Logic Design - PAL Implementation

The design of microprocessor systems usually includes the design of "glue" logic to interconnect the microprocessor with memory and peripheral interface chips. This lecture describes logic implementation using a simple programmable logic device (PLD) called a PAL (Programmable Array Logic).

Programmable Logic

The choice of logic implementation technique (standard logic function ICs, programmable devices or custom ASICs), will depend on factors such as the number of units to be built, engineering expertise available, the available design tools and the time allocated for the design. Current practice is to use programmable logic devices (PALs and FPGAs (Field Programmable Logic Arrays) to implement most custom logic functions except on very high volume designs. The advantages of using PLDs are reduced parts count, shorter design time, reduced inventory and ease of modification.

There are a wide variety of programmable logic devices available including PALs, CPLDs (complex programmable logic devices) and FPGAs (field-programmable logic devices). PALs are limited to a simple sum-of-products architecture with 8 to 10 inputs and outputs. PALs are widely used to implement simple combinational logic circuits and state machines. CPLDs typically include multiple PAL-like elements in the same chip with programmable interconnections between them. They often allow more than one clock input and can implement several independent functions. FPGAs typically include hundreds of very simple logic blocks (e.g. a programmable 4-bit sum-of-products logic block with a 4-bit register) with very flexible interconnection arrangements. It should be noted that different manufacturers have different many interpretations of what these terms mean. One manufacturer may call a certain type of device a CPLD and another may call a similar device an FPGA.

PAL Architecture

Each output on a PAL implements a programmable sum-of-products function of it's inputs. The maximum number of terms in the sum depends on the device. For the 18CV8 device we will use as an example, there can be up to 8 terms. Each product term can include any combination of the inputs and their complements.

Registered PALs have D flip-flops on their outputs with the sum-of-product results driving the inputs to the flip-flops. This allows registered PALs to implement arbitrary state machines. Modern PAL devices can mix registered and combinational outputs and can often be programmed to .

PALS are thus well suited to implementing the simple combinational circuits and state machines that we have seen in previous lectures.

PAL Design

PALs are usually designed using simple hardware description languages that describe the logic function to be performed. An "assembler" then translates this functional specification of the device into a programming file ("JEDEC" file). This file is then used to program the device using a device programmer similar to an EPROM programmer.

discrete logic	PLD
specification	specification
minimization	run assembler
schematic	JEDEC file
connect ICs	program device

Design using CUPL

We will use the CUPL programmable logic language since it is widely used and is available on the EE Sun workstations. Other popular "languages" include ABEL and PALASM.

A complete study of CUPL's features would take several lectures so here we will only cover the minimum subset required for our relatively simple designs.

The CUPL input file uses the .pld file name suffix by convention. Comments can be inserted anywhere in the file using /* and */ delimiters. The input file contains three sections:

The first section includes a number of statements to identify the design. The last statement of this section gives a code to identify the device we will use. In this course we will use P18CV8. Note that each statement ends with a semicolon.

The second section of the file uses PIN statements to assign symbolic names to the input and output pins. The 18CV8 is a 20-pin package with 10 inputs and 8 outputs. Pins 1 to 9 and 11 are inputs while pins 12 to 19 can be used as inputs or outputs (see data sheet). If a clock input is used it must be on pin 1. Variable names are case sensitive and should begin with a letter.

The final section gives the logic equations that define the outputs (or register inputs) as functions of the inputs. The operators !, &, # correspond to NOT, AND and OR. If an output variable is to be a registered output, the `.D` extension must be added to the output variable name so that the expression applies to the D input of the output flip-flop. Registered designs often include an input to clear or preset the registers to a known state when the device is first powered up. On the P18CV8 this is done using the `.SP` (synchronous preset) and `.AR` (asynchronous reset) extensions on output variables.

The command[1]

```
cupl -ajlxn example1.pld
```

will compile the file `example1.pld` and produce listing (`.lst`) and documentation (`.doc`) files, and an `.abs` file to use with the CSIM simulator (see below). Other types of outputs can be generated (use the cupl command with no arguments to see the possible options).

[1] only available on the EE Sun-4 (SPARC) machines.

Simulation using CSIM

Since PAL logic design can be error-prone, a simulator is almost always used to test PAL designs without having to program devices and then put them in a circuit. The input to the simulator is a description of the PAL logic (for CUPL this is the `.abs` file) and a set of test vectors. Each test vector specifies a set of inputs to the PAL and the expected outputs. The simulator compares the expected output to that which would be produced by the logic described in the `.abs` file. The simulator output file notes any discrepancies between the expected and simulated outputs.

The simulator input file is also divided into three parts. The first part is the identification section and is identical to that required for CUPL.

The second part of the file uses the ORDER command to define the input and output variables that are to be included in the test vectors and the order in which they appear in the test vectors and in the output. As shown in the example, spaces may be inserted in the simulation output by using a percent sign followed by the number of spaces.

The VECTORS: line indicates the start of the test vectors. The test vectors appear with one vector per line (and no terminating semicolon). The values of the input variables are specified as 0, 1, X (don't care), C for a positive-going clock input or K for a negative-going clock edge. The values of the outputs are given as the logic levels H or L.

The command

```
csim -jln example1
```

will run the simulator using the simulation input file `example1.si` and produce a simulation output file `example1.so`.

CUPL and CSIM Examples

The listings show the CUPL and CSIM input and output files for the previous combinational and sequential logic designs.