

# nae-ex-001

## NewAE Example Application #001

### *Framegrabber*

#### Summary

<b><i>Target</i></b>	<b><i>LoonBoard</i></b>
<b><i>Target Revision</i></b>	<b><i>2.4</i></b>
<b><i>Latest Version of App.</i></b>	<b><i>Jan 23, 2006 (Beta)</i></b>
<b><i>Summary</i></b>	<b><i>Framegrabber application, loads data from the LoonBoard to the PC and vice-versa. Also supports transparency.</i></b>

#### Device Utilization

<b><i>AVR Flash Used</i></b>	<b><i>6564 of 15872 (41%)</i></b>
<b><i>FPGA Total Number Slice Registers</i></b>	<b><i>234 of 4704 (4%)</i></b>
<b><i>FPGA Total Number 4-in LUTs</i></b>	<b><i>334 of 4704 (7%)</i></b>
<b><i>Block RAMs</i></b>	<b><i>3 of 14 (21%)</i></b>

## 1.0 Introduction

A framegrabber is a device that records a single frame of video. The LoonBoard makes this very easy, and it barely even uses the maximum capabilities of the LoonBoard.

You will need a fairly standard setup – one LoonBoard connected to the computer, video in, and video out. Video in can come from something like a VCR, DVD player, or camera. Video out will likely be a TV monitor.

You should have read the tutorial in NAE-AN-003, as that will familiarize you with the tools that will be used.

## 2.0 The FPGA

Start up ISE and let's take a look at the project. All the files for the project are in the framegrabber\FPGA\framegrabber directory. Open the project file framegrabber.ise in Xilinx ISE version 7.1 or later. If you are not familiar with ISE you should read the document “ISE 7 In-Depth Tutorial” on the Xilinx website, chapters 1 and 2 ( [http://direct.xilinx.com/direct/ise7\\_tutorials/ise7tut.pdf](http://direct.xilinx.com/direct/ise7_tutorials/ise7tut.pdf) ).

You will notice four hierarchical blocks – hardware\_interface, avr\_protocol, bt656\_to\_loonbus, and spi. The hardware\_interface block is the overall block that interfaces to the LoonBoard itself. As well in contains any small logic that is specific to the application.

The avr\_protocol contains the link between the AVR and the FPGA. This is done using SPI, which is one of the fastest links available between them. The spi module provides the physical interface. The bt656\_to\_loonboard provides an interface between the video encoder and video decoder.

Future examples will include compression to greatly reduce RAM usage. This is especially true for overlaying objects, where currently if you had large areas of the same colour it will send the entire frame despite the fact it is 90% blank.

You can program the hardware\_interface.bit file in by running:

```
lubloader -f hardware_interface.bit -P /dev/ttyUSB0
```

## 3.0 The AVR

The AVR part of the design is fairly simple, being close to the framework.

There is additional code to communicate between the AVR and the FPGA, and AVR and computer.

Building it is accomplished from the AVR directory by running “make clean” and then “make”. The main.hex file can be programmed in by:

```
lubloader -a main.hex -P /dev/ttyUSB0
```

### 3.1 Communications between AVR and FPGA

The communications protocol between the AVR and FPGA is very simple. It runs over the SPI and looks like this:

<b>Mode</b>	<b>Address 1</b>	<b>Address 2</b>	<b>Data 1</b>	<b>Data 2</b>	...	<b>Data n</b>
-------------	------------------	------------------	---------------	---------------	-----	---------------

Everything is transmitted MSB first.

#### Mode Byte

<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
<b>R/W</b>	<b>BS3</b>	<b>BS2</b>	<b>BS1</b>	<b>A19</b>	<b>A18</b>	<b>A17</b>	<b>A16</b>

#### R/W

1 when writing to memory, 0 when reading to memory (reading unsupported when reading from register space right now)

#### BS3 – BS0

Bank Select:

000 is registers (see hardware\_interface.vhd for registers)

001 is main RAM

#### A19-A16

High address bits

#### Address 1 Byte

<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
<b>A15</b>	<b>A14</b>	<b>A13</b>	<b>A12</b>	<b>A11</b>	<b>A10</b>	<b>A9</b>	<b>A8</b>

#### A15-A8

Medium address bits

#### Address 2 Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
A7	A6	A5	A4	A3	A2	A1	A0

### A7-A0

Low address bits

It is very important when reading data to put in one dummy read after the last address byte. The address auto increments on both reading and writing, so for reading start the address at 1 less than what you want to read from to counter the dummy read.

## 4.0 The PC Interface

The PC interface makes extensive use of the libtiff library. To build the software you will need this library, most Linux distributions should come with it. If building on Windows Cygwin can install this library.

This file format here is TIFF, as they can be saved in the YCrCb format which requires no processing, since that is how data is stored in RAM. You can then convert to other formats as needed.

For building on Linux just type "make" in the PC directory. For building on Windows edit the makefile to set it up as Windows. Do this by setting the "WIN32NATIVE" flag and changing the ser\_posix.c to ser\_win32.c.

The interface between LoonBoard and PC is a low 115200 serial link. As well the protocol is very simple, if synchronization is lost you need to restart the LoonBoard. This would happen if a download got interrupted for example, as the LoonBoard would be expecting more video data to be sent.

### 4.1 Using the PC Interface

The example application provides a fairly simple framegrabber interface. You can download or upload TIFF images from or to the LoonBoard. If you run the program with

```
grab.exe
```

or

```
./grab
```

you should see something like this:

```
grab [options] -t file
Options:
  -P value      Serial port to use
  -c           Grab frame from video to
```

	RAM.
-f	Output contents of RAM on video out.
-i	Output video input on video out.
-d	Get RAM contents from LoonBoard, store in file specified,
-u	Send file specified to the LoonBoard.
-b value	Set background color to value, value is from 1 to 254
-s skip:read	Set number of pixels to skip then read. Multiples of 2 only. See docs.
-v	Verbose output.

Now you need to do some tests. Connect up a video source, and check that the board is responding. Capture a frame and display it by typing

```
grab -P /dev/ttyUSB0 -cf
```

Then force the output to display the output again by typing:

```
grab -P /dev/ttyUSB0 -i
```

Hopefully that worked! If it did let's try grabbing a frame from the device. First capture a frame as above, then type:

```
grab -P /dev/ttyUSB0 -d -t capture.tif
```

This will slowly download the image over the serial line. It will take about 70 seconds – you'll see why you don't use a serial link to transfer raw image data!!

If that works, open the image in an image editor. Put some text in it or whatnot to distinguish, and save it again. If you can't edit TIFF images, then use the `dod.tif` instead as in:

```
grab -P /dev/ttyUSB0 -u -t dod.tif -f
```

You should see the image displayed on the screen. Now let's look at putting other images onto the LoonBoard.

There are two blank canvases for you to use – one is 720x480, the other is 720x640. The reason behind this is that TV's have non-square pixels, they have a 4:3 aspect ratio. The 720x480 is the actual resolution, but that is with square pixels. 480 multiplied by (4/3) is 640, which is where the corrected image size comes from. This means that for the image to look right you should draw it on

the 720x640 canvas, and the resize that to 720x480 to send.

You can also set a transparency based on luminance values. To do this draw part of the image a certain colour. Then when sending the image send it with the -v option, which will display each pixel. Make a note of what one seems to occur at the end of the line, this is probably your background colour.

```
grab -P /dev/ttyUSB0 -uvf -t trastest.tif
```

The output will look like this:

```
.....  
Pixel 717 is Y: 1 Cb: 128 Cr: 128  
Pixel 718 is Y: 1 Cb: 128 Cr: 128  
Pixel 719 is Y: 1 Cb: 128 Cr: 128  
Pixel 720 is Y: 1 Cb: 128 Cr: 128
```

And the value of 1 is the transparency in this case. Set this with

```
grab -P /dev/ttyUSB0 -b 1
```

And you should see everywhere that is black disappear, with whatever video is on the input appearing there instead.

Also important is that you can't use the outer edges of the image. Most TV's overscan a bit, so part of the image is outside the visible area of the TV. Use about 90-95% of the available area and you should be safe.

The framegrabber also now supports compression of the downloaded video. This is very simple, you set the LoonBoard to skip X number of pixels and read Y number of pixels. So for example full resolution would have 0:1, which means skip 0 pixels and read 1 pixel. It's a fancy way of saying read every pixel and every line.

To scale the image by half you can set -s 1:1 to skip a pixel and then read a pixel. Note that you are NOT specifying the compression ratio itself. So 1:1 and 16:16 are both a 2 to 1 compression ratio, but will look considerably different.

```
grab -P /dev/ttyUSB0 -d -s 1:1 -t capture.tif
```

It will make the download times faster, and is great for previewing! Later releases will include more compression features, including a pallet based compression for reducing time to download images to the LoonBoard.

## **4.2 Viewing the Resulting Image**

Currently the output is a TIFF file, which not all software can read correctly. If you are using Linux support seems to be very good, almost any application will read it. You can use *The GIMP* to edit and view images.

Using Windows you can also get version of The GIMP, although it is a big download. The best application seems to be a Viewer from WildBit Software, which you can get for free at <http://koti.mbnet.fi/mhieta/software.shtml>

## **5.0 - Summary**

Hopefully this short document got you up and running with a framegrabber. There is lots of room for improvement, so you can see the potential of the LoonBoard.

## ***Document Revision History***

January 23, 2006:

- Added ability to scale image size downloads
- Fixed code to allow using -cd option all at once

December 7, 2005:

- Initial document release (Beta)

**Disclaimer:** NewAE assumes no liability whatsoever and disclaims any express, implied, or statutory warranty related to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or non-infringement. In no event shall NewAE be liable for any direct, indirect, consequential, punitive, special or incidental damages (including, without limitation, damages for loss of profits, business, interruption, or loss of information) arising out of the use or inability to use this document, even if NewAE has been advised of the possibility of such damages. NewAE makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to the specifications and product descriptions at any time without notice. NewAE does not make any commitment to update the information contained herein. NewAE's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© NewAE 2005-2006. All rights reserved.