IDE Controller Chip Design/Programming

By Charles Hett, Lenexa Kansas

This article will describe how we came to use the particular Altera chip used in the design and an overview of how to use the Altera development software. The information presented here is not intended to be a step by step tutorial on how to use the software. There are several application notes available at the Altera website for this. Note that the software is fairly complicated and will require some effort to become familiar with it.

When designing the Z100 IDE Interface card we first built the interface with regular 'LS TTL logic. This design worked, sort of but was prone to noise and therefore was unreliable and was difficult to modify. It also took too many parts.

The next iteration was done with a MAX7000S series EPM7032S 44 pin Altera Field Programmable Logic Device (FPLD). Altera was chosen for the following reasons:

1. Parts were available from Digikey and Newark.
2. Development software was free.
3. I had a friend at work who was familiar with Altera products and software.
4. Functionality met the requirements we needed.

This design also worked but was still prone to noise and unreliable operation because there weren't quite enough pins available to do the things that we determined were needed: fully decode the I/O address space and provide separate buffer control for the two IDE connectors.

So the next and final iteration was done with an EPM7064S 84 pin Altera chip which provided enough functionality for us to do everything we needed and a little bit more. This chip was more expensive but it eliminated several 'ALS parts and sockets plus the design finally worked reliably.

The development software used was the `Quartus® II Web Edition Software` available at www.altera.com. This software is free but a license is required which will need to be renewed every six months. This is all explained at the Altera website.

The steps are: download the software, obtain a license, and enter the design.

There are various ways to enter the design. We chose to enter by schematic but we could have entered it by means of the VHDL language. None of us were familiar with VHDL so that method was not chosen. *VHDL* is an acronym for Very High Speed Integrated Circuit Hardware Description Language which is a programming language used to describe a logic circuit by function, data flow behavior, or structure.

Our design is fairly straightforward and John Beyers improved the design by using the Z100's PHI clock to enable the edges of the strobes.

Choosing the input and output pins is an important step. Which pin is assigned to which function is primarily a function of board layout. Most of the pins of the EPM7064S can be programmed as either inputs or outputs.

Special attention should be paid to the use of pins 1, 2, 83 and 84. These pins can be inputs or special functions GCLRn (1), GCLK2 (2), OE1 (84) and GCLK1(83). We used 83 for S100 PHI (Clock) and 1 for RESET. Pins 14, 23, 62, 71 are reserved for programming the chip.

The major option we used was fast or slow slew. Slow slew added about 5ns to the rise time of the signals which reduced board noise. After going to fully buffered outputs, it was no longer necessary to set this to slow slew which may allow the board to operate at a slightly higher clock speed. The board was extensively tested at clock speeds of 8MHz and 10MHz. It may work at speeds higher than that, especially when appropriate wait states are added but do so at your own risk. See the description of wait states in the companion article that describes the overall design.

When the design has been entered you proceed to the compile phase. If compilation is successful (no errors but you may get warnings which may or may not be ignored as in our design) then you may simulate the design or program your chip.

To simulate, you need to tell the software what inputs you want and outputs you want. This is done by means of a vector file. Then run the simulation and the waveform timing is displayed similarly to how it would appear on an oscilloscope.

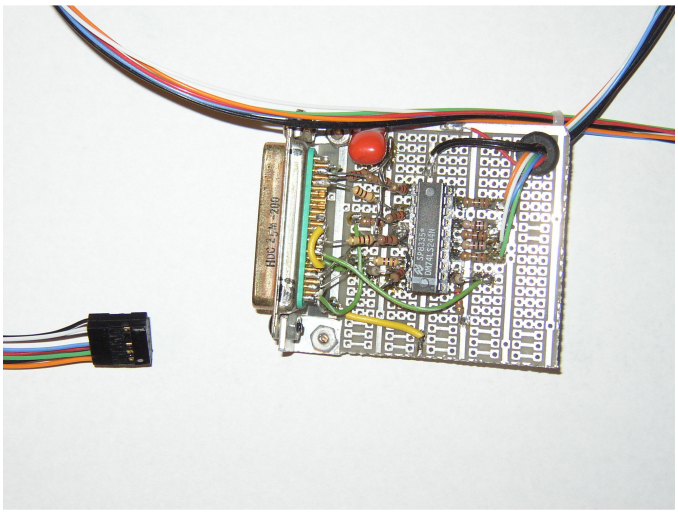
Programming

Once you are happy with the simulation, or have skipped it, it is time to program the chip. The Altera Byte Blaster II programmer was built. Altera has a note on how to do this.

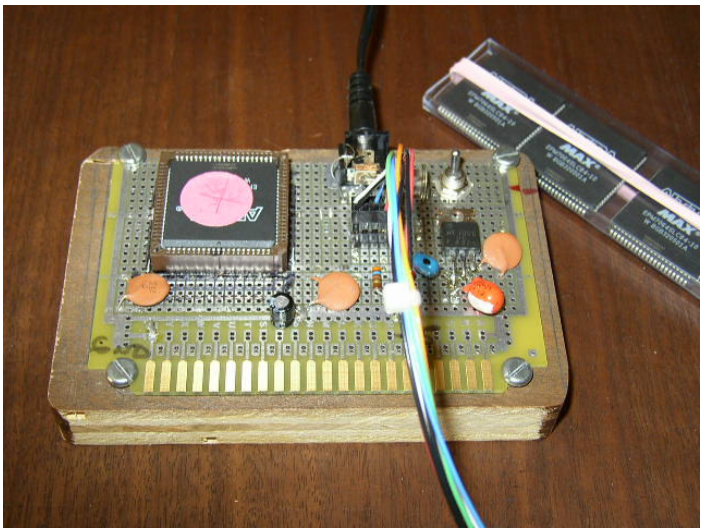
http://www.altera.com/literature/ug/ug_bbii.pdf

Note: use the JTAG connections shown in Table 2-2 of that note. It is basically a buffer for the four Programming signals to and from the Altera chip connected to a PC Printer Port. A ribbon cable with standard header connectors to connect the programmer to the Z100 IDE Interface board is recommended.

Here is a photo of the programmer interface board.



The IDE Interface board is designed so that the controller chip can be programmed while installed in the Z100. I never actually did that. Most of the time, I preferred to program the card while plugged into a special card test rig I built. Later on, I built a special board for programming that had a PLCC socket so that it would not be necessary to subject the interface boards to the wear and tear of multiple chip insertions/removals. The Interface board or the socket board requires 5v power. This powers both the Altera chip and the programmer at the PC. I powered this special socket board with a wall wart module. I put a 5v three terminal regulator on the board for safety and more reliable operation. Here is a photo of this special socket board.



If all has been done correctly, you can now program the chip. Setup the Altera software for the Byte Blaster programmer and begin programming. You might first want to do a blank check verify (if you are programming the chip that came with the board it shouldn't be blank). Then program. The software will give you a progress status bar and say programming successful when it is complete. Then you will want to verify the results.

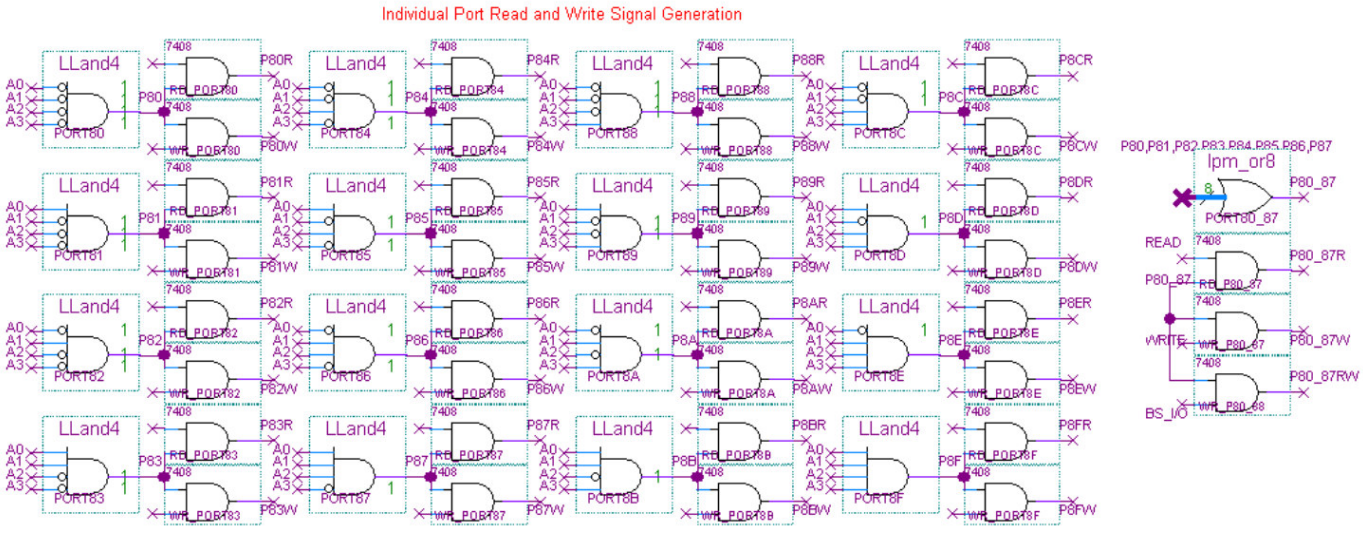
Note: if you do not intend to change any I/O pins, you can program the chip when installed in the Z100. This is a convenient way to provide power to things. If you do change I/O pins for any reason, remove any chips that connect to the Altera chip. If you accidentally get an Altera output connected to another output it might damage the Altera chip or the other chip or both.

It is highly recommended that you obtain an FPLD extractor tool OK Industries EX-6 Digikey K374-ND (about $11) or equivalent if you intend to remove the chip from the socket for any reason. It is possible to do it with a scribe or similar tool but you run the risk of damaging the socket or the chip.

Chip Circuit Description

Note: The following description does not address each schematic in detail. Several details are embedded in the Altera design software. The intent here is to give an overview of the design.

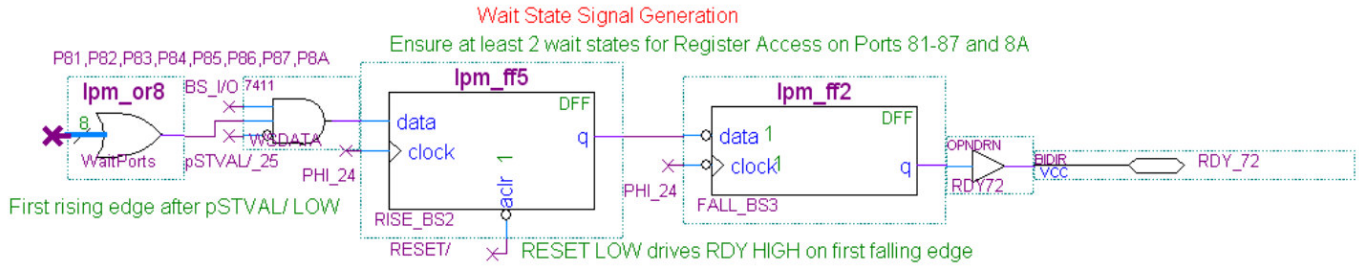
Address Decoding



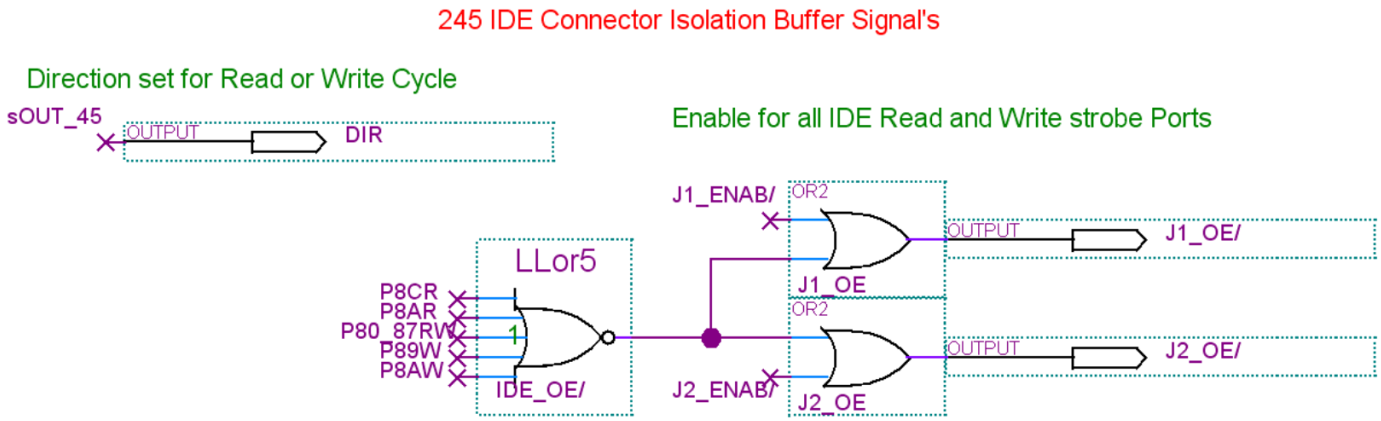
Sixteen read and sixteen write selects are decoded. For example, port 80 read is indicated as P80R. port 80 write is indicated as P80W

Wait State Generation

Two wait states are generated by this circuit. They are not programmable. The software design includes wait state programmability but it is not required for the production hardware. Wait states were programmable on early prototypes.



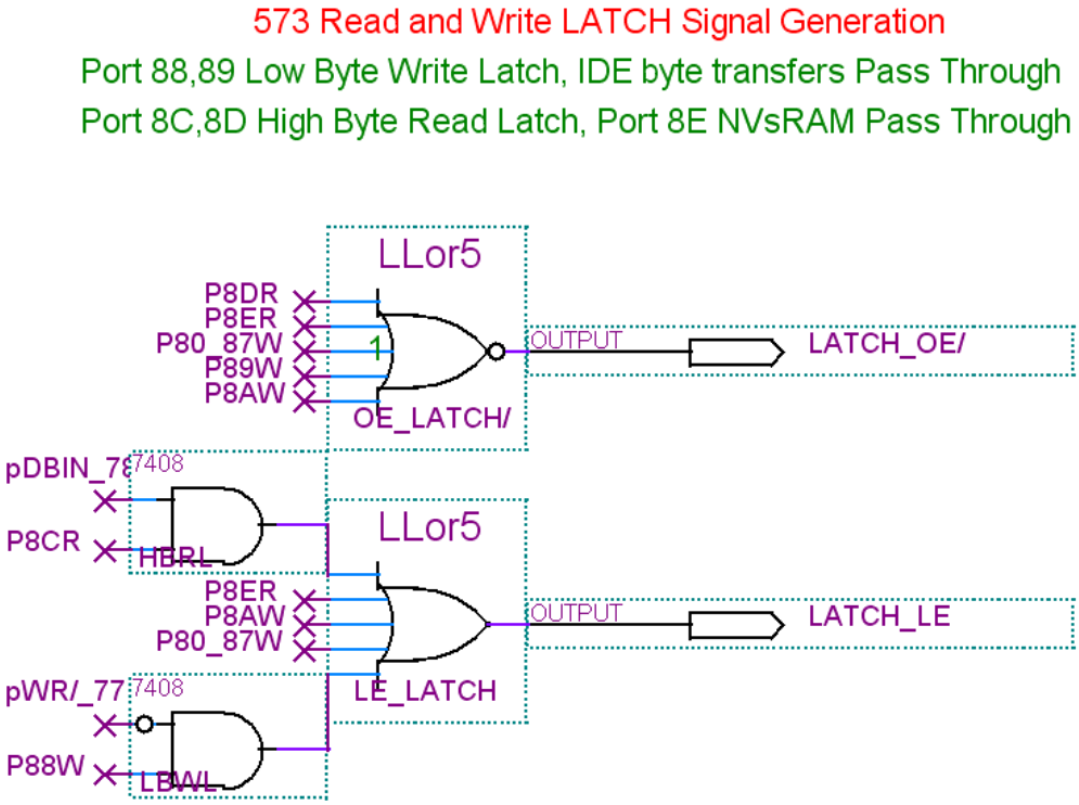
IDE Connector Isolation Buffer Signals



This section provides the J1 and J2 IDE Output Enable controls for read and write operations.

A Data Direction control pass through is also shown.

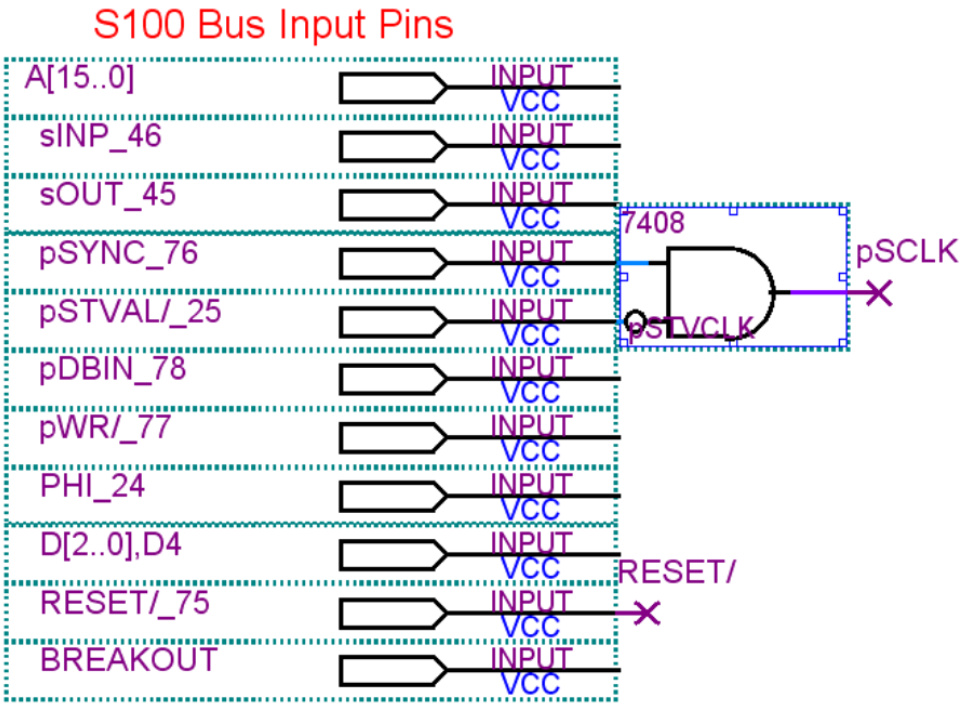
Read and Write Latch Signal Generation



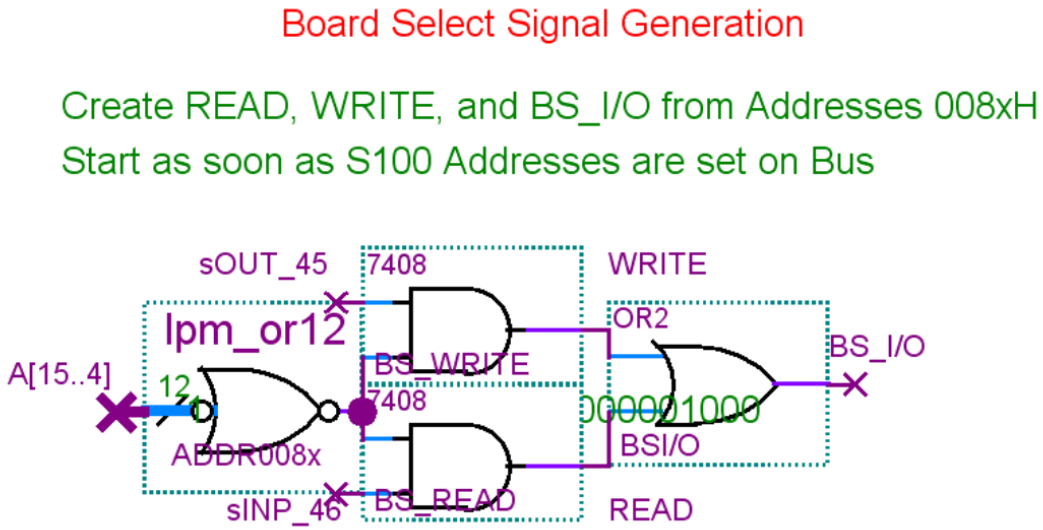
This section generates the read/write latch output enable and latch enable signals.

S100 Bus Input Pins

Simply defines these pins for use elsewhere in the chip.

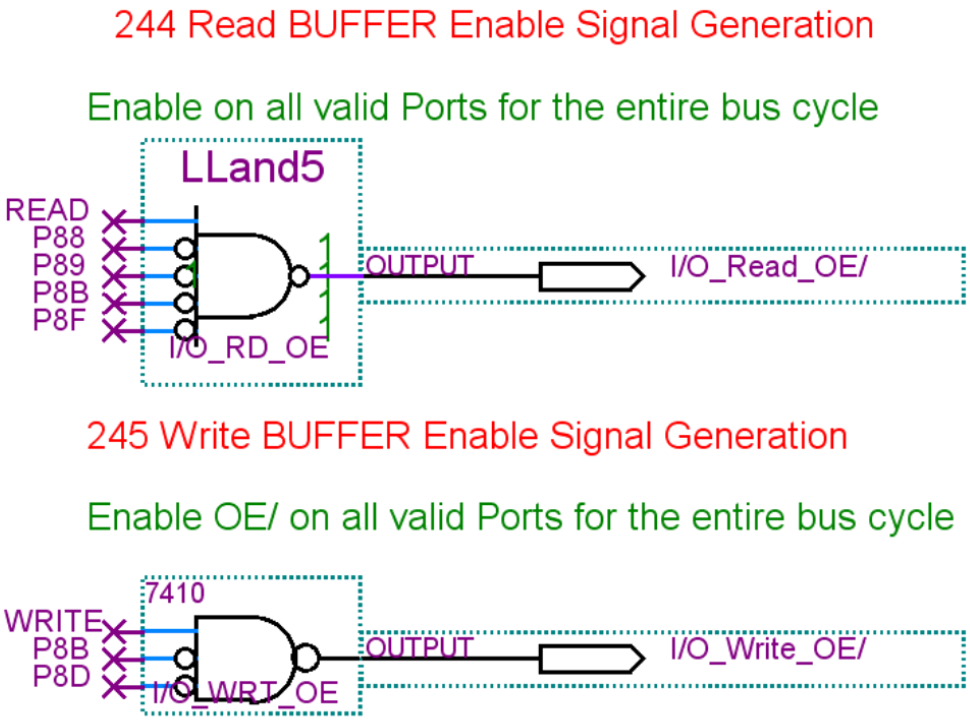


Board Select



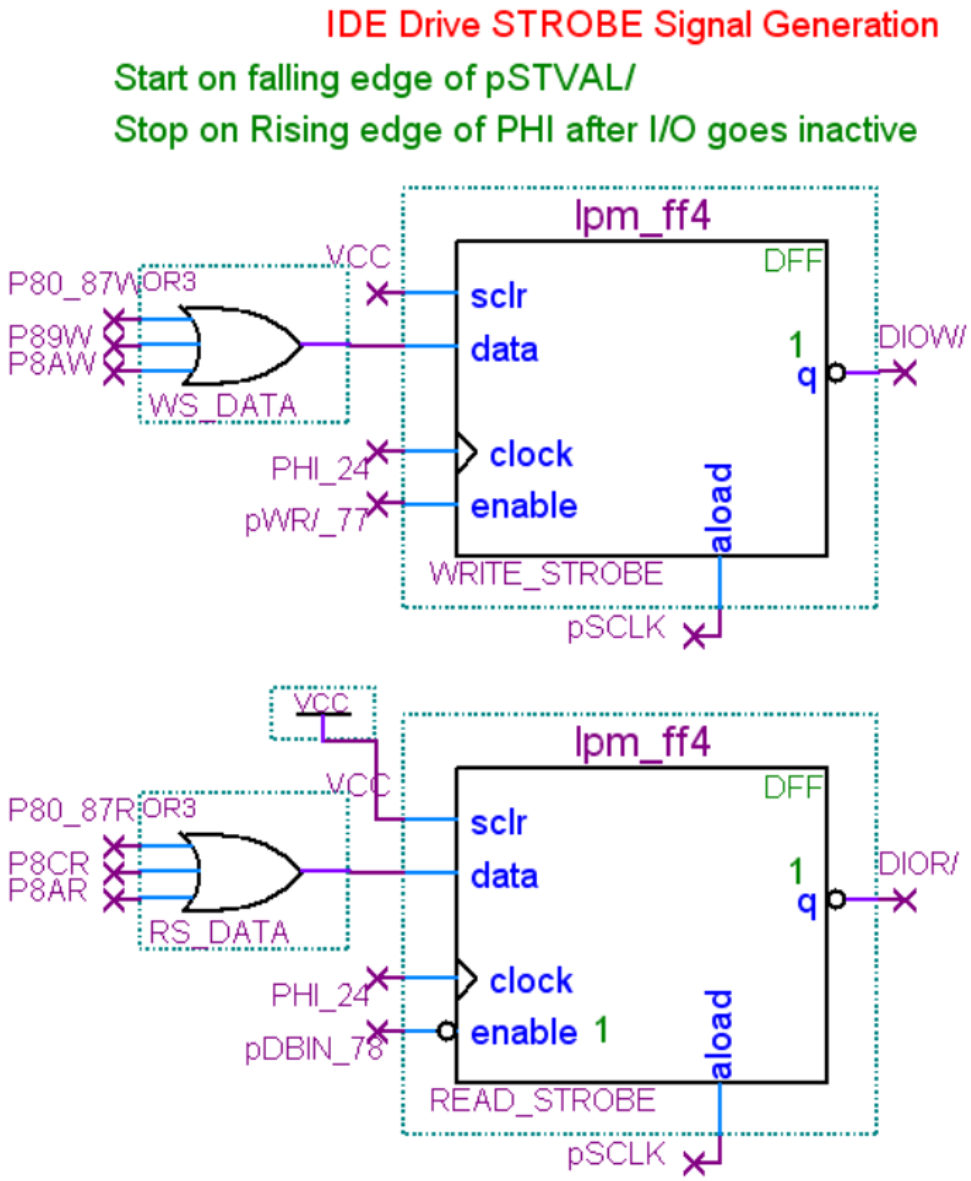
Generates the on chip board select signal.

Read/Write Buffer Enable



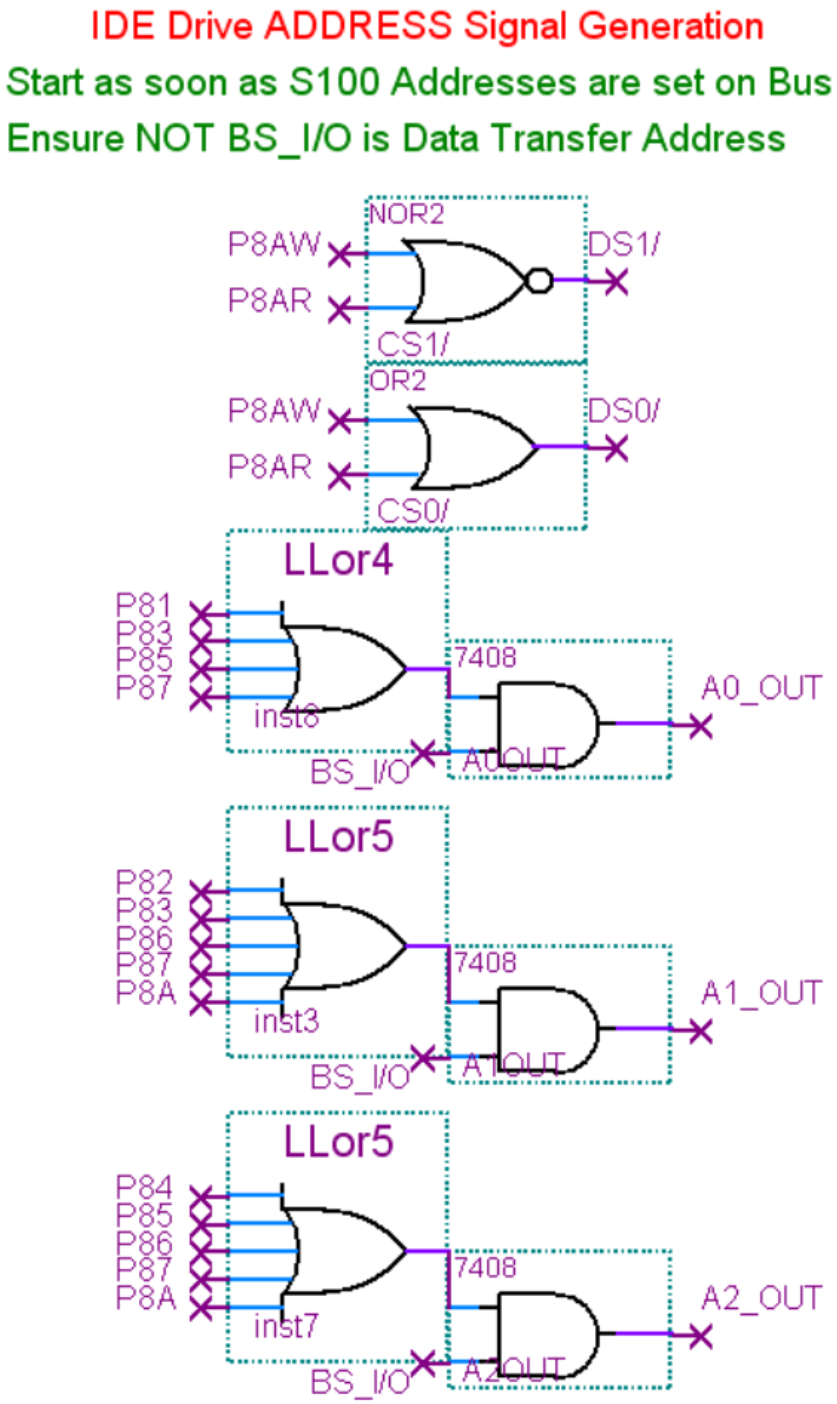
Generates IO read and IO write output enable signals.

IDE Drive Read/Write Strobe Signals



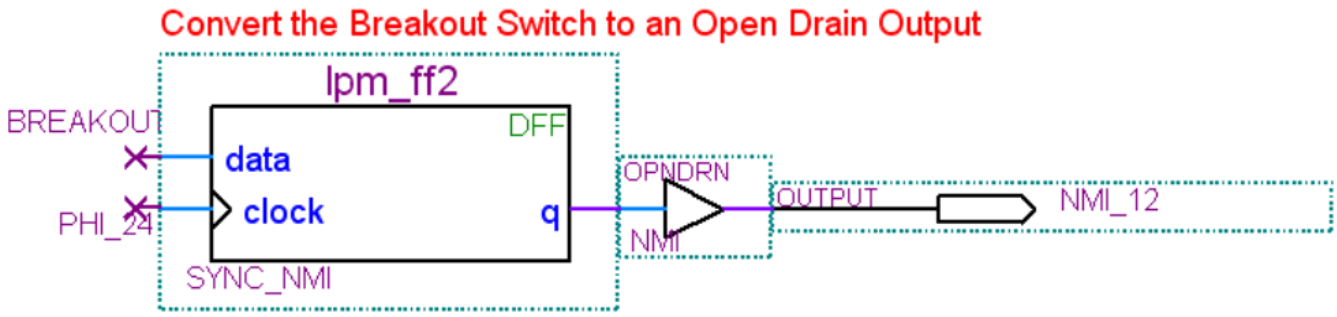
Generates drive IO Read and IO Write strobe signals

IDE Drive Address

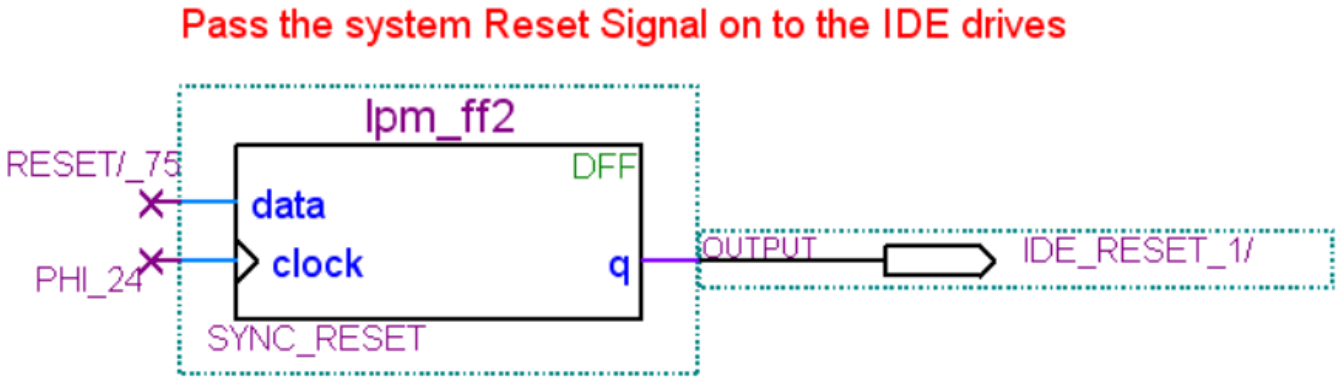


Generates the Drive Select DS0, DS1 and A0, A1, A2 outputs.

Breakout Switch Conversion

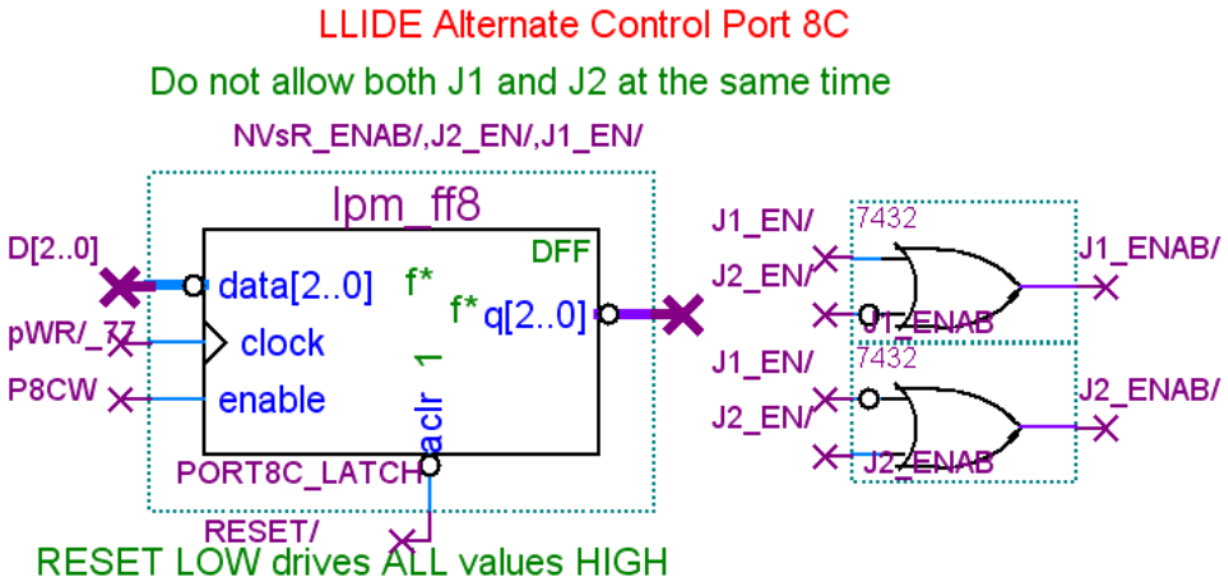


System reset to IDE

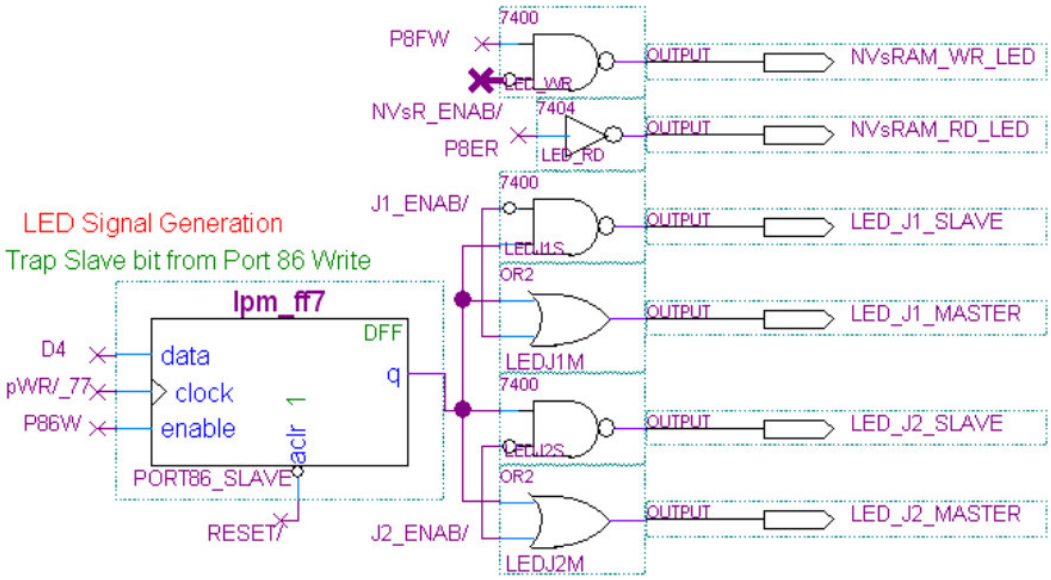


Pass through via flip flop for proper deglitching

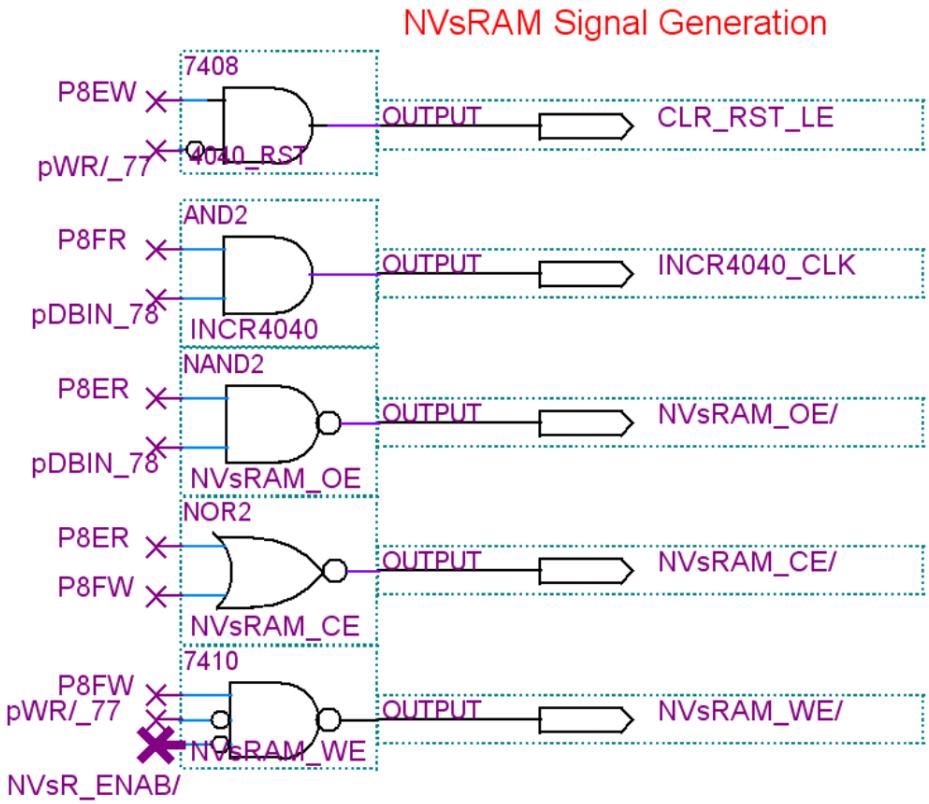
IDE Alternate Control Port Enable



Drive Activity LED Signal Generation



NVsRAM Controls



IDE J1 and J2 Control Signals