SNAPPER Technical Notes

DataCell Limited

v4.0.1

Disclaimer

Copyright Notice

Trademarks

“Apple”, “Macintosh” and “MacOS” are trademarks of Apple Computer Inc. “AMCC” is a registered trademark of Applied Micro Circuits Corporation. “Dallas” is a registered trademark of Dallas Semiconductor Corporation. “Dell” is a registered trademark of Dell Computer Corporation. “Flash Graphics” and “X-32VM” are trademarks of Flashtek Limited. “IBM”, “PC/AT”, “PowerPC” and “VGA” are registered trademarks of International Business Machine Corporation. “MetroWerks” and “CodeWarrior” are registered trademarks of MetroWerks Inc. “Microsoft”, “CodeView”, “MS” and “MS-DOS”, “Windows”, “Windows NT”, “Windows 95”, “Windows 98”, “Win32”, “Visual C++” are trademarks or registered trademarks of Microsoft Corporation. “National Semiconductor” is a registered trademark of National Semiconductor Corporation. “Sun”, “Ultra AX” and “Solaris” are registered trademarks of Sun Microsystems Inc. All “SPARC” trademarks are trademarks or registered trademarks of SPARC International Inc. “VxWorks” and “Tornado” are registered trademarks of Wind River Systems Inc. “Xilinx” is a registered trademark of Xilinx.
All other trademarks and registered trademarks are the property of their respective owners.

Part Information

Part Number: SNP-TECHNOTES

Version v4.0.1 September 1999

Printed in the United Kingdom.

Contact Details

Europe & ROW	Web	www.datacell.co.uk
	Sales	info@datacell.co.uk
	Support	techsupport@datacell.co.uk
USA	Web	www.datacell.com
	Sales	info@datacell.com
	Support	techsupport@datacell.com

Head Office:
DataCell Limited.
Falcon Business Park, 40 Ivanhoe Road,
Finchampstead, Berkshire, RG40 4QQ, UK

Tel +44 (0) 1189 324324
Fax +44 (0) 1189 324325

Technical Notes

- Snapper-16 and Crunch-ISA: Q & A
- Snapper-16 and Crunch-ISA: Compression Speeds Explained
- Snapper-16: Switching Cameras and Achieving Rapid Locking
- Snapper-24: Level Controls
- Snapper-Dig16: Porting New Cameras
- Snapper-8 & Snapper-24: Vertical Banding
- Snapper-8 & Snapper-24: Achieving Very Stable Sampling
- Snapper-SDK for MS-Dos: Minimizing Real Mode Memory Usage
- Snapper-Dig16: Line Scan Support
- Snapper-SDK for Windows NT: Interrupts and Callback Functions

SNAPPER!

Snapper-16 and Crunch-ISA: Q & A

v1.3 March 1999 TECHNICAL NOTE 1

This technical note provides some answers to common questions regarding hardware JPEG compression using the ISA JPEG Bus Interface Board ("Crunch-ISA") and Snapper-16.

Q: *What types of image formats can be compressed?*

A: The CRUNCH library accepts TIFF, TARGA and BMP files. For TIFF this includes 8 bit grayscale, paletted and 24 bit colour files. For TARGA files this includes 8 bit grayscale, 15/16 bit colour and 24 bit colour files. For BMP this includes paletted (including grayscale) and 24 bit colour files.

The compression hardware itself accepts image data in a raw raster format - For grayscale each pixel is represented by one byte, with 0 as black and 255 as white. For colour images, the raw data must be in 24 bit RGB format or 16 bit YCbCr format. Other RGB formats such as 1 line of red, then green etc can be accommodated easily in software.

Q: *What format is the compressed data saved in?*

A: Compressed data is saved as a JPEG file. This is a universal format as defined in the JPEG specification and can be decompressed by compatible hardware or software decompressors. There is also limited support for JFIF (JPEG File Interchange Format) - this format includes extra markers such as comment strings (which are supported by CRUNCH) etc. Any JPEG reader should be able to read in a JFIF file and vice-versa.

Q: *Can the compressed image be saved to host memory or directly to file?*

A: Yes the compressed data can be saved in the PC's host memory or can be streamed directly to file, selectable by one parameter to the compression functions.

Q: *What is the speed of compression for a 640 x 480 colour and monochrome image and how is this calculated?*

A: When compressing a video image directly from Snapper-16, the speed of compression is very fast - as a rough indication, you can expect around 10Mbytes/sec, probably faster depending on the complexity of the image and the compression ratio. Thus a colour image can be compressed from Snapper-16 in around 60ms and a monochrome (grayscale) image in about 40ms.

When compressing from disk across the ISA bus (i.e. not from Snapper but from a file on the hard disk), the speed is limited by the speed of the ISA bus. As a conservative estimate, colour images can be compressed at about 1 Mbyte per second and grayscale images at 2 Mbytes per second. In practice it is possible to achieve about double this throughput depending on the type of PC in use. Using the conservative figures, these means that a 640 x 480 x 24 bit image would take about 900ms and a 640 x 480 x 8 bit image about 300ms.

Q: *What type of image formats are supported when the JPEG file is decompressed?*

A: TIFF, TARGA and BMP formats are supported. In detail:

TIFF	24 bit colour (RGB), 8 bit colour/grayscale paletted, 8 bit grayscale. Uncompressed data format only.
TARGA	24 bit colour (RGB), 15/16 bit colour, 8 bit grayscale. Uncompressed data format only.
BMP	24 bit colour (RGB), 8 bit colour/grayscale paletted. Uncompressed data format only.

Q: *Precisely what type of TIFF files does the CRUNCH software support?*

A: First of all the TIFF reader should be able to read any TIFF file as long as the data within it is not compressed.

As for the TIFF format of the files written - they are fairly straight forward using a minimum set of tags which are written out at the start of the file. The image data is written as one strip. For further information, please refer to the TIFF specification. This can be obtained from Aldus Corporation in the US on (206) 628 6593 and Aldus Europe Limited in the UK on 0131 453 2211.

Q: *Typically what speeds can be achieved recording a sequence of colour JPEG files to disk - perhaps for a security application?*

A: Typical speeds using a 486DX2-33 and an IDE drive are as follows:

Full resolution (768 x 576 or 640 x 480)	6 Hz
Single fields (768 x 576 or 640 x 480)	9 Hz
Half resolution (384 x 288 or 320 x 240)	12.5/15 Hz
Quarter resolution (192 x 144 or 144 x 140)	25/30 Hz

Grayscale acquisition rates are 8Hz, 12.5/15Hz, 25/30Hz and 25/30Hz respectively.

SNAPPER!

Snapper-16 and Crunch-ISA: Compression Speeds Explained

v1.2 March 1999 TECHNICAL NOTE 2

INTRODUCTION

This technical note attempts to explain all the issues related to the speed of compression and decompression using the ISA JPEG Bus Interface Board (“Crunch-ISA”) and Snapper-16.

JPEG PROCESSOR SPEED

CRUNCH uses the 30MHz version of C-Cube’s CL550 JPEG chip. This part can generate (or accept) compressed data at a rate of about 1.5 Mbytes per second. This rate is constant. Thus the rate at which raw data is sucked in (during compression) or generated in decompression, is a function of the compression ratio, which itself is a function of the complexity of the image. For example a pure tone image - such as white or mid-gray will have a compression ratio of around 80 to 1. Thus raw data is processed at the maximum rate of 30 Mbytes per second. For typical images using the default Q factor, compression ratios are around 10 to 1, often slightly more (15 to 1). So for example a typical video image (640 x 480) will get processed at a rate of 15 Mbytes per second. Thus a theoretical 20ms for monochrome and 40ms for colour. Now in practice, because the image content varies (and thus the compression ratio), images will take typically 30% longer than this. This is because of certain parts of the image not compressing as well - thus causing the pixel processing to stall.

ISA BUS / SNAPPER-16 INTERFACE

Although the C-Cube chip can process at the high data rates discussed above, the ISA bus is relatively slow, typically running at 1 to 2 Mbytes per second. This varies between machines, dependent on the ISA bus chipset used in the PC. CRUNCH uses the special assembler instructions, *outsw* and *insw* to achieve optimum performance in the PC environment. This can give data rates of up to 4 Mbytes per second in some PCs. Note the ISA bus speed is sometimes set via the PC’s BIOS setup under an option called “I/O Bus Recovery Time” - if so this should be set to a minimum - usually “2 BCLKS”.

Thus when processing a raw file from disk, the speed of compression is determined by the ISA bus. The utilities *comp* and *decomp* include overall timings which includes file I/O as well as the processing time. Thus the processing rate is usually limited by the disk access rate - typically 200 to 500K bytes per second for an IDE drive and 1 to 2 Mbytes per second for a SCSI (local bus) drive.

When compressing from SNAPPER, the JPEG chip can read the data directly from SNAPPER and does not go via the ISA bus. Thus images can be compressed very fast - 40ms for full resolution (CCIR) grayscale and 80ms for full resolution (CCIR) colour. If a frame capture time of 40ms is included, this results in 12.5Hz for grayscale and 8.33Hz for colour sequence acquisition rates for CCIR video. A 60Hz video image has about 30% less data than 50Hz video and therefore will run proportionally faster.

OTHER ISSUES

Colour decompression is done in RGB format. This involves two ISA bus cycles per pixel. Alternatively data could be read out in YUV format (only one bus transfer per pixel) and then converted via a fast LUT to 8 or 16 bit colour. (The LUT would be too large to generate full quality 24 bit output). This can potentially increase decompression speed by a factor of two for applications requiring colour decompression for 8 or 16 bit displays.

When benchmarked against C-Cube's "Luigi" demonstration board, CRUNCH-ISA is around two to three times faster.

BENCHMARKS

Typical speeds achieved on a 486-DX2-66 with VESA Local Bus disk and graphics are as follows:

Decompress and display a 640 x 480 image from disk (including file I/O):

Colour: 880ms to 24 bit RGB, 600ms to 16 bit RGB
Grayscale: 280ms.

Compression/decompression time for a 640 x 480 image (not including file I/O):

Colour: 600ms.
Grayscale: 160ms.

SNAPPER!

Snapper-16: Switching Cameras and Achieving Rapid Locking

v1.2 March 1999

TECHNICAL NOTE 3

INTRODUCTION

This technical note explains the Snapper-16 software calls that are required to achieve near instantaneous locking when switching between cameras.

SWITCHING CAMERAS

To switch cameras, *SNP16_set_video_source* is used to select one of three video input channels. However under the default set up conditions it can take up to 1 second to achieve full lock (video lock and colour lock).

ACHIEVING RAPID LOCK AFTER SWITCHING CAMERAS

In order to achieve rapid lock after switch cameras, Snapper-16 should be set up using the following commands:

```
/* Fast vertical lock */
SNP16_set_vertical_mode(Hsnapper, SNP16_VERT_SEARCH);
/* fast line lock       */
SNP16_set_vcr_mode(Hsnapper, TRUE);
```

Using the above mode, video lock (that is to produce a stable correct picture) should take 1 frame and colour lock should take anything from 1 to 6 frames (typically 2 or 3, rarely more).

If switching from a source with no camera connected, it typically takes 1 extra frame in additional to the time stated in the paragraph above.

If images are being acquired from one channel and then from another, a further enhancement can be achieved by switching cameras before reading out the data as shown by the following code fragment:

```
SNP16_capture(Hsnapper);
SNP16_select_video_source(Hsnapper, 2);
/*
 * whilst we are reading out data for the last frame from the
 * previous camera, we are allowing Snapper-16 a significant
 * amount of additional time to achieve lock.
 */
CRUNCH_compress_from_SNP16(Hcrunch, Hjpeg_image);
```

The function *SNP16_is_locked* can be used as normal.

SNAPPER!

Snapper-24: Level Controls

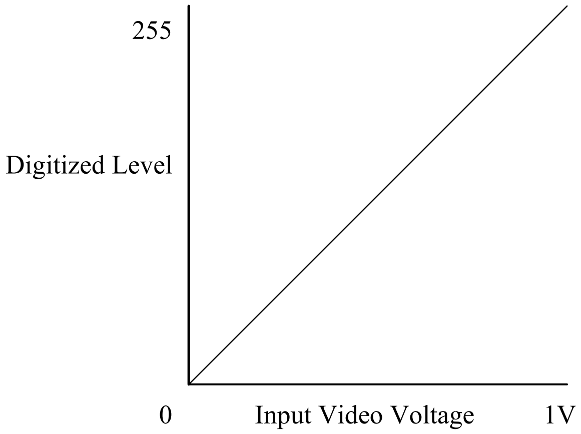
v1.3 March 1999

TECHNICAL NOTE 4

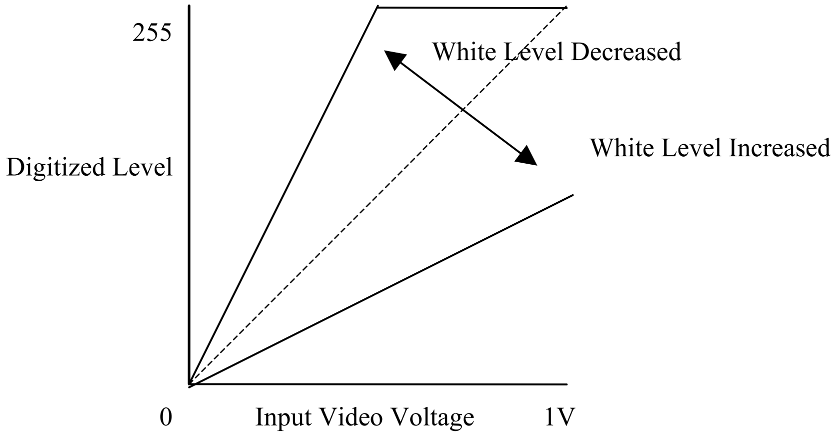
This technical note shows the effect of the Snapper-24 controls for white level, black level and clamp level. These are controlled by the library function *SNP24_set_levels*. These controls allow brightness and contrast to be adjusted, and they also allow the full digitization range of the board to be applied to a certain brightness region of the image. This note also applies to Snapper-8 except that on Snapper-8 the clamp level cannot be adjusted from its default setting.

TRANSFER CHARACTERISTICS

With all the levels at default values the following transfer characteristic between input video and digitized levels will result[1]:



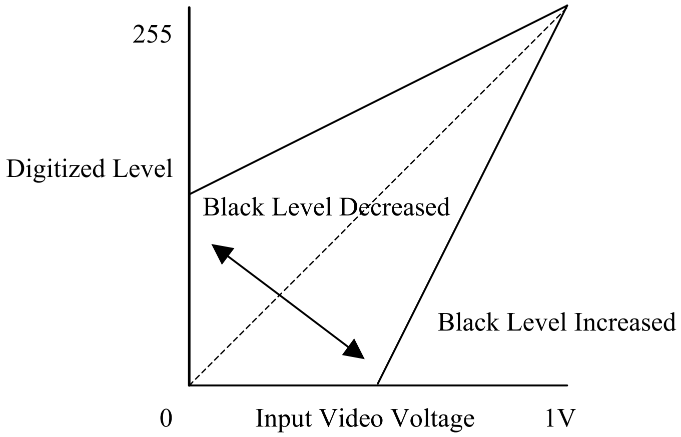
If the white level is changed with all other levels at default values the following transfer characteristics would result:



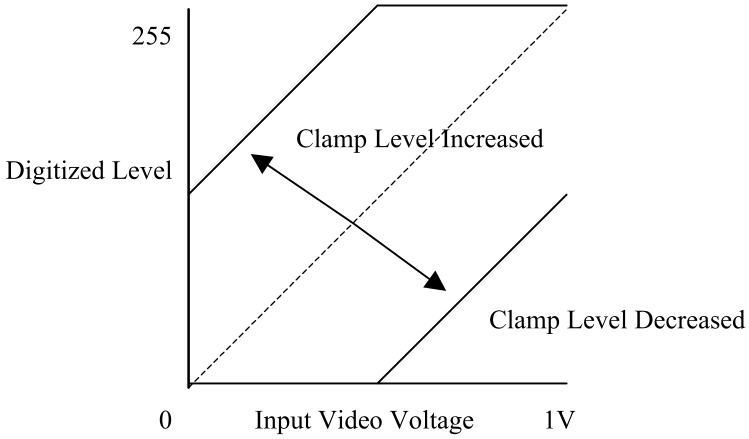
---

[1] The scale on these graphs is shown to help understand the graphs and is not intended to indicate precise digitization values for a given input.

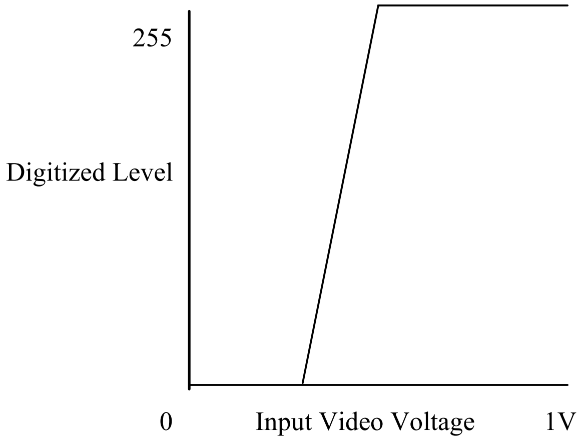
If the black level is changed with all other levels at default values the following transfer characteristics would result:



If the clamp level is changed with all other levels at default values the following transfer characteristics would result:

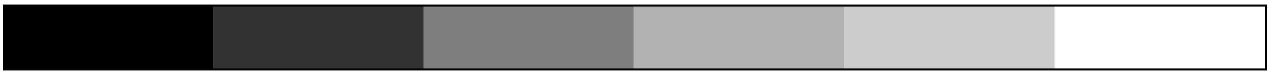


If the black level is increased (e.g. to 60) and the white level decreased (e.g. to 90), but with clamp at its default value the following the following transfer characteristic would result:



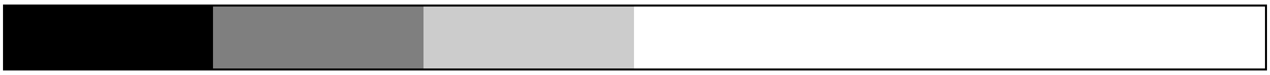
VIDEO BARS EXAMPLE

If a mono bars source is connected to the Snapper-24 an image similar the that shown below should result with all the levels at default values.



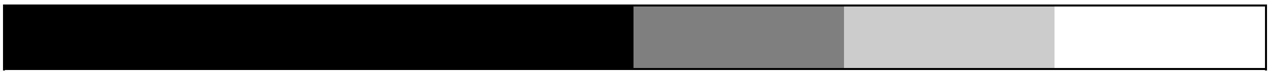
Note that there is a uniform range between the left hand bar being black and the right hand bar being white.

If the white level is decreased by 50 with all other levels at default values the following image would result:



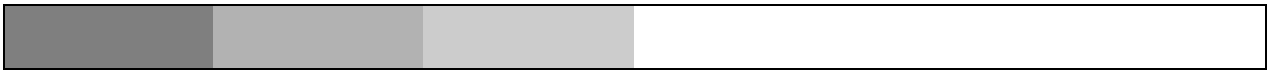
Note that the left hand bar is still black, but bars on the right of the image have all become white. Those inbetween still have a uniform black to white range, but with a greater step change in brightness between bars than with the default settings.

If the black level is increased by 50 with all other levels at default values the following image would result:



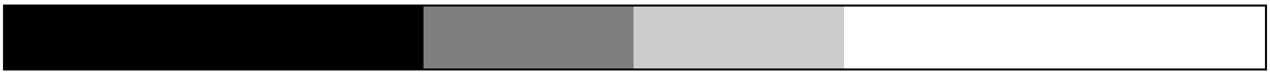
Note that the right left hand bar is still white, but bars on the left of the image have all become black. Those inbetween still have a uniform black to white range, but with a greater step change in brightness between bars than with the default settings.

If the clamp level is increased by 50 with all other levels at default values the following image would result:



Note that the left hand bar is now grey, and bars on the right of the image have all become white. Those inbetween have the same step change in brightness between bars as with the default settings.

If the black level is increased (e.g. to 60) and the white level decreased (e.g. to 90), but with clamp at its default value the following image would result:



Note that bars on the left of the image have all become black, and bars on the right of the image have all become white. Those inbetween still have a uniform black to white range, but with a greater step change in brightness between bars than with the default settings.

SNAPPER!

Snapper-DIG16: Porting New Cameras

v1.5 March 1999

TECHNICAL NOTE 5

This technical note describes how to add support for new digital frame cameras to Snapper-DIG16.

CHECK REQUIRED CAMERA FEATURES CAN BE SUPPORTED

First check for any features on the camera which Snapper-DIG16 does not support:

- Check that the physical interface is compatible by comparing the camera's data sheet with the Snapper-DIG16 data sheet. For example check that the data signals are to RS-422 standard, count the number of TTL and RS-422 control/status lines required, check that standard control signals exist (pixel clock, line enable, frame enable). If serial comms are needed check the baud rate used.
- Check that the software library supports the required modes. For example the initial software release does not support serial comms.

If a problem shows up while doing these checks contact Active Imaging technical support - it may be possible to add extra software library support, or because the hardware control of Snapper-DIG16 uses a field programmable gate array (FPGA) which is software loadable it may be possible to supply an alternative FPGA design to work around the problem.

MAKE THE INTERFACE CABLE

Check the pinout of the camera and that of the Snapper-DIG16 from their datasheets and work out the required cable pinout:

- *Data:* Connect the MSB of the camera to MSB on the Snapper-DIG16; the next bit to MSB-1 and work upwards until all camera data bits have been connected. There is no need to tie unused inputs on the Snapper-DIG16. This defines an MSB aligned cable, which is the recommended connection.
- *Control:* Directly connect pixel clock, line enable and frame enable. Similarly directly connect serial comms lines. The choice of pins for general purpose control lines is arbitrary, but note that OUT_A is the only line which can be driven with a hardware exposure control pulse, and drive any TTL control lines from OUT_A+ to OUT_D+.

For reliable operation at fast data rates, and to minimise radiated interference, screened twisted pair cable must be used. Similarly, to comply with EMC regulations good quality screened cables and connector hoods should be used, and ferrite rings may be needed on each end of the cable.

If all signal connections are RS-422 then cables up to 10m long should work, but if any control signals are TTL then make the cable as short as practical and definitely not more than 2m long.

CALL SNAPPER-DIG16 LIBRARY CONFIGURATION FUNCTIONS

Call *DIG16_initialize* with the parameter *DIG16_CUSTOM_CAMERA* to allow the application to set up all the camera parameters. Then call the following functions to set up the Snapper-DIG16 to work with the camera (look at the supplied source code to *DIG16_initialize* in dig16ini.c for example code to copy), and note that the order that the functions are called in is important - for instance *DIG16_set_active_area* must be called before *DIG16_set_ROI*:

- *DIG16_set_camera_info:* This tells the software how the camera cable has been wired, and how many data bits are coming from the camera.
- *DIG16_set_active_area:* Unless the camera data sheet is very good some trial and error may be needed to get the parameters precisely right. The simplest way to do this is to deliberately set the active area slightly too large, and then adjust the ROI in the application until the picture is correct, then note the ROI setting used and correct the active area setting. It is recommended to start with an ROI which is much smaller than the full image, and roughly central. Once this is working extend the ROI to find the full active area.
- *DIG16_set_ROI_rounding:* A suggested starting value is 2, 1.
- *DIG16_set_format:* Y8 format is recommended as a default setting to get maximum transfer speed. Alternatively use Y16 for cameras with more than 8 data bits if image quality is more important than speed.
- *DIG16_set_LUTs:* It is recommended that this is called with the *DIG16_LUT_INIT* parameter. It is necessary to set up the memory for the LUT array, and it is recommended that this is done dynamically - see the *DIG16_initialize* code for examples.
- *DIG16_set_ROI:* Set to the required initial size. It is recommended to start with an ROI which is much smaller than the full image, and roughly central. Once this is working extend the ROI to the full active area. If the image breaks up or capture timeouts occur near the expected full ROI size the setting of *DIG16_set_active_area* is still wrong, allowing the Snapper-DIG16 to try to capture more data than the camera is providing.
- *DIG16_set_capture:* The mode selected depends on the camera.
- *DIG16_set_trigger:* The mode selected depends on the camera.
- *DIG16_set_clk:* The mode *DIG16_CLK_IN_POS* is most common, but if the image is very noisy try *DIG16_CLK_IN_NEG*. The camera data sheet should show which mode is correct.
- *DIG16_set_ctrl_io:* The modes selected depend on the camera and the cable pinout chosen. If the required modes are complex it may be easiest to package them into a separate function similar to *DIG16_set_mplus_ctrl* in the standard library.

TROUBLESHOOTING

The camera should work first time, but if not ...

- If nothing works connect a camera supported by the standard library and run a standard application to check that the Snapper-DIG16 is installed correctly and working.
- If capture timeouts occur there is a control problem - check the pixel clock, line enable and frame enable are connected correctly. If they are connected OK then check that they are active (use an oscilloscope) - if not check that the control lines are set correctly. If these are OK try reducing the ROI - it may be bigger than that of the camera (see notes earlier).
- If FIFO full errors occur (possibly with capture timeouts as well) the board may not be keeping up with the camera. In this case reduce the camera's pixel clock if possible, otherwise increase the sub-sample factor and /or reduce the ROI. If this cures the problem it may be necessary to use a different computer which supports faster DMA - contact technical support.
- Image noisy: Check that the pixel clock polarity is correct.

- Image brightness wraps round or some areas of image in inverse video: Check that all the data bits are connected and in the correct order. Are the numbers passed to *DIG16_set_camera_info* correct? Has an incorrect LUT been passed to *DIG16_set_LUTs*?
- Image slanted: Try increasing the numbers passed to *DIG16_set_ROI_rounding*. This might be showing a bug elsewhere in the library or application.
- Program crashes with memory problems such as access violations or segmentation faults: Check that the lut passed to *DIG16_set_LUTs* is large enough.

SNAPPER!

Snapper-8 & Snapper-24: Vertical Banding

v1.2 March 1999

TECHNICAL NOTE 6

This technical note explains why vertical bands sometimes appear on images captured with Snapper-8 and Snapper-24, and how to eliminate them. This banding is alternate light and dark bands running down the image, with typically several tens of bands across each line.

CAUSE OF BANDING

The banding is caused by ‘aliasing’, and is normally the result of the output from a video camera containing high frequency components from its internal CCD clock.

CURE

This problem could be cured using external anti-aliasing filters[1] but this is an expensive solution which should not be necessary in most cases.

The recommended solution is to vary the pixels per line using the function *SNP24_set_pix_per_line* so that sampling clock precisely matches the clock used by the camera.

The correct value to pass to *SNP24_set_pix_per_line* can be found by trial and error, but it is easy to calculate the value if the data sheet or instruction manual for the camera gives both the pixel clock frequency and the horizontal sync frequency. The required value is obtained by dividing the pixel clock by the horizontal frequency. As an example, the Pulnix TM-765 datasheet gives the following figures:

• Clock	28.375 MHz
• Pixel Clock	14.1875 MHz
• Horizontal Frequency	15.725 kHz

The ‘Clock’ is not relevant (this is the CCD clock, which for most cameras is double the pixel clock), so dividing 14.1875×10^6 by 15.725×10^3 gives 902 as the value for *SNP24_set_pix_per_line.*

Having found the optimum value of pixels per line it is worth trying different values of PLL phase using the options *SNP24_PLL_PHASE_0* to *SNP24_PLL_PHASE_270* of function *SNP24_set_clk*. This may give an overall improvement in image quality.

---

[1] Snapper-8 and Snapper-24 do not have anti-aliasing filters fitted as standard. This is because good filters are very expensive and there is not room on the PCB to fit them. Filters compact enough to fit the board would degrade the image quality for all cameras.

SNAPPER!

Snapper-8 & Snapper-24:
Achieving Very Stable Sampling

v1.2 March 1999 TECHNICAL NOTE 7

This technical note explains the advantages and disadvantages of the numerous options provided on Snapper-8 and Snapper-24 for video synchronisation and clock generation, emphasising the effect on sampling jitter. All of these options are entirely satisfactory for most uses, but for critical applications (e.g. stereoscopic imaging) it is important to select an option with very low jitter. The ideal setup described below gives 0 nanoseconds jitter, but the worst gives more than ±25 nanoseconds jitter. A technical background section for each of the options explains why the performance is good or bad, but the recommendations can be used without needing to understand these explanations.

TECHNICAL BACKGROUND

As a reference, one pixel is typically 81ns wide for EIA video standards, so for example jitter of ±10ns is ¼ of a pixel.

Note that it is assumed that the video source itself is stable - if the video source is poor (e.g. from a VCR), none of the options described here will make much difference.

Most frame grabbers, including Snapper-8 and Snapper-24, use a phase locked loop (PLL) to generate their pixel clock from the horizontal sync (HSYNC) from the input video. All phase locked loops will generate some jitter even if the input HSYNC is perfectly stable, but any degradation of HSYNC will increase the jitter.

All the figures are quoted for the PLL in both fast lock and slow lock mode. In general slow lock should always be used where jitter performance is important, but note that the better the sync waveform, the less the difference is between the two modes.

The jitter figures have been obtained using a digital storage oscilloscope by measuring the time spread of the output of the PLL divided down to HSYNC frequency while triggering the oscilloscope from the sync output of a test pattern generator. These are measured figures which give a good comparative feel for the relative merits of the options, but should not be taken as part of the official specification of Snapper-8 or Snapper-24. The worst case figures quoted are the maximum jitter over 1 minute; the typical figures are a somewhat subjective judgement obtained by ignoring the ‘ghost’ traces which very rarely occurred.

OPTION 1: SYNC OFF VIDEO; PLL CLOCK

This is the default, as set by *SNP24_initialize*. It has the considerable advantage of working with all cameras, and not needing any additional cabling for synchronisation. The resulting images are entirely satisfactory for most uses, but critical applications may not work because of the relatively poor jitter.

Jitter: Typical figures from a clean video source of constant brightness are ±7ns (slow lock) or ±10ns (fast lock). Worst case figures are ±12ns (slow lock) or ±25ns (fast lock). Note that even a slight amount of electrical noise will make the jitter much worse.

Technical background: This poor jitter is the result of extracting all the sync information from an analogue waveform. This waveform is bandwidth limited by the camera so that the sync’s rise time is around 250ns, has a peak to peak sync voltage of 0.3V, and is extracted from an AC coupled video waveform. This (essential) AC coupling and DC restoration means that if the average brightness of video waveform changes, the DC restoration circuit must charge or discharge the coupling capacitor to maintain the required DC level. This will happen very quickly, but it will still take a finite time. The sync extractor will try to track this change of voltage, but because of the slow rise time of the sync signal even a small voltage change will significantly shift the time at which the sync is detected. Therefore even in an electrically quiet environment it is unreasonable to expect to get good jitter performance from this configuration.

OPTION 2: SYNC OFF ANALOGUE CSYNC; PLL CLOCK

Here an extra cable is needed to take CSYNC from the camera to the CSYNC input on the Snapper. From a jitter viewpoint this is only slightly better than same as option 1.

Jitter: Typical figures are ±7ns (slow lock) or ±8ns (fast lock). Worst case figures are ±11ns (slow lock) or ±21ns (fast lock).

Technical background: Analogue CSYNC should be bandwidth limited, so the rise time is as slow as in option 1, and the sync must still be AC coupled. The only benefits are that the video and sync are not combined on the same waveform, and that the sync does not have to pass through the DC restoration circuitry. Therefore there is no possibility of changes in image brightness affecting the jitter.

OPTION 3: SYNC OFF HSYNC AND VSYNC; PLL CLOCK

Here extra cables are needed to take HSYNC and VSYNC from the camera to the HSYNC and VSYNC inputs on the Snapper. From a jitter viewpoint this is much better than options 1 or 2, especially if an RS-422 connection is used rather than TTL. The only problem here is that not all cameras output HSYNC and VSYNC, and even if they do, most output them as TTL.

Jitter: A typical figure using RS-422, or TTL with a **very** short cable, is ±3ns (slow and fast lock). Worst case figures are ±6ns (slow lock) or ±7ns (fast lock).

Technical background: Both TTL and RS-422 signals are digital with fast rise times, and the HSYNC can be passed directly to the phase locked loop. Hence the jitter performance is limited only by the PLL. TTL/RS-422: TTL is not suitable for driving cables, and even cables under 1 metre long can significantly degrade the waveform, resulting in noisy edges and hence more jitter. Cables more than a few metres long probably will not work at all with TTL. It is recommended to either change to a camera with RS-422 outputs, or make a small PCB with TTL to RS-422 converter chips and fit it directly onto the camera's connector. RS-422 should work with cables up to 10m long if proper screened twisted pair cable is used.

OPTION 4: SYNC OFF HSYNC AND VSYNC; EXTERNAL CLOCK

This is the same as option 3, except that an additional cable takes the RS-422 clock from the camera to the PCLK input on the Snapper. From a jitter viewpoint this the optimum connection, with the direct connection between the camera and the clock giving zero jitter. The only problem here is that very few analogue cameras output RS-422 pixel clock[1].

Jitter: Zero.

---

[1] An alternative is to use a digital camera in conjunction with Snapper-DIG16. Almost all digital cameras use separate RS-422 signals for clock, horizontal 'sync' and vertical 'sync', and hence do not suffer from jitter problems.

Technical background: Check that the correct output is used from the camera - some cameras output their CCD clock, which is often double the frequency of the pixel clock. It is unreasonable to expect a TTL clock connection to work - RS-422 has got to be used. If a camera outputs a TTL clock it is unlikely that this will drive a cable more than several centimetres long, so as with option 3 either change the camera or make a converter PCB.

OPTION 5: SYNC OFF VIDEO OR CSYNC; EXTERNAL CLOCK

This might appear to be a shortcut to the performance of option 4. From a clock jitter viewpoint this is still optimum, but a few pixels side to side jitter of entire lines may be present.

Jitter: Zero on clock, but a few pixels side to side jitter of entire lines.

Technical background: The problem here is that the HSYNC and VSYNC extracted from the video or CSYNC have the same jitter as in options 1 and 2. Therefore while the pixel clock will be stable, so that there will be no jitter within a line, the clock pulse in which the extracted HSYNC is detected will vary from line to line. This will cause individual lines to jitter from side to side, often by only 1 pixel, possibly by several pixels. This will not happen as much if the PLL is used to generate the clock because the PLL will track the jitter on HSYNC.

OPTION 6: INTERNAL SYNC

Here the Snapper is put into internal sync mode, and it provides the sync(s) and optionally clock to the camera. Depending on which connections are made, equivalents to options 2, 3, 4 and 5 result, with the same advantages and disadvantages.

The equivalent of option 5, where HSYNC, VSYNC and pixel clock are all connected, should work just as well as option 5 if a camera can be found which accepts these inputs. The jitter performance of the equivalents of options 2, 3 and 4 depends on how good the synchronisation and clock circuits are in the camera.

SUMMARY

For many applications the level of jitter resulting from any one of the options described is insignificant. However where jitter performance is critical, option 4, using a direct connection of HSYNC, VSYNC and pixel clock, is clearly the best; with option 3 a recommended alternative.

SNAPPER!

Snapper SDK for MS-DOS: Minimizing Real Mode Memory Usage

v1.2 March 1999 TECHNICAL NOTE 8

This technical note explains how to minimize the memory usage when using real mode DOS.

TECHNICAL BACKGROUND

When developing applications under real mode DOS, it is desirable to make the final executable as small as possible, since there is only 640K of memory available and then a proportion of this is taken up by drivers and TSRs etc.

The most desirable solution is not to use real mode DOS at all, but use protected mode DOS, which allows 32Mb of flat memory space (and is faster too). Snapper is fully supported under protected mode DOS using the Watcom and Symantec C++ compilers and is the preferred development path for DOS developers. See the Snapper SDK datasheet for more details.

There are basically two methods that allow the memory usage of the application to be reduced - firstly to make the actual executable smaller; and secondly, to use less memory dynamically during execution. The next two sections list steps to take to minimize memory usage using both of these techniques.

STEPS TO MINIMIZE MEMORY USAGE - EXECUTABLE SIZE

1. Set the compiler optimizations for minimum size. This is done in the Microsoft and Borland project makefile examples in the SDK.
2. Only link in the Snapper libraries that are necessary. For example if Snapper-16 on the ISA-BIB is being used, then only the TMG, Base and Snapper-16 libraries are needed. The Snapper compiler directives (*_SNP16*, *_SNP24*) ensure only the relevant Snapper library is compiled into the executable. See the Snapper Developer's Guide and the example project makefiles for more information.
3. To minimize the amount of functions linked into the final application from the TMG library, don't use the *TMG_CK* or *TMG_LUT* functions unless directly needed in the application.
4. Do not use *TMG_image_write* or *TMG_image_read*. Instead use the functions that they actually call. For example if you need to write a TIFF file, use the function *TMG_write_TIFF_file* directly. This saves linking in all the other file read/write routines. See the TMG manual pages for *TMG_image_read* and *TMG_image_write*.
5. Similarly the function *CRUNCH_compress_image_to_image* should not be used (or *CRUNCH_decompress_image_to_image*). This is because this function pulls in all the file reading/writing functions. In fact this function is not part of the Crunch library for real mode DOS anyway, but given in source form in the "lib\src\crunch" directory in the SDK. If absolutely necessary this source code may be compiled in, but it is preferable to directly use the functions *CRUNCH_compress* and *CRUNCH_decompress* in a strip processing loop. See the "readme.txt" file in the "lib\src\crunch" directory for more details. Also note that the TMG software JPEG compression and decompression functions are not supported under real mode DOS.
6. Do not use *TMG_image_convert*, but one of the functions it calls directly. The functions are listed here and the format conversion should be self-explanatory from the name of the function

itself. Note that the "convert_to" functions only accept YUV422 if that format appears in their name. Every function takes three parameters - input image handle, output image handle and *TMG_RUN*. The functions are:

TMG_convert_to_RGB24
TMG_convert_to_RGBX32
TMG_convert_to_BGRX32
TMG_convert_to_XBGR32
TMG_convert_to_BGR24
TMG_convert_to_RGB16
TMG_convert_to_RGB15
TMG_convert_to_RGB8
TMG_convert_to_CMYK32
TMG_convert_to_Y8
TMG_convert_to_Y16
TMG_convert_to_paletted_LUT
TMG_convert_to_RGB8_dither
TMG_paletted_to_RGB24_or_Y8
TMG_convert_to_YUV422
TMG_YUV422_to_RGBX32
TMG_YUV422_to_BGRX32
TMG_YUV422_to_XBGR32
TMG_YUV422_to_RGB24
TMG_YUV422_to_BGR24
TMG_YUV422_to_RGB16
TMG_YUV422_to_RGB15
TMG_YUV422_to_paletted_LUT
TMG_YUV422_to_RGBX32_LUT
TMG_YUV422_to_BGRX32_LUT
TMG_YUV422_to_XBGR32_LUT
TMG_YUV422_to_RGB24_LUT
TMG_YUV422_to_BGR24_LUT
TMG_YUV422_to_RGB16_LUT
TMG_YUV422_to_RGB15_LUT
TMG_YUV422_to_Y8

STEPS TO MINIMIZE DYNAMIC MEMORY USAGE

1. Set the "*TMG_LINES_THIS_STRIP*" image parameter option to a minimum of 8 for JPEG functions and 4 or even 2 if necessary for other imaging functions. See the concepts section in the TMG manual for more details.

2. If the TMG library colour conversion LUT is used to convert from YUV422 to RGB or paletted data (for example Snapper-16 code will typically use this - see conversion functions in "s16dos.c"), then the size of this LUT may be varied by changing the following #define values in "tmg.h" to smaller numbers (this will result in reduced colour conversion quality though):

```
#define TMG_Y_BITS16        5    /* for YUV422 to RGB16 */
#define TMG_U_BITS16        5
#define TMG_V_BITS16        5

#define TMG_Y_BITS8         5    /* for YUV422 to paletted */
#define TMG_U_BITS8         5
#define TMG_V_BITS8         5
```

   This option is only available to OEMs who have source code under non-disclosure.

3. Remove all unnecessary DOS TSRs and drivers and/or put them into high memory.

SNAPPER!

Snapper-DIG16: Line Scan Support

v1.3 March 1999

TECHNICAL NOTE 9

This technical note describes how the Snapper-DIG16 supports digital line scan cameras.

CONCEPTS

The Snapper-Dig16 can be configured for a variety of line scan camera applications, due to the flexible I/O configurations. In this technical note, (and library manuals) the following terminology is used:

The *LineTrigger* is an input signal which can be used to synchronise the camera to the rate of motion of the object being observed. For example, by mechanically connecting a shaft encoder to a conveyor belt which is carrying the object, the aspect ratio of the object will remain fixed, irrespective of the speed of the conveyor belt. Alternatively for systems that run at a fixed speed, or where variations in the aspect ratio are not important, Snapper-Dig16 can use a hardware timer to generate the *LineTrigger* reference pulse. For cameras which "free run", ie do not need a *LineStartOut* signal to trigger them, the *LineTrigger* signal can be ignored.

LineStartOut is an output signal generated by Snapper to trigger the camera.

LineStartIn is an input signal generated by the camera at the start of each line of data.

LineStart is the signal used as an X axis reference by the Snapper ROI (Region Of Interest) hardware.

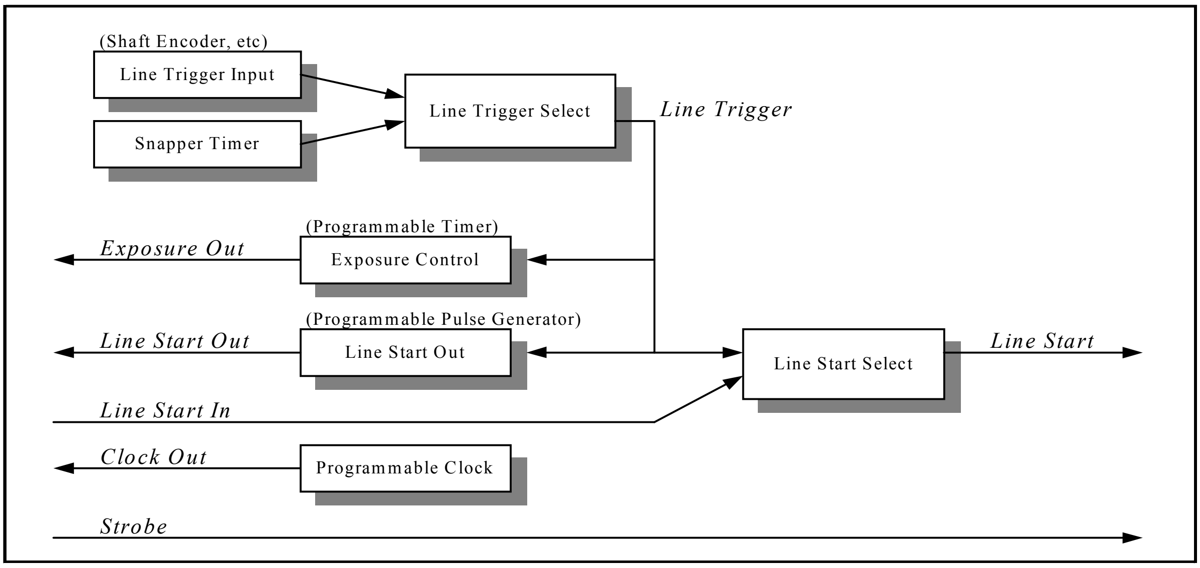
The cameras exposure can be controlled with the *ExposureOut* signal.

Snapper can generate a *ClockOut* signal at discrete frequencies to drive the camera.

AcquisitionTrigger is used to determine whether a given line of camera data should be acquired or ignored.

BLOCK DIAGRAM OF SIGNAL PATHS

The diagram below shows how the line control signals are arranged.



LINE START

The *LineStart* signal is used by the Snapper-Dig16 as a reference point along a line of data. The *LineStart* signal can be taken from one of three sources:

- An external *LineTrigger* source, ie shaft encoder connected to a conveyor belt. This is specifically for ensuring that objects moving at a variable rate, for example on a conveyor, are still acquired with the same aspect ratio.
  Note that the *LineTrigger* is different from the *AcquisitionTrigger*, which is discussed later.
- An internal trigger source is used when there is no suitable external event, and either the conveyor is moving at a fixed speed, or changes in the aspect ratio do not affect any subsequent image processing.
- The *LineStartIn* signal is provided by the camera, when it starts outputting valid data. This is usually a fixed number of pixel clocks after the *LineStartOut* signal has been driven to the camera.

LINE START OUT

The width of the *LineStartOut* signal is programmable under software control, to be a fixed number of camera clocks. This is a requirement of some line scan cameras to correctly control the CCD.

ACQUISITION TRIGGER

A capture trigger input can be used to initiate the capture of a number of lines. This is typically used to start acquisition whenever an object passes underneath the line scan camera. A light barrier or similar PIP (Part-In-Place) detector can be used to generate the capture trigger pulse.

The *AcquisitionTrigger* signal can be selected from any of the I/O input signals or TTL inputs signals for rev 4 boards and later), and can be used in one of three modes:

- The input signal is ignored, and capture starts on the line immediately following the software call to arm the hardware.
- The input is used to start the acquisition. Once the hardware has been armed, acquisition is stalled until there is an active edge on the selected *AcquisitionTrigger* input. Once this edge has occurred, the input is ignored.
- The input is used to “envelope” the acquisition. Once the hardware has been armed, any given line will be acquired while the selected *AcquisitionTrigger* input is asserted at the start of that line.

A light or proximity sensor typically generates this AcquisitionTrigger signal whenever an object passes underneath the line scan camera.

CLOCKS

The Snapper-DIG16 has a discrete pixel clock generator, which can drive a clock signal out to the camera. Generally the camera will provide a strobe signal back to Snapper, though this signal is not necessarily of the same frequency. When the camera provides a clock signal, this is used by Snapper to sample the control and data signals. Alternatively, if the camera does not supply a clock, then one of four phases of the *ClockOut* signal can be used internally.

The Snapper-DIG16 supports both single and dual channel line scan cameras (hence data lines 0-15 may be required). For line scan acquisition the maximum pixel clock is 25 MHz and all data lines, line enable and pixel clock must be RS-422.

EXPOSURE CONTROL

Snapper DIG16 also supports programmable integration or exposure control with line scan cameras. The exposure control line is driven high on the start of the selected line enable input. This holds off the exposure for a user defined period up to 13.6ms with a resolution of 53µs. Alternatively changing the polarity of this line allows the exposure period itself to be defined. This can ensure that for brightly illuminated scenes the camera is exposed to the desired amount of light. The nature of exposure control will vary from manufacturer to manufacturer, and possibly camera to camera, so it is advised that you examine the manual supplied with the camera to ensure the correct desired performance.

EXAMPLE LINE SCAN APPLICATIONS

Having installed the software (see the Snapper! Installation Guide) look at the program 'd16line' for examples for controlling the line scan acquisition from a Dalsa CLCX camera. The application sheet D16LINE contains full details of this program.

SNAPPER!

Snapper SDK for Windows NT: Interrupts and Callback Functions

v1.1 March 1999 Technical Note 10

This technical note explains how to use interrupt and callback functions in Windows-NT. It is also relevant to other operating systems supporting interrupts; currently VxWorks, LynxOS and Solaris.

WHAT ARE INTERRUPTS AND CALLBACKS?

The Snapper hardware can signal a number of different events. These events occur asynchronously to the application, and are called interrupts, as they interrupt the main program flow.

The available interrupt sources are different for each Snapper family; for Snapper-Dig16 these currently include Start of Frame, End of Line, FIFO Overflow, Acquisition Trigger Asserted, and End of DMA Transfer; for Snapper-8/24 these currently include Vertical Sync, Data Ready and End of DMA Transfer.

Each source can be masked (effectively disabled), such that the given event does not generate an interrupt. However if the relevant source is enabled when an interrupt occurs, a user defined function (Callback function) is called by the Snapper libraries. This Callback function runs in a separate thread from the main application and at a higher thread priority. The application code can then execute any required event processing; this may include capturing a new image, processing the current image, or signalling to the main application that data is available.

MAIN PROGRAM STEPS

There are a number of program steps required in order to use interrupts and Callbacks

1. Initialise the hardware, and configure for the appropriate video source.
   If the Callback function is used to signal events to the main application, then instantiate all the required semaphores.
2. Supply a pointer to user defined function which will get called when the interrupt occurs. There is a defined API for this function, in terms of the parameters it accepts and its return values.
3. Enable all the relevant interrupt sources.
4. Process all interrupts as they occur, including signalling via semaphores to the main application.
   Continue to do this until the application is to be terminated.
5. Disable all interrupt sources.
6. Reset the Callback function pointer to be NULL.
7. Deconfigure all the Snapper hardware. The application must ensure that the Callback function is NOT currently executing. If the hardware is deconfigured whilst the Callback is blocked waiting on an event, it is likely that the Callback will produce an access violation when trying to access the deconfigured hardware.

PROGRAM EXAMPLE

```
main()
{
   /* 1.  Perform Application Specific Initialisation */
    …
   CreateSemaphore();

   /* 2.  Register our CallBack function */
   DIG16_set_callback( hSnapper, DIG16_CALLBACK_SET, MyCallback, &MyData );

   /* 3.  Enable interrupts – acquisition trigger and capture complete */
   DIG16_set_interrupts( hSnapper, DIG16_INT_ACQ_TRIGGER, TRUE);
   DIG16_set_interrupts( hSnapper, DIG16_INT_CAPTURE_COMPLETE, TRUE);

   while (ProcessImages)
   {
      /* 4a.  Wait for an image to be acquired */
      WaitForSemaphore();

      /* 4b.  Process the acquired image */
       …
   }

   /* 5.  Disable all interrupts */
   DIG16_set_interrupts( hSnapper, DIG16_INT_ACQ_TRIGGER, FALSE);
   DIG16_set_interrupts( hSnapper, DIG16_INT_CAPTURE_COMPLETE, FALSE);

   /* 6.  Unregister the Callback function */
   DIG16_set_callback( hSnapper, DIG16_CALLBACK_SET, NULL, NULL);

   /* 7.  Deconfigure the Snapper hardware */
    …
}


void EXPORT_FN MyCallback (Thandle hSnapper, ui32 dwIntSource, void *pvMyData)
{
   /* Start another acquisition on each trigger occurrence */
   if ((dwIntSrc & DIG16_INT_ACQ_TRIGGER) != 0)
      DIG16_capture_to_image( hSnapper, hImage, DIG16_START_AND_RETURN);

   /* Signal semaphore each time acquisition completes */
   if ((dwIntSrc & DIG16_INT_CAPTURE_COMPLETE) != 0)
      SignalSemaphore();
}
```

This example does not perform any error checking for clarity. For further examples, refer to the Snapper-8/24 Programmer’s Manual, Snapper-Dig16 Programmer’s Manual, and the example applications.