UWord32 WINAPI DisplayAcqDataThread(LPVOID pDummy)
{
    LARGE_INTEGER     PerfCounter1,
                      PerfCounter2;

    TCHAR  sbuf[30],
          *pix,
          *pix2;

    DWORD *pPix,
          *pPix2;

    short *ptrImg,
          *ptrImg2,
          *endImg,
          *endImg2;

    long  dispcnt = 0;

    int   i;

    HDC   lclhDCImg,
          lclhDCImg2;

    SetThreadPriority(
          hDisplayThread,
          THREAD_PRIORITY_ABOVE_NORMAL
    );

    …

    clkTimeStart = GetTickCount();

    lclhDCImg = GetDC(hWndImg);
    lclhDCImg2 = GetDC(hWndImg2);

    // While program is running
    while (!bExitProgram)
    {
          // Wait until frame is ready
          WaitForSingleObject(hDispEvent, 1500);
          bDispBusy = true;

          …

          if (nPalRefresh)
          {
```

```
	nPalRefresh = 0;
	DisplayPalette();
	Sleep(0);
}

if (bRecalcZone)
{
	bRecalcZone = false;
	FindZone();
}

if (nPixDepth < 24)
{
	…
}
else
{
	// Hard coded for 2 images
	ptrImg = &imgBuf2[0][0];
	pPix = (DWORD *)pBitsImg;
	endImg = ptrImg + FRAME_PIXELS;

	ptrImg2 = &imgBuf4[0][0];
	pPix2 = (DWORD *)pBitsImg2;
	endImg2 = ptrImg2 + FRAME_PIXELS;

	if (relative)
	{
		while (ptrImg < endImg)
		{
			*pPix++ = clrFastLUT[(*ptrImg++) + nOffset];
		}
	}
	else
	{
		while (ptrImg < endImg)
		{
			*pPix++ = clrFastLUT[(*ptrImg++)];

			*pPix2++ = clrFastLUT[(*ptrImg2++)];
		}
	}
}

// Clear trigger
ResetEvent(hDispEvent);

if (bShowAll)
{
	SelectObject(hDCMemImg, hGPen);

	for (i = 0; i < MAX_ZONES; i++)
	{
		if (i!= nZoneNum)
			Polyline(hDCMemImg, &aZonePts[i][0], aZone1Pts[i]);
	}

	MoveToEx(hDCMemImg, nHorPtB - 5, nVerPtB, NULL);
	LineTo(hDCMemImg, nHorPtB, nVerPtB);

	MoveToEx(hDCMemImg, nHorPtB + 1, nVerPtB, NULL);
	LineTo(hDCMemImg, nHorPtB + 6, nVerPtB);

	MoveToEx(hDCMemImg, nHorPtB, nVerPtB - 5, NULL);
	LineTo(hDCMemImg, nHorPtB, nVerPtB);

	MoveToEx(hDCMemImg, nHorPtB, nVerPtB + 1, NULL);
	LineTo(hDCMemImg, nHorPtB, nVerPtB + 6);
```

```
      MoveToEx(hDCMemImg, nHorPtA - 5, nVerPtA, NULL);
      LineTo(hDCMemImg, nHorPtA, nVerPtA);

      MoveToEx(hDCMemImg, nHorPtA + 1, nVerPtA, NULL);
      LineTo(hDCMemImg, nHorPtA + 6, nVerPtA);

      MoveToEx(hDCMemImg, nHorPtA, nVerPtA - 5, NULL);
      LineTo(hDCMemImg, nHorPtA, nVerPtA);

      MoveToEx(hDCMemImg, nHorPtA, nVerPtA + 1, NULL);
      LineTo(hDCMemImg, nHorPtA, nVerPtA + 6);
}

SelectObject(hDCMemImg, GetStockObject(WHITE_PEN));
SelectObject(hDCMemImg, GetStockObject(WHITE_BRUSH));

// Reticle/Cursor 1
if (nZoneNum == 10)
{
      MoveToEx(hDCMemImg, nHorPtA - 5, nVerPtA, NULL);
      LineTo(hDCMemImg, nHorPtA, nVerPtA);
      MoveToEx(hDCMemImg, nHorPtA + 1, nVerPtA, NULL);
      LineTo(hDCMemImg, nHorPtA + 6, nVerPtA);
      MoveToEx(hDCMemImg, nHorPtA, nVerPtA - 5, NULL);
      LineTo(hDCMemImg, nHorPtA, nVerPtA);
      MoveToEx(hDCMemImg, nHorPtA, nVerPtA + 1, NULL);
      LineTo(hDCMemImg, nHorPtA, nVerPtA + 6);
}

// Reticle/Cursor 2
if (nZoneNum == 11)
{
      MoveToEx(hDCMemImg, nHorPtB - 5, nVerPtB, NULL);
      LineTo(hDCMemImg, nHorPtB, nVerPtB);
      MoveToEx(hDCMemImg, nHorPtB + 1, nVerPtB, NULL);
      LineTo(hDCMemImg, nHorPtB + 6, nVerPtB);
      MoveToEx(hDCMemImg, nHorPtB, nVerPtB - 5, NULL);
      LineTo(hDCMemImg, nHorPtB, nVerPtB);
      MoveToEx(hDCMemImg, nHorPtB, nVerPtB + 1, NULL);
      LineTo(hDCMemImg, nHorPtB, nVerPtB + 6);
}

// Draws selected zone (in white)
Polyline(hDCMemImg, &aZonePts[nZoneNum][0], aZone1Pts[nZoneNum]);

// Paints large image window
StretchBlt(
      lclhDCImg,
      0,
      0,
      rectImgWnd.right - 70,
      rectImgWnd.bottom,
      hDCMemImg,
      nXZoom,
      nYZoom,
      FRAME_WIDTH/nZoom,
      FRAME_HEIGHT/nZoom,
      SRCCOPY
);

// Paints small image window
StretchBlt(
      lclhDCImg2,
      0,
      0,
      rectImg2Wnd.right,
      rectImg2Wnd.bottom,
```

```
                hDCMemImg2,
                0,
                0,
                FRAME_WIDTH,
                FRAME_HEIGHT,
                SRCCOPY
          );

          nDispComplete = -1;

          …
    }

    ReleaseDC(hWndImg, lclhDCImg);
    ReleaseDC(hWndImg2, lclhDCImg2);

    return (DISPLAY_THREAD_RETVAL);
}
```

5.4 Display Zone Information Thread

The 'DisplayZoneInfoThread' routine is responsible for updating the zone reporting values. The routine is loop based and runs continually until the program exits.

Below is a copy of the thread routine. Some portions have been removed to save space and readability – refer to source code for complete routine.

```
/*****************************************************************************/
UWord32 WINAPI DisplayZoneInfoThread(LPVOID pDummy)
{
    TCHAR   sbuf[100], wbuf[20];
    HDC     hdcZ;
    HFONT   hFont;
    int     i, scale;
    char    units;
    double  Period = 0;

    hdcZ = GetDC(hWndZoneData);

    SetBkColor( hdcZ, RGB(192, 192, 192));
    SetTextColor( hdcZ, RGB(0, 0, 0));

    hFont = CreateFont(
          16,
          8,
          2,
          2,
          FW_NORMAL,
          FALSE,
          FALSE,
          FALSE,
          ANSI_CHARSET,
          OUT_DEFAULT_PRECIS,
          CLIP_DEFAULT_PRECIS,
          PROOF_QUALITY,
          FIXED_PITCH | FF_MODERN,
          "System"
    );

    SelectObject(hdcZ, hFont);

    while (!bExitProgram)
    {
          Sleep(200);
          scale = 10;

          if (nConv2Temp == UNITS_C)
```

```
                units = 'C';
        else if (nConv2Temp == UNITS_F)
                units = 'F';
        else
        {
                units = ' ';
                scale = 1;
        }

        if (nZoneNum == 10)
                SetTextColor( hdcZ, RGB(128, 0, 0));
        else
                SetTextColor( hdcZ, RGB(0, 0, 0));

        sprintf(sbuf, "   Cursor 1");
        TextOut(hdcZ, 10, 20, sbuf, strlen(sbuf));

        if (nConv2Temp)
                sprintf(sbuf, "T   %4.1f°%c  ", (double)(imgBuf2[nVerPtA][nHorPtA])/scale,
units);
        else
                sprintf(sbuf, "T   %4.0f     ", (double)(imgBuf2[nVerPtA][nHorPtA])/scale,
units);

        TextOut(hdcZ, 10, 36, sbuf, strlen(sbuf));
        sprintf(sbuf, "X   %4d  ", nHorPtA);
        TextOut(hdcZ, 10, 52, sbuf, strlen(sbuf));
        sprintf(sbuf, "Y   %4d  ", nVerPtA);
        TextOut(hdcZ, 10, 68, sbuf, strlen(sbuf));

        if (nZoneNum == 11)
                SetTextColor( hdcZ, RGB(128, 0, 0));
        else
                SetTextColor( hdcZ, RGB(0, 0, 0));

        sprintf(sbuf, "  Cursor 2");
        TextOut(hdcZ, 100, 20, sbuf, strlen(sbuf));

        if (nConv2Temp)
                sprintf(sbuf, "    %4.1f°%c  ", (double)(imgBuf2[nVerPtB][nHorPtB])/scale,
units);
        else
                sprintf(sbuf, "    %4.0f     ", (double)(imgBuf2[nVerPtB][nHorPtB])/scale,
units);

        TextOut(hdcZ, 100, 36, sbuf, strlen(sbuf));
        sprintf(sbuf, "    %4d  ", nHorPtB);
        TextOut(hdcZ, 100, 52, sbuf, strlen(sbuf));
        sprintf(sbuf, "    %4d  ", nVerPtB);
        TextOut(hdcZ, 100, 68, sbuf, strlen(sbuf));

        SetTextColor( hdcZ, RGB(0, 0, 0));
        sprintf(sbuf, "Zone Data");
        TextOut(hdcZ, 450, 4, sbuf, strlen(sbuf));
        sprintf(sbuf, "AVG  ");
        TextOut(hdcZ, 230, 36, sbuf, strlen(sbuf));
        sprintf(sbuf, "MAX  ");
        TextOut(hdcZ, 230, 52, sbuf, strlen(sbuf));
        sprintf(sbuf, "MIN  ");
        TextOut(hdcZ, 230, 68, sbuf, strlen(sbuf));

        if (nConv2Temp)
        {
                for (i = 0; i < MAX_ZONES; i++)
                {
                        if (i == nZoneNum)
                                SetTextColor( hdcZ, RGB(128, 0, 0));
                        else
```

```
                        SetTextColor( hdcZ, RGB(0, 0, 0));

                  sprintf(sbuf, "%5d", i+1);
                  TextOut(hdcZ, 275+i*52, 20, sbuf, strlen(sbuf));
                  sprintf(sbuf, "%5.1f  ", dZoneAvg[i]);
                  TextOut(hdcZ, 275+i*52, 36, sbuf, strlen(sbuf));
                  sprintf(sbuf, "%5.1f  ", dZoneMax[i]);
                  TextOut(hdcZ, 275+i*52, 52, sbuf, strlen(sbuf));
                  sprintf(sbuf, "%5.1f  ", dZoneMin[i]);
                  TextOut(hdcZ, 275+i*52, 68, sbuf, strlen(sbuf));
            }
      }
      else
      {
            for (i = 0; i < MAX_ZONES; i++)
            {
                  if (i == nZoneNum)
                        SetTextColor( hdcZ, RGB(128, 0, 0));
                  else
                        SetTextColor( hdcZ, RGB(0, 0, 0));

                  sprintf(sbuf, "%5d", i+1);
                  TextOut(hdcZ, 275+i*52, 20, sbuf, strlen(sbuf));
                  sprintf(sbuf, "%5.0f  ", dZoneAvg[i]);
                  TextOut(hdcZ, 275+i*52, 36, sbuf, strlen(sbuf));
                  sprintf(sbuf, "%5.0f  ", dZoneMax[i]);
                  TextOut(hdcZ, 275+i*52, 52, sbuf, strlen(sbuf));
                  sprintf(sbuf, "%5.0f  ", dZoneMin[i]);
                  TextOut(hdcZ, 275+i*52, 68, sbuf, strlen(sbuf));
            }
      }

      // Debug Info
      sprintf(sbuf, "DFPS %2d", nFrmPerSec);

      nSkip = 0;

      lir_GetCameraInfo(&sCamInfo, nCamNum);
      sprintf(wbuf, "%I64d  ", i64ProcTime);
      Period = (3*Period + atof(wbuf))/4;
      sprintf(sbuf+strlen(sbuf), "  Ex %4.1f", Period/dTimeBase + sCamInfo.TransferTime);

      sprintf(sbuf+strlen(sbuf), "  SK %2d", nSkip);
      SendMessage (hWndStatus, SB_SETTEXT, 3, (LPARAM)sbuf);

      sprintf(sbuf, "T0 =%5.1f°C  T1 =%5.1f°C  P =%4.1fv ", dRemote0Temp, dRemote1Temp,
dBatteryPwr);
      SendMessage (hWndStatus, SB_SETTEXT, 2, (LPARAM)sbuf);
  }

  DeleteObject(hFont);

  return (ZONE_THREAD_RETVAL);
}
```

5.5 Get Temperature Sensor Thread

The 'GetTempSensorThread' routine is responsible for monitoring temperature and power sensors inside the camera. It is not necessary to perform this process and it is being shown for reference only. The routine is loop based and runs continually until the program exits or is offline.

Below is a copy of the thread routine. Some portions have been removed to save space and readability – refer to source code for complete routine.

```
/*****************************************************************************/
UWord32 WINAPI GetTempSensorThread(LPVOID pDummy)
{
```

```
    Sleep(100);

    while (!bExitProgram)
    {
        if (bOnline)
        {
            dRemote0Temp = lir_GetTemp(REMOTE_0_TEMP, nCamNum);
            dRemote1Temp = lir_GetTemp(REMOTE_1_TEMP, nCamNum);

            dFpaTemp = lir_GetTemp(FPA_TEMP, nCamNum);
            dSptTemp = lir_GetTemp(SPT_TEMP, nCamNum);

            dBatteryPwr = lir_GetTemp(BATTERY_PWR, nCamNum);
        }

        Sleep(100);
    }

    return (TEMP_SENSOR_THREAD_RETVAL);
}
```

Appendix A - Header Data Structures

```
/* Data Structure for storage of key header information */
struct _img_Header
{
      bool  bCamAct;                // true if camera is active
      bool  bCamPolling;            // true if camera in polling mode (acquiring data)

      UWord16     nCamNum;          // Camera index (0 - 3)
      UWord16     nCamType;         // Camera Type (currently not used)
      UWord16     nNucTable;        // Current NUC table index

      UWord32     nSerialNum;       // Camera serial number
      UWord32     nFrameNum;        // Current frame number
      UWord32     nDataSize;        // Size of image data

      rad_IrconParams   sRadParams; // Structure containing radiometric calibration
                                    // parameters

      rad_IrconTCData   sTC;        // Currently not used - placeholder only

      adc_Temp sAdc;                // Camera ADC values (currently not used)

      char  reserved[64];
};
typedef struct _img_Header img_Header, *ptr_img_Header;

/* Radiometric Parameters for the Ircon Temperature Calculation Algorithm */
struct _rad_IrconParams
{
      float calZeroTemp;      // Low point calibration in temperature
      float calZeroCount;     // Low point calibration in counts
      float calHotTemp;       // High point calibration in temperature
      float calHotCount;      // High point calibration in counts
      float lowWave;          // Cut On window/filter transmittance wavelength
      float highWave;         // Cut Off window/filter transmittance wavelength
      float minTemp;          // Minimum temperature in calculated range
      float maxTemp;          // Maximum temperature in calculated range
};
typedef struct _rad_IrconParams rad_IrconParams, *ptr_rad_IrconParams;
```

Appendix B – Camera Info Structure

```
/* Data Structure to hold camera specific information */
struct _img_CameraInfo
{
      UWord16     nNucTable;              // Index of camera's active NUC table
      UWord16     Heighth;                // Camera image height
      UWord16     Width;                  // Camera image width

      double      MinTemp;                // Minimum temperature in calculated range
      double      MaxTemp;                // Maximum temperature in calculated range
      double      VideoZero;              // Low calibration temperature
      double      VideoFull;              // High calibration temperature
      double      LoWave;                 // Cut On window/filter transmittance wavelength
      double      HiWave;                 // Cut Off window/filter transmittance wavelength

      double      FrameRateAcquired;      // Currently not used
      double      FrameRateDeliverd;      // Currently not used

      double      DLLVersion;             // DLL version data
      double      TransferTime;           // Currently not used
};
typedef struct _img_CameraInfo img_CameraInfo, *ptr_img_CameraInfo;
```

Appendix C - Frame I/O Structure

```
// Structure to pass information for acquiring frame data
struct _img_GetFrameInfo
{
      int         nCamNum;               // Desired camera (0 - 3)
      int         nUnits;                // Desired buffer units (A/D, F, C)
      int         nReserved[6];

      double      dEmissivity;           // Image emissivity
      double      dBkGndTempInF;         // Image Background Temp
      double      dReserved[2];

      short*      pOutShort;             // Pointer to output buffer (short)
      float*      pOutFloat;             // Pointer to output buffer (float)
};
typedef struct _img_GetFrameInfo img_GetFrameInfo, *ptr_img_GetFrameInfo;
```

Appendix D – Conversion I/O Structure

```
// Structure to pass information for converting frame data or
// building a temperature look up table.
//
// Note:
//          To get the proper conversion these values should match those
//          used to create the temperature values initially.  Most of
//          which should come from the camera instrumentation header.
struct _img_ConversionInfo
{
      int         nUnits;           // See "_eDisplayUnits"

      double      dEmissivity;
      double      dBkGndTemp;       // In degress Fahrenheit

      double      dLoWave;
      double      dHiWave;
      double      dZSTemp;          // In degress Fahrenheit
      double      dZSCounts;
      double      dFSTemp;          // In degress Fahrenheit
      double      dFSCounts;
};
typedef struct _img_ConversionInfo img_ConversionInfo, *ptr_img_ConversionInfo;
```