

nae-an-003

**NewAE Application Note
#003**

Loonbus Quick-Start* Guide

*Quicker than starting a car in Winnipeg during winter anyway

Step 1 - Install Lubloader

You'll have to check the lubloader manual for this one, there's no easy step.

Step 2 - Connect up Hardware

Connect your computer to the LoonBoard, and then power up the LoonBoard. If using the lb-cable see the user manual for the lb-cable about setting up the fuse as a power switch. It will look something like this when you are done:



Step 3 - Test Hardware

From terminal run the test described in the lubloader install instructions. Make sure you are using the fuse as a power switch, you can't power down the input to the lb-cable since that would kill off the USB connection as well. See the lb-cable manual.

Step 4 - Install avr-gcc

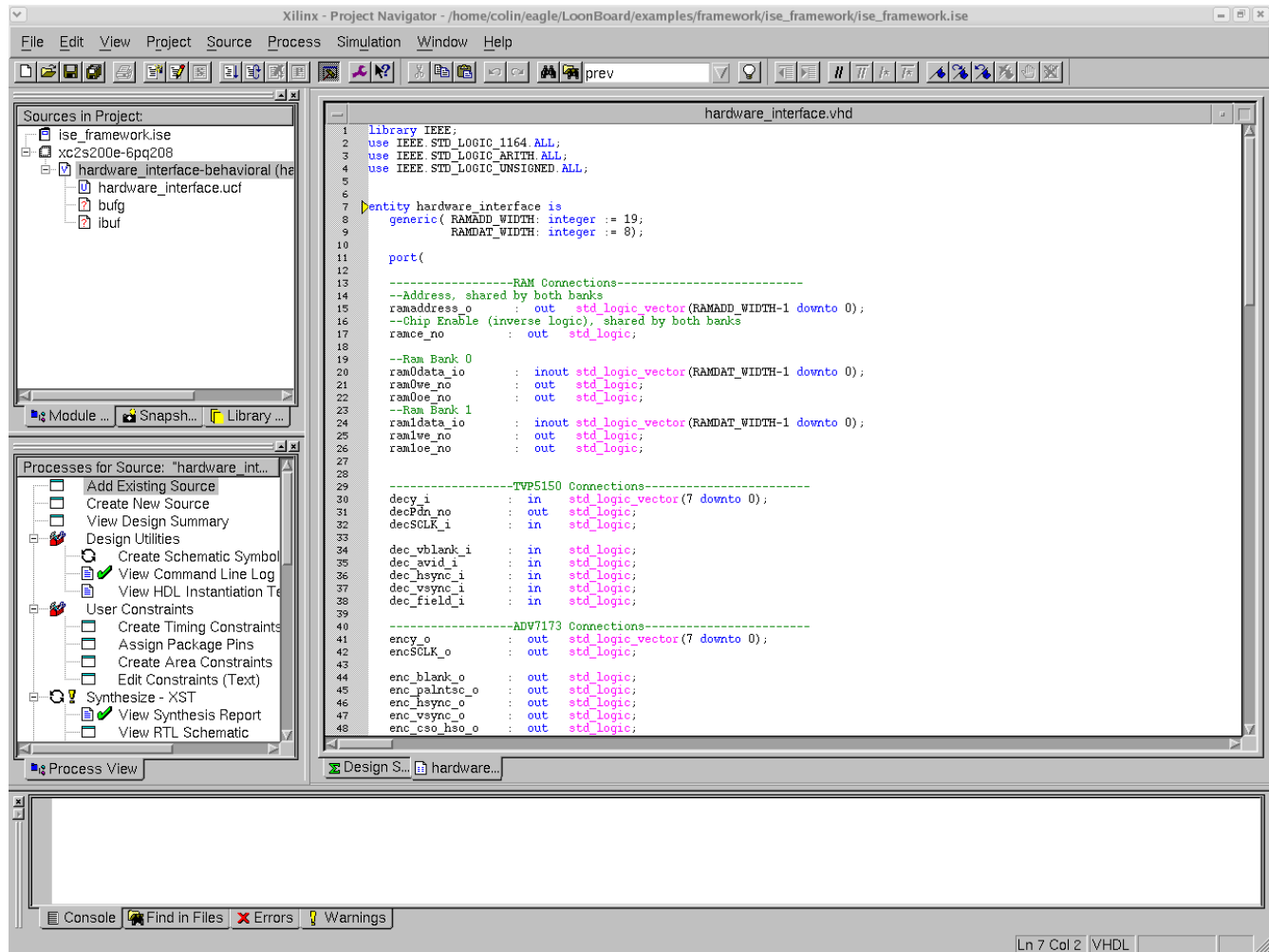
You will need a C compiler – the example are all written using avr-gcc. If you are using Windows then you can use the WinAVR distribution to get avr-gcc.

Step 5 - Install Xilinx ISE Webpack 7.1

You can get this at
http://www.xilinx.com/ise/logic_design_prod/webpack.htm

Step 6 - Open the “framework” Example in ISE

The framework example gives you an idea of what is needed to run the LoonBoard. You can get it from the NewAE website in the protected area in the forums. If you cannot access that e-mail NewAE. In ISE go to “File --> Open Project” and open the ise_framework.ise file. Double-click on the “hardware_interface-behavioral (hardware_interface.bit)” in the sources window so it opens, and your display looks like so:



You will see that all the system is doing is setting everything to initial states. As well it passes video through. Let's add a bit more to the mix, say a vertical line.

Go to <http://www.newae.com/loonboard/loonboard.html> and download the “LoonBus Interface” module. This will make the video slightly easier to work with. Extract it to the project directory with hardware_interface.vhd in it. Now add it to the project by hitting the “Add Existing Source” process then add loonbus.vhd as a “VHDL Design File”.

Now you need to connect this into the system. Find the following lines

```
--Passthrough Video
ency_o <= decy_i;
encSCLK_o <= decclk;
```

then change them to:

```
--Passthrough Video
ency_o <= loonbus_data;
encSCLK_o <= loonbus_clk;
```

```
eagl: bt656_to_loonbus port map
(
  bt656_clk_i => decclk,
  bt656_data_i => decy_i,
  video_o => loonbus_data,
  video_pixel_o => pixel_counter,
  video_line_o => line_counter,
  video_typ_o => loonbus_typ,
  video_clk_o => loonbus_clk);
```

Now you have the LoonBus data mapped in. This will give you extra information about the video as it is fed in. Then declare the signals you use by adding after the following line

```
--Buffered clock signals
signal decclk, userclk1, userclk2, sck : std_logic;
```

Add the following after that line:

```
--Loonbus Data
signal loonbus_data : std_logic_vector(7 downto 0);
signal loonbus_clk : std_logic;
signal loonbus_typ : std_logic_vector(2 downto 0);

signal pixel_counter : std_logic_vector(9 downto 0);
signal line_counter : std_logic_vector(9 downto 0);
```

Now this doesn't do anything but output the input on the output. Let's see if we screwed up anywhere first though. On the "sources in project" window make sure that the hardware-interface module is selected. Then double-click on the "Generate Programming File" process. Look at the bottom of the screen and you'll see something like this:



So it failed. Not to worry though – click over to the “Errors” tab and see what happened. Likely it is complaining that there is an “Undefined symbol 'bt656_to_loonbus'.”. This is because we forgot to include the package in our project. Go to the start of the file and add:

```
library work;  
use work.loonbus.all;
```

Again try to run the “Generate Programming File” process, and you should find it works all the way. But this will be very boring, all it does is pass the video through. Nobody wants to spend \$300 on something that can be replaced by a piece of wire, so now add some inversion.

Change the line that says:

```
ency_o <= loonbus_data;
```

to something like:

```
ency_o <= not loonbus_data when (  
    (loonbus_typ = "000" or  
    loonbus_typ = "001" or  
    loonbus_typ = "010")  
    and (pixel_counter > CONV_STD_LOGIC_VECTOR(130,10))  
    and (pixel_counter < CONV_STD_LOGIC_VECTOR(330,10))  
    and (line_counter > CONV_STD_LOGIC_VECTOR(130,10))  
    and (line_counter < CONV_STD_LOGIC_VECTOR(330,10))  
) else  
    loonbus_data;
```

This code is fairly simple. The data is equal to the inverse of the input data when the following conditions are met:

- Video is valid data (“000”, “001”, “010”)
- Video is inside an imaginary 200x200 pixel box

Now we have the FPGA done, but there's still more work! We need to program the AVR to do something. What it needs to do is set up the system to a useful state.

Step 7 – Open the main.c file

The main.c file is really all you need here, in the AVR directory. On startup the main.c file does three main things:

1: Setup the X1226

This is fairly easy to do, call the x1226_init() routine. This will try to enable random write access to the Clock and Control Register (CCR) region of the device.

2: Setup the TVP5150A

This is also fairly easy – in fact we are mainly checking that the device code matches what we are expecting. If it does then the device is present. As well enable the outputs by writing bit 3 and 0 high at address 0x03. This assumes NTSC.

3: Setup the ADV7173

We also enable the ADV7173. There are a number of default settings that must be written in, unlike the other devices. This assumes NTSC.

You will see a point in the file that says:

```
/* System is now ONLINE */
```

Which is all we need. Then add a while loop to make sure we keep the system in a known state and not just ending:

```
/* System is now ONLINE */  
while(1==1);
```

Change to the directory that the file is in, and run make at the command line. You should see some output with no errors reported. The output main.hex is your programming file.

Step 8 - Program the AVR and FPGA Files in

Now you are almost home free. It's time to program the files in, sacrifice a virgin AVR chip, and hope it works.

With your LoonBoard connected and the software installed, change directory to the level below the the directory with the AVR files and FPGA files. The run:

```
lubloader -f ise_framework/hardware_interface.bit -a AVR\  
Files/main.hex -P /dev/ttyUSB0
```

Or you can run the command to program the AVR files and FPGA files separately. You should see the LED go green if it worked. If you don't check you have a while loop at the end.

Step 9 - Testing

Hook up a video input to the video in jack, and the video output to the video out jack. You should see the input on the output with one area of the screen that is inverted.

It won't be a perfect square you will notice – it's a rectangle! This is because TV's have a 4:3 aspect ratio, this isn't a bug in the system. TV has non-square pixels, so you'll need a non-square dimensions to get a square.

In fact every other problem with the system that doesn't result in fire is a feature, not a bug (just kidding).

Step 10 - Beyond This

This very simple example should start to give you some ideas. It's very easy to do much more, as you can think of the video stream as a number of pixels and not worry about the physical stream.

Be sure to check out the other great reads:

-nae-an-002: LoonBus Interface

-LoonBoard Manual

Document Revision History

December 6, 2005:

- Initial document release

Step 11 - Not Suing NewAE

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. All rights reserved.