# ISP™ Solution for Plug-and-Play

## Introduction

As PC systems migrate from the desktop to the laptop, add-on hardware configuration becomes even more challenging. The concept of Plug-and-Play has been advanced, making hardware configuration a software controlled function. Plug-and-Play employs configuration software to automatically configure add-on hardware so it will coexist with the rest of the system. A Plug-and-Play compliant system eliminates the need for user manipulation of add-on hardware switches or jumpers prior to card installation. Lattice Semiconductor's In-System Programmable™ (ISP) technology allows Plug-and-Play systems to be implemented with ease.

The traditional method to upgrade a PC with add-on cards is subject to human error and usually becomes a frustrating process of trial and error. Before installation, the PC must be powered off and partially disassembled. Then, each add-on card's unique address and interrupt levels must be set manually set by mechanical DIP switches or jumpers. Finally, the card is installed and the PC is powered up. If the first attempt is unsuccessful, the user must remove the card, reread the manual, reset the jumper or DIP switch, plug the card back in, and hope that the jumpers are set correctly. If not, the process is repeated. Mechanical components are also historically unreliable, further complicating the procedure. When the user is unsuccessful, frustration results along with lost time and effort.

## ispGDS™ Solution

The solution to this problem is provided by Lattice's family of in-system programmable Generic Digital Switches (ispGDS), which easily replace mechanical DIP switches and jumpers. At 14, 18 and 22 outputs, even the smallest ispGDS device contains enough I/Os to replace up to two seven-bit wide DIP switches or seven jumpers. Each ispGDS output can either be set to a logic '1' or '0' to emulate a DIP switch. In addition, the ispGDS I/Os are equally divided into two banks: bank A and bank B. Any input in bank A can be connected to any or all outputs in bank B and vice-versa, effectively emulating mechanical jumpers. Jumper emulation allows devices to act as a cross switch matrix, providing 7x7, 9x9 and 11x11 matrix solutions. Furthermore, as semiconductors have no moving parts, they are more reliable than their mechanical counterparts.

## System Implementation of Plug-and-Play

Although ispGDS devices can be configured either by in-system programming or by using industry standard PLD programmers, in-system programming better satisfies the Plug-and-Play requirements of the add-on card. By programming the ispGDS devices "in-system," add-on cards can be inserted into the PC without having to manually set any DIP switches or jumpers; the configuration software for the add-on card can set the card's address and interrupt configurations. The ability to program the card using 5V TTL level signals, combined with the ispGDS C language routines from Lattice, provides the user with an easy approach to implement Plug-and-Play compliant add-on cards.

There are two system design problems to consider when programming ispGDS devices in-system. First, the ispGDS device must have its own address space so it can be addressed and programmed anywhere on the PC bus. Unfortunately, most PC addresses are already dedicated for specific purposes per the IBM PC specification. However, some addresses are specified as read only, which means they are uncommitted for write functions. For example, the address for the game port (200-207) is specified as read only. By using an external GAL® device to decode the ispGDS address to 200-207 (write only), the ispGDS can be programmed across the PC bus.

The second problem occurs when two or more PC bus add-on cards have ispGDS devices residing at the same write address. If the add-on card configuration software tries to program an ispGDS device on one of these cards, the ispGDS devices on the other cards are also addressed and programmed. This problem is remedied by using the logic in the interface GAL device and the ISP feature of the ispGDS device. Take, for example, a card with three daisy-chained ispGDS devices on it (see Figure 1). By assigning three of the outputs of the first serial ispGDS device (as shown by GDSSEL0..2 in Figure 1) the card's ispGDS write address can be moved within the address space. ispGDS devices come from the factory with their outputs set to the high impedance state. By using external pull-up resistors on the three ispGDS outputs dedicated for the ispGDS write address space, the three Least Significant Bit (LSB) outputs default to 111. Hence, straight from the factory, the ispGDS address space can be made to default to address 207. When the add-on card is inserted into the PC for the first

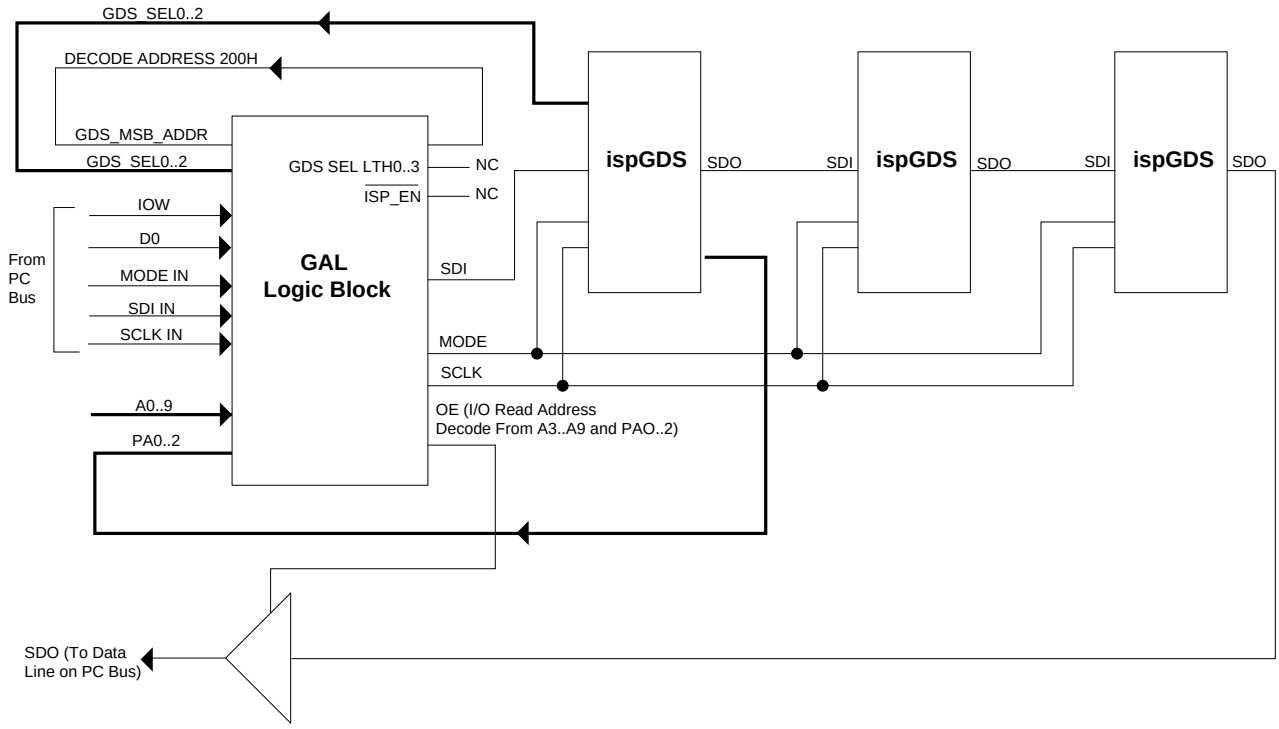
time, the configuration software finds an unused write address, other than the default 207, and programs the ispGDS chain address to this address. This is accomplished by programming the ispGDS devices with a JEDEC fuse map, in-system, via the PC bus. This sets the dedicated ispGDS address to the desired write address value. Once the ispGDS address has been set, the address of the card, interrupt levels, and other components can be set in-system by downloading another JEDEC fuse map.

The block diagram (Figure 1) and the ABEL example file that implements the GAL decoder logic are included with this application note. The ispGDS C source code is provided by Lattice. For a detailed explanation of this source code, please consult the ispCODE Reference Manual. Using the ABEL example file, it is easy to implement ISP on the ispGDS. Simply modify the ispGDS C source code to control data bit 0 across the PC bus; in the GAL decoder logic provided, data bit 0 acts like an ISP enable signal when written to the ispGDS write address.

## Conclusion

The DIP switch and jumper approach is obsolete for Plug-and-Play add-on cards. Using minimal decoder logic, the ispGDS C source code provided by Lattice, and the in-system programming capability, ispGDS devices can provide an effective Plug-and-Play solution.

**Figure 1. ispGDS Interface Block Diagram**



**Listing 1. Decoder Logic ABEL Source File**

```
module gdsdcode

title ‘gds isp controller/decoder’;

gdsdcode  device  ‘p20v8’ ;

iow pin 1;

a2,a1,a0 pin 2,3,4;
lsb_addr=[a2,a1,a0];

gds_msb_addr pin 23;
”a9,a8,a7,a6,a5,a4,a3 pin 5,6,7,8,9,10,11

gds_sel2,gds_sel1,gds_sel0 pin 5,6,7;
gds_sel=[gds_sel2,gds_sel1,gds_sel0];

gds_sel_ltch2,gds_sel_ltch1,gds_sel_ltch0 pin22,21,20;
gds_sel_ltch=[gds_sel_ltch2,gds_sel_ltch1,gds_sel_ltch0];

gds_addr pin 19;

isp_en pin 18 istype ‘reg_d,invert’;

d0 pin 8;

mode_in   pin 9;
mode   pin 17 istype ‘reg_d,invert’;

sdi_in   pin 10;
sdi      pin 16  istype ‘reg_d,invert’;

sclk_in   pin 11;
sclk      pin 15 istype ‘reg_d,invert’;

“gds_msb_addr=a9 & !a8 & !a7 & !a6 & !a6 & !a5 & !a4 & !a3
gds_lsb_addr=!((gds_sel_ltch2 $ a2) # (gds_sel_ltch1 $ a1) # (gds_sel_ltch0 $ a0));

equations

gds_sel_ltch= gds_sel & !isp_en
            # gds_sel_ltch & isp_en;

gds_addr=(gds_lsb_addr & gds_msb_addr);

isp_en.d=isp_en & !gds_addr
            # d0    &  gds_addr;

isp_en.clk=iow;

mode.d= mode & !gds_addr
            # (isp_en & mode_in) & gds_addr;

mode.clk=iow;

sclk.d= sclk & !gds_addr
            # sclk_in & gds_addr;

sclk.clk=iow;

sdi.d= sdi & !gds_addr
            # sdi_in & gds_addr;

sdi.clk=iow;

end;
```



LATTICE SEMICONDUCTOR CORPORATION
5555 Northeast Moore Court
Hillsboro, Oregon 97124 U.S.A.
Tel.: (503) 681-0118
FAX: (503) 681-3037
http://www.latticesemi.com

November 1996