Using Programmable Logic and the PALCE22V10

Programmable logic chips (like the PALCE22V10) provide a convenient solution for glue logic and state machine control required by your design. A single PAL chip can replace several MSI and SSI parts. It is strongly recommended (and even required) that you use programmable logic devices wherever there is not an efficient use of available MSI or SSI parts.

Programmable logic devices such as the PALCE22V10 allow the designer to enter a description of the desired hardware (either as a schematic or using some form of hardware description language) and the implement that logic using the hardware resources built onto the chip. Chips like the PALCE22V10 have a dedicated number of input pins with the remaining pins configurable as either outputs or more inputs. Details on the PALCE22V10 structure, pin out, and other details may be found on the EE 459 website.

The PALCE22V10 chip structure provides one D-FF per output pin. The FF output can connect to the output pin of the chip. In front of each FF is a set of AND-OR (sum of products) logic gates. Different outputs can support different numbers of product terms ranging from 8 to 16. (Remember in an equation like Y = AB + B'C, AB and B'C are called product terms and they are OR'ed together to form a sum of products expression.) This AND-OR structure can serve as the input to a D-FF or connect directly to an output pin.

When using the PALCE22V10 you must describe the desired hardware configuration using the hardware description language, CUPL. Using the CUPL language, combinational logic can be described by simply writing a logic equation, describing it using a function table, or even using conditional 'if' statements. State machines and sequential logic can be described using equations or simple state transition descriptions. A more complete discussion of CUPL can also be found on the EE 459 website. However, we will try to highlight the structure that you will use the most in your design.

Much more information can be found on the EE 459 website and at the following website: http://www.atmel.com/dyn/resources/prod_documents/DOC0737.PDF

Pin Declarations

The first section of your file should be the input and output pin declarations, though you do not need to explicitly specify which pins are inputs or outputs. Remember that on the PAL22CEV10 the clock is pin 1 and the rest of the pins 2-11 and 13 are dedicated inputs. Pin 14 through 23 can be either inputs or outputs.

When declaring a pin the format should be:

```
Pin # = name;
```

Similarly, buses can be declared as shown in the following example:

```
Pin [2..5] = [cnt3..0];
```

Combinational Logic

After your pin declarations you may start describing your logic. CUPL provides several different options for specifying combinational logic. To illustrate the different methods of specifying combinational logic we will use a 2-to-1 mux as a test case.

Equations

Using equations forms you must find the logical relationship between inputs and output on your own and then simply enter the equation into the CUPL file. For the 2-to-1 mux, the output Y would have the following equation:

```
Y = (I0 & !S) # (I1 & S)
```
(where & = AND, # = OR, ! = NOT)

Truth Table (Function table)

The truth table form of entry allows you to specify the output for specific input combinations or ranges of input combinations. When specifying a 2-to-1 mux with a truth table, realize that the output will only be true when I0 = 1 and S = 0 or I1 = 1 and S = 1. (If you concatenate I0,I1,S to form a 3-bit number, the output should only be true for combinations 100, 110, and 111 (4, 6, and 7).

```
FIELD muxin = [I0, I1, S]
TABLE muxin => Y {
'b'000 => 0;
  1..3 => 0;
     4 => 1;
     5 => 0;
  6..7 => 1;
}
```

CONDITION statements

CONDITION statements use and "IF THEN ELSE" structure similar to software programming languages to indicate when an output should be true. Again you need not cover cases when an output is deasserted but only when an output should be true.

```
CONDITION {
IF I0 & !S out Y
IF I1 & S out Y
}
```

Checking Equality or Range

Often times you may want to check whether a set of bits is equal to a certain value or within a certain range. For example if you have an 8-bit value CNT, you may want to have logic to check if it is equal to a particular end value or within a certain range. This can easily be accomplished in CUPL by use the ':' operator.

```
end = [cnt7..0]:E0;   /* checks if cnt = 1110 0000 (default
                         base is hex) */

enable = [cnt7..0]:[1A..E0];  /* enable = 1 if cnt is in
                                 range 1A to E0  */
```

These styles of specifying combinational logic provide flexibility for different designs. The equations form is probably best used for single-output functions with a small number of terms. The truth table method can be used for multiple-bit inputs or outputs allowing you to specify large ranges of values in a single case. Finally, condition statements can be used well when there are multiple outputs that are asserted for only a few combinations of the inputs.

Sequential Logic and State Machines

State machines can be specified using a SEQUENCE statement. A SEQUENCE statement requires you to list the state variables and then simply specify transitions using a PRESENT state to NEXT state structure. Mealy (asynchronous) or Moore (synchronous) style outputs can also be specified in the statements.

Consider a simple "101" sequence detector of a serial input stream, X, and producing a single Moore-style (synchronous) output, Z, when the sequence is found. A state machine can be designed using the state diagram represented in the following state transition table

Current State		Next State				Ouput
		X = 0		X = 1		
St.	Q1 Q0	St.*	Q1* Q0*	St.*	Q1* Q0*	Z
Sinit	0 0	Sinit	0 0	S1	0 1	0
S1	0 1	S10	1 0	S1	0 1	0
S10	1 0	Sinit	0 0	S101	1 1	0
S101	1 1	S10	1 0	S1	0 1	1

This state machine can be described with the following SEQUENCE statement:

```
$DEFINE   Sinit     'b'00
$DEFINE   S1        'b'01
$DEFINE   S10       'b'10
$DEFINE   S101      'b'11

SEQUENCE Q1,Q0 {
  present Sinit
    if X next S1;
    default next Sinit;
  present S1
    if !X next S10;
    default next S1;
  present S10
    if X next S101;
    default next Sinit;
  present S101
    out Z;
    if X next S1;
    default next S10;
}
```

A Mealy style implementation can be achieved using the following state machine.

Current State		Next State					
		X = 0			X = 1		
St.	Q1 Q0	St.*	Q1* Q0*	Z	St.*	Q1* Q0*	Z
Sinit	0 0	Sinit	0 0	0	S1	0 1	0
S1	0 1	S10	1 0	0	S1	0 1	0
S10	1 0	S1	0 1	0	Sinit	0 0	1

```
$DEFINE   Sinit     'b'00
$DEFINE   S1        'b'01
$DEFINE   S10       'b'10

SEQUENCE Q1,Q0 {
  present Sinit
    if X next S1;
```

```
    default next Sinit;
  present S1
    if !X next S10;
    default next S1;
  present S10
    if X next S1;
    if X out Z;
    default next Sinit;
}
```

These SEQUENCE descriptions will be converted to the necessary logic equations for the flip-flop inputs and the outputs. One thing to note is that in the PALCE22V10 flip-flops are associated with output pins and thus any state variable will use an output pin.

You can also specify the inputs of a D-FF directly and use the outputs as inputs to other logic. We could have implemented the "101" sequence checker by simply using a shift register as shown below:

```
Q0.D = X;  /* Specifies the input of the Q0 D-FF to be X */
Q1.D = Q0;
Q2.D = Q1;
Z = Q2 & !Q1 & Q0;
```

Reset

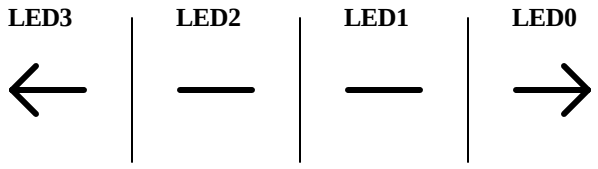
Remember that whatever method you use to specify your sequential logic and state machines that you should implement a reset so that you start in a known state. The PALCE22V10 provides a common synchronous preset (SP) and asynchronous reset (AR).

In the examples using the SEQUENCE statements above an initial state can be implemented using the following lines of CUPL code:

```
Q0.AR = RESET;
Q0.SP = 'b'0;
Q1.AR = RESET
Q1.SP = 'b'0;
```

CUPL Example of State Machine and Combinational Logic

The following file (cupl_ex.pld) implements a state machine of left and right arrow detour lights with 4 segments



It also implements combinational logic to check an 8-bit number if it is equal to 20 decimal.

Below is the contents of the CUPL.pld file:

```
Name     cupl_ex ;
PartNo   01 ;
Date     1/24/2005 ;
Revision 01 ;
Designer Mark Redekopp ;
Company  USC ;
Assembly None ;
Location None ;
Device   g22v10 ;


/* inputs */
Pin 1 = CLK;
Pin 2 = reset;
Pin 3 = left;

Pin 4 = CNT0;
Pin 5 = CNT1;
Pin 6 = CNT2;
Pin 7 = CNT3;
Pin 8 = CNT4;
Pin 9 = CNT5;
Pin 10 = CNT6;
Pin 11 = CNT7;

/* outputs - notice we define these as active-low */
Pin 14 = !LED3;  /* Left-most light segment */
Pin 15 = !LED2;
Pin 16 = !LED1;
Pin 17 = !LED0;  /* Right-most light segment */

Pin 18 = start;

Field mystate = [LED3,LED2,LED1,LED0];

mystate.ar = reset;
mystate.sp = 'b'0;

/* Logic for the left/right arrows */
SEQUENCE mystate {
/* a 0 in the constants below mean "deasserted".  Thus, since the LED signals are
declared active-low, the output will actually be a logic '1' */
  present 'b'0000
    if left next 'b'0010;
    default next 'b'0100;
  present 'b'0010
    default next 'b'0110;
  present 'b'0110
    if left next 'b'1110;
    default next 'b'0111;
  present 'b'1110
    default next 'b'0000;
  present 'b'0100
    default next 'b'0110;
  present 'b'0111
    default next 'b'0000;
}
```

```
field count = [CNT7,CNT6,CNT5,CNT4,CNT3,CNT2,CNT1,CNT0];

/* the : operator is an equality comparison test; start will be 1 when count = 20
decimal */
start = count:'d'20;
```

Test vector file, “cupl_ex.si”

```
Name     cupl_ex;
PartNo   01;
Date     1/24/2005;
Revision 01;
Designer Mark Redekopp;
Company  USC;
Assembly None;
Location None;
Device   g22v10;


ORDER: CLK, CNT0, CNT1, CNT2, CNT3, CNT4, CNT5, CNT6, CNT7, !LED0, !LED1, !LED2,
!LED3, left, reset, start;


VECTORS:
C00101000****11*
C00101000****11*
C00100000****10*
C00111000****10*
C00001000****10*
C01101000****00*
C10101000****00*
C00101000****00*
C00101000****00*
C00101000****00*
C00101000****00*
```