Erasable Programmable Logic Device

Application Note

Using Programmable Logic Devices

Introduction

This application note covers three areas:

- Where and *why* do I use Programmable Logic Devices (*PLD*s)?
- *How* do I use PLDs?
- Software and hardware *support* for Atmel PLDs.

Where Do I Use PLDs?

Any digital logic design can be done using PLDs. If you normally begin your design by:

- Using AND and OR functions
- Thinking of 7400 series components
- Using truth tables, or
- State diagrams

You are already on the path to using PLDs.

Designing a microprocessor-based system, with memory and I/O? How about all that “glue” logic you use to interface with the bus, provide chip selects, and any unusual signals required by special chips? Most of these functions are currently done with 7400 series TTL. *How about using a PLD instead?*

Designing a stand-alone PC board which uses a state machine to control multiple output signals? Using latches to synchronize signals? Using counters to divide down master clock frequencies? Converting parallel-to-serial and back again? All of these functions fit easily in modern PLDs. *Most anything found in your TTL Databook can be replaced with your own, PERSONALIZED, programmable logic device.*

PLD Applications

- **Glue Logic**
- **State Machines**
- **Synchronization**
- **Decoders**
- **Counters**
- **Bus Interfaces**
- **Parallel-to-Serial**
- **Serial-to-Parallel**
- **Subsystems**
- **and Many Others**

Why PLDs?

Maybe you have already heard all the wonderful reasons for using PLDs. Well, they’re true! First, let’s review some of the more important ones:

- **Increased Integration.** You can reduce the package count of your designs while simultaneously increasing the features offered by your product.
- **Lower Power.** CMOS and fewer packages combine to reduce power consumption.
- **Improved Reliability.** Lower power plus fewer interconnections and packages translate into greatly improved system reliability.
- **Lower Cost.** PLDs reduce inventory costs, too.
- **Easier To Use**! Yes, believe it or not, once you get past the initial learning period, PLDs are easier to use than discrete logic functions.
- **Easier to Change.** Oops! Need to make a change? You won’t need “blue wire” when you use a PLD – all changes are internal,and can be done quickly. ECNs are a snap – and system reliability is maintained!

Rev. 0485C–09/99

Let's Get Started!

Figure 1 describes the PLD design process. After having read the first part of this application note, you now have the perfect application for a PLD, right? So here you go!

How do you translate your idea into a working prototype? *First, you need a computer with an editor of some kind.* If you have a workstation with a schematic editor, you may input your design using familiar logic blocks. Otherwise, a line or full screen text editor, used in the non-document mode will do. An example of an ABEL™ text file is on the next page.

Next, turn the **logic compiler** loose on your design. First it will *check for typographical errors* and any inconsistencies in your specification. Most compilers then attempt to *reduce your logic* using standard logic reduction theory. Then, a **simulator** will check the test vectors you input, comparing your logic description against the predicted responses. This is an excellent way to verify your design. Check with the appropriate software manuals for more information.

At the end of the compilation process, a *JEDEC* file is output. This file is a standard format accepted by most programming hardware. Next *download this file* to your chosen programmer.

At this point you are ready to "build" your **prototype**. Make sure the programmer has the correct information to program the device you have chosen (an Atmel PLD, of course), plug in your device, and go! Most programmers will even functionally test your prototype for you if you include test vectors in your JEDEC file.

Take your configured PLD, and *plug it into your system.* If you find any errors, just use your editor to make the necessary changes, and repeat the process. It's easy!

Example Design

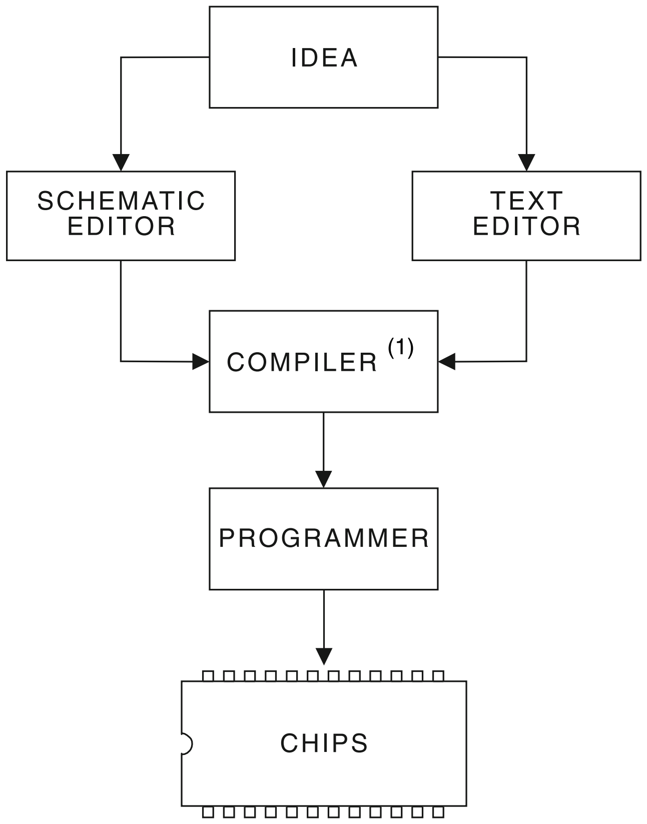
The following design is a simple example using ABEL™ to process the logic description file and an AT22V10 as the target device. The equations are on the next page, and are a direct reproduction of the actual ABEL input file.

Each of the three allowable input formats are shown. A truth table is used to describe a simple 2-to-4 decoder, as is often used to decode chip selects in a microprocessor system. Next, the state machine format is used to describe a divide-by-4 counter. And finally, Boolean equations are used to describe some random logic.

Note the test vectors used to test the device. The "c" nomenclature means that this pin has a low-to-high-to-low series of transitions for this vector. Each time this happens, the counter should increment. Also note that the counter starts in the reset condition, which is both outputs "1" for an active low output.

Now you're ready to go - Have fun!

Figure 1. PLD Design Process



Note: 1. Examples of compilers are ABEL, CUPL™, PLDesigner XL™, nd LOG/iC. Each of these products contains modules which allow simulation of your design. They also minimize your logic equations, which gives you flexibility in describing your design.

Example ABEL Description File

```
module X3;
title ’Example using 22V10 - KHG 1/6/88’;
X310 device ’P22V10’;
“
Clk,A12,A13               pin1,2,3;
CE0,CE1,CE2,CE3           pin20,21,22,23;
Q1,Q2,CarOut              pin17,18,14;
CarEn,A,B,C,D             pin6,7,8,9,10;
Out1,Out2                 pin15,16;
“
X,Z,c                     = .X. , .Z. , .C.;
“
“Counter States
State1                    =  ^b00;    State2   =  ^b01;
State3                    =  ^b10;    State4   =  ^b11;
“
“The following truth table defines the 2 to 4 decoder, which decodes
“ A13 and A12 into CE0, CE1, CE2, and CE3.
truth_table ([A13,A12] -> [CE0,CE1,CE2,CE3])
[ 0, 0 ]  ->  [ 0, 1, 1, 1 ];
[ 0, 1 ]  ->  [ 1, 0, 1, 1 ];
[ 1, 0 ]  ->  [ 1, 1, 0, 1 ];
[ 1, 1 ]  ->  [ 1, 1, 1, 0 ];
“The following state description defines the divide by 4 counter
state_diagram [Q2,Q1]
State State1:   GOTO   State2;
State State2:   GOTO   State3;
State State3:   GOTO   State4;
State State4:   GOTO   State1;
“ The following equations are general in nature to illustrate Boolean input
“ format.  The CarOut equation uses state 4 above to produce a carry.
Equations
CarOut                    = Q2  &  Q1  &  CarEn;    “& = AND
Out1                      = A &  B +  C &  D;  “+ = OR, AND takes precedence
Out2                      = A &  C +  B &  D;
“The following are the appropriate test vectors
test_vectors
“
([Clk,CarEn,A13,A12,A,B,C,D]- >[CE0,CE1,CE2,CE3,Q2,Q1,CarOut,Out1,Out2]);
[ 0, 0, 0, 0, 0, 0, 0, 0 ] - >[ 0, 1, 1, 1, 1, 1, 0, 0, 0 ];
[ c, 0, 0, 1, 1, 1, 0, 0 ] - >[ 1, 0, 1, 1, 0, 0, 0, 1, 0 ];
[ c, 0, 1, 0, 1, 0, 1, 0 ] - >[ 1, 1, 0, 1, 0, 1, 0, 0, 1 ];
[ c, 0, 1, 1, 0, 0, 1, 1 ] - >[ 1, 1, 1, 0, 1, 0, 0, 1, 0 ];
[ c, 0, 0, 0, 0, 1, 0, 1 ] - >[ 0, 1, 1, 1, 1, 1, 0, 0, 1 ];
[ 0, 1, 0, 1, 1, 1, 1, 1 ] - >[ 1, 0, 1, 1, 1, 1, 1, 1, 1 ];

end X3;
```

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel U.K., Ltd.
Coliseum Business Centre
Riverside Way
Camberley, Surrey GU15 3YL
England
TEL (44) 1276-686-677
FAX (44) 1276-686-697

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

0485C–09/99/xM