

Logic Design - PAL Implementation

Control systems often include digital logic circuits either to implement the controller or to interconnect the micro-processor with peripheral interface chips. This lecture describes how to implement digital logic using a simple programmable logic device (PLD) called a PAL (Programmable Array Logic).

After this lecture you should be able to use the CUPL logic design language to design a simple PAL-based circuit.

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 which use ASICs.

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 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 its inputs. The maximum number of terms in the sum depends on the device. For the 16V8 device we will use in the lab

example, there can be up to 8 terms. Each product term can include any combination of the inputs, the outputs 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.

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 logic design 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.

Design using CUPL

We will use the CUPL programmable logic language since it is reasonably popular and a version that supports the GAL16V8 device that we will use in the lab is available for free. Other popular “languages” for PLD design include ABEL and PALASM. FPGAs are usually designed with one of two more powerful hardware description languages, VHDL or Verilog. 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 G16V8. 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 16V8 is a 20-pin package with 16 inputs and 8 outputs (each output is also fed back and is available as an input). Pins 1 to 9 and 11 are inputs while pins 12 to 19 can be used as inputs or outputs. If an input is to be used to clock flip-flops 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 !, &, # and \$ correspond to NOT, AND, OR and XOR. Adding .D extension to the name of an output forces it to be a registered (flip-flop) output. The extension must be added to the output variable name so that the expression applies to the D input of the output flip-flop.

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 test them in the circuit. The input to the simulator is a description of the generated PAL logic equations (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 in the test vectors 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), and C for a low-high-low clock pulse. The values of the outputs are given as the logic levels H or L.

CUPL and CSIM Examples

The listings show the CUPL and CSIM input and output files for a simple sequential logic circuit.

```
Name Lab4 ;
PartNo 0 ;
Date 19/3/98 ;
Rev 1 ;
Designer Ed Casas ;
Company UBC ECE ;
Assembly 0 ;
Location 0 ;
Device g16v8 ;

Pin 1 = clk ; /* clock */
Pin 2 = show ; /* display enable */
Pin 19 = q2 ; /* state FFs */
Pin 18 = q1 ;
Pin 17 = q0 ;
Pin 16 = out2 ; /* outputs */
Pin 15 = out1 ;
Pin 14 = out0 ;

/* Random Number Generator using shift register with reset */

q2.d = ( q2 $ q0 ) # ( ! ( q2 # q1 # q0 ) ) ;
q1.d = q2 ;
q0.d = q1 ;

out2 = show & q2 ; /* gated display */
out1 = show & q1 ;
out0 = show & q0 ;

Name Lab4 ;
PartNo 0 ;
Date 19/3/98 ;
Rev 1 ;
Designer Ed Casas ;
Company UBC ECE ;
Assembly 0 ;
Location 0 ;
Device g16v8 ;

/* (assumes the registers are initialized to 1's) */

Order: clk, show, %4, out2, out1, out0 ;

Vectors:
C 1 L H H
C 1 H L H
C 1 L H L
C 1 L L H
```

C 1 H L L
C 0 L L L