SOURCE FILE / PAL EQUATIONS

```
#TITLE     'LPNCPAL0.SRC';
#ENGINEER  'Bill Sisk';
#COMPANY   'Arecibo Observatory';
#REVISION  '0';
#PROJECT   'IP Interface, Long (40bits) PN Code Generator';
#COMMENT   '9/17/98';

"11/5/99 Added 3 bits to the shift enable length from original 12-bit version.
"A shift enable counter with 16 bits will not fit.

"Note 1: The ip_clk can be 8MHz or 32MHz.  The multclk is usually 20MHz.
"Note 2: Multclk is global clock 3. An asynchronous macrocell can only use global
"clocks 1 and 2.  Thus an asynchronous macrocell using multclk will use a product
"term clock. The nodes are partitioned so that the all the critical macrocells are
"synchronous. Synchronous macrocells in a block use common set and/or reset signals.

"Reference for 40-bit PN Code :
"  Pseudo-Random Sequences and Arrays, F. Jessie MacWilliams and Neil Sloane
"  Proceedings of IEEE,Vol 64,No 12,Dec 1976, p. 1715

INPUT ip_clk;
LOW_TRUE INPUT iosel;
LOW_TRUE INPUT idsel;
INPUT ad[3..1];
INPUT rw;
LOW_TRUE INPUT ip_reset;
BIPUT ip_d[15..0]           ENABLED_BY read;
LOW_TRUE D_FLOP OUTPUT ack  PRESET_BY ip_reset CLOCKED_BY ip_clk;
D_FLOP NODE sbits                        CLOCKED_BY ip_clk;

D_FLOP NODE w_com_wd        RESET_BY ip_reset  CLOCKED_BY ip_clk;
D_FLOP NODE w_se_len        aESET_BY ip_reset  CLOCKED_BY ip_clk;
D_FLOP NODE w_config_wd     RESET_BY ip_reset  CLOCKED_BY ip_clk;
D_FLOP NODE w_st_wd0        RESET_BY ip_reset  CLOCKED_BY ip_clk;
D_FLOP NODE w_st_wd1        RESET_BY ip_reset  CLOCKED_BY ip_clk;
D_FLOP NODE w_st_wd2        RESET_BY ip_reset  CLOCKED_BY ip_clk;
D_FLOP NODE read            RESET_BY ip_reset  CLOCKED_BY ip_clk;
D_FLOP NODE code_wclk       RESET_BY ip_reset  CLOCKED_BY ip_clk;

INPUT fp_extclk;
INPUT rp_extclk;
PHYSICAL NODE extclk;
D_FLOP NODE sync_reg[1..0]  RESET_BY ip_reset  CLOCKED_BY (/extclk);
OUTPUT clk_out;
INPUT multclk;

D_FLOP NODE se_start_im     RESET_BY ip_reset  CLOCKED_BY w_com_wd;
D_FLOP NODE se_stop         PRESET_BY ip_reset CLOCKED_BY w_com_wd;
D_FlOP NODE se_sync_stop    RESET_BY ip_reset  CLOCKED_BY w_com_wd;
D_FLOP NODE en_extclk       RESET_BY ip_reset  CLOCKED_BY w_com_wd;
INPUT trig;
D_FLOP NODE se_reg[2..0]    RESET_BY stop      CLOCKED_BY multclk;
D_FLOP node en_se           RESET_BY stop      CLOCKED_BY multclk;
PHYSICAL NODE stop;

D_FLOP NODE ai[14..0]                          CLOCKED_BY w_se_len;
T_FLOP NODE a[14..0]        RESET_BY /en_se    CLOCKED_BY multclk;
D_FLOP NODE ldaa            PRESET_BY /en_se   CLOCKED_BY multclk;
D_FLOP NODE se              RESET_BY /en_se    CLOCKED_BY multclk;
D_FLOP NODE marker          RESET_BY /en_se    CLOCKED_BY multclk;
D_FLOP NODE code                               CLOCKED_BY multclk;
D_FLOP NODE c[39..0]                           CLOCKED_BY multclk;
D_FLOP OUTPUT code_out                         CLOCKED_BY multclk;
D_FLOP OUTPUT marker_out                       CLOCKED_BY multclk;
D_FLOP OUTPUT clk_sel       RESET_BY ip_reset  CLOCKED_BY w_config_wd;
D_FLOP OUTPUT code_sel      RESET_BY ip_reset  CLOCKED_BY w_config_wd;

se.d = ld_s;

ld_s.d = /ld_s* a[0]* /a[1]* /a[2]* /a[3]* /a[4]* /a[5]* /a[6]* /a[7]*
       /a[8]* /a[9]* /a[10]* /a[11]* /a[12]* /a[13]* /a[14]
       + /ai[0]* /ai[1]* /ai[2]* /ai[3]* /ai[4]* /ai[5]* /ai[6]* /ai[7]*
       /ai[8]* /ai[9]* /ai[10]* /ai[11]* /ai[12]* /ai[13]* /ai[14];

ai[14..0].D = ip_d[14..0];

a[0].T = /ld_s +
      ld_s* (a[0] (+) ai[0]);

a[1].T = /ld_s* /a[0] +
      ld_s* (a[1] (+) ai[1]);

a[2].T = /ld_s* /a[0]* /a[1] +
      ld_s* (a[2] (+) ai[2]);

a[3].T = /ld_s* /a[0]* /a[1]* /a[2] +
      ld_s* (a[3] (+) ai[3]);

a[4].T = /ld_s* /a[0]* /a[1]* /a[2]* /a[3] +
      ld_s* (a[4] (+) ai[4]);

a[5].T = /ld_s* /a[0]* /a[1]* /a[2]* /a[3]* /a[4] +
      ld_s* (a[5] (+) ai[5]);

a[6].T = /ld_s* /a[0]* /a[1]* /a[2]* /a[3]* /a[4]* /a[5] +
      ld_s* (a[6] (+) ai[6]);

a[7].T = /ld_s* /a[0]* /a[1]* /a[2]* /a[3]* /a[4]* /a[5]* /a[6] +
      ld_s* (a[7] (+) ai[7]);

a[8].T = /ld_s* /a[0]* /a[1]* /a[2]* /a[3]* /a[4]* /a[5]* /a[6]* /a[7] +
      ld_s* (a[8] (+) ai[8]);

a[9].T = /ld_s* /a[0]* /a[1]* /a[2]* /a[3]* /a[4]* /a[5]* /a[6]* /a[7]*
      /a[8] +
      ld_s* (a[9] (+) ai[9]);

a[10].T = /ld_s* /a[0]* /a[1]* /a[2]* /a[3]* /a[4]* /a[5]* /a[6]* /a[7]*
      /a[8]* /a[9] +
      ld_s* (a[10] (+) ai[10]);

a[11].T = /ld_s* /a[0]* /a[1]* /a[2]* /a[3]* /a[4]* /a[5]* /a[6]* /a[7]*
        /a[8]* /a[9]* /a[10] +
      ld_s* (a[11] (+) ai[11]);

a[12].T = /ld_s* /a[0]* /a[1]* /a[2]* /a[3]* /a[4]* /a[5]* /a[6]* /a[7]*
        /a[8]* /a[9]* /a[10]* /a[11] +
      ld_s* (a[12] (+) ai[12]);
```

```
a[13].T = /ld_s* /a[0]* /a[1]* /a[2]* /a[3]* /a[4]* /a[5]* /a[6]* /a[7]*
        /a[8]* /a[9]* /a[10]* /a[11]* /a[12] +
      ld_s* (a[13] (+) ai[13]);

a[14].T = /ld_s* /a[0]* /a[1]* /a[2]* /a[3]* /a[4]* /a[5]* /a[6]* /a[7]*
        /a[8]* /a[9]* /a[10]* /a[11]* /a[12]* /a[13] +
      ld_s* (a[14] (+) ai[14]);

clk_sel.d = ip_d[0];
code_sel.d = ip_d[1];

IF clk_sel THEN
      extclk = fp_extclk;
ELSE
      extclk = rp_extclk;
END IF;

sync_reg[0] = en_extclk;
sync_reg[1] = sync_reg[0];
clk_out = extclk * sync_reg[1] + code_wclk;


IF se THEN
      c[15..0].D = c[16..1];
ELSIF w_st_wd0 THEN
      c[15..0].D = ip_d[15..0] ;
ELSE
      c[15..0].D = c[15..0];
END IF;


IF se THEN
      c[31..16].D = c[32..17];
ELSIF w_st_wd1 THEN
      c[31..16].D = ip_d[15..0];
ELSE
      c[31..16].D = c[31..16];
END IF;


IF se THEN
      c[38..32].D = c[39..33];
ELSIF w_st_wd2 THEN
      c[38..32].D = ip_d[6..0];
ELSE
      c[38..32].D = c[38..32];
END IF;


IF se THEN
      c[39].D = c[21] (+) c[19] (+) c[2] (+) c[0];
ELSIF w_st_wd2 THEN
      c[39].D = ip_d[7];
ELSE
      c[39].D = c[39];
END IF;

STATE_MACHINE acknowledge
      CLOCKED_BY ip_clk
      STATE_BITS  sbits
      RESET_BY ip_reset;
      STATE zero :
            IF (iosel + idsel) THEN
                  GOTO one;
            ELSE
                  GOTO zero;
            END IF;
            ack.d = 0;
      STATE one :
            IF (iosel + idsel) THEN
                  GOTO one;
            ELSE
                  GOTO zero;
            END IF;
            ack.d = 1;
END acknowledge;

CASE [ip_d[1..0]]
        WHEN 0 =>
                se_start_im.d = 0;
                se_stop.d = 1;
            se_sync_stop.d = 0;
            en_extclk.d = 0;
        WHEN 1 =>
                se_start_im.d = 0;
                se_stop.d = 0;
            se_sync_stop.d = 1;
            en_extclk.d = 0;
        WHEN 2 =>
                se_start_im.d = 0;
                se_stop.d = 0;
            se_sync_stop.d = 0;
            en_extclk.d= 1;
        WHEN 3 =>
                se_start_im.d = 1;
                se_stop.d = 0;
            se_sync_stop.d = 0;
            en_extclk.d = 1;
END CASE;

stop = se_stop + se_sync_stop * /sync_reg[1];

se_reg[0].d = se_start_im + trig + se_reg[0];
en_se.d = se_reg[0];
se_reg[1].d = en_se;
se_reg[2].d = se_reg[1];

IF se THEN
      code.d = c[0];
      marker.d = /se_reg[2];
ELSE
      code.d = code;
      marker.d = marker;
END IF;

marker_out.d = marker;
code_out.d = code;

w_st_wd0.d    = iosel * /rw * /ad[3] * /ad[2] * /ad[1] + w_st_wd0 * /ack;
w_st_wd1.d    = iosel * /rw * /ad[3] * /ad[2] *  ad[1] + w_st_wd1 * /ack;
w_st_wd2.d    = iosel * /rw * /ad[3] *  ad[2] * /ad[1] + w_st_wd2 * /ack;
code_wclk.d   = w_st_wd0 + w_st_wd1 + w_st_wd2;
```

```
w_se_len.d    = iosel * /rw * /ad[3] *  ad[2] *  ad[1];
w_com_wd.d    = iosel * /rw *  ad[3] * /ad[2] * /ad[1];
w_config_wd.d = iosel * /rw *  ad[3] * /ad[2] *  ad[1];
read.d        = iosel *  rw + read * /ack;

CASE [ad[3..1]]
        WHEN 0 =>
                ip_d[15..0] = c[15..0];
        WHEN 1 =>
                ip_d[15..0] = c[31..16];
        WHEN 2 =>
                ip_d[7..0] = c[39..32];
                ip_d[15..8] = 0;
        WHEN 3 =>
                ip_d[14..0] = ai[14..0];
                ip_d[15] = 0;
        WHEN 4 =>
                ip_d[0] = en_se;
            ip_d[1] = clk_sel;
            ip_d[2] = code_sel;
            ip_d[15..3] = 0;
        ELSE
                ip_d[15..0] = 0ffffh;
END CASE;
```

PONOUT

Pin	Type	Signal	Pin	Type	Signal
1	GND		43	GND	
2	Vcc		44	Vcc	
3	Biput		45	Biput	
4	Biput		46	Biput	
5	Biput		47	Biput	ip_d[15]
6	Biput		48	Biput	ip_d[14]
7	Biput		49	Biput	clk_out
8	Biput		50	Biput	ip_d[13]
9	Biput		51	Biput	ack
10	Biput		52	Biput	ip_d[12]
11	GND		53	GND	
12	Biput		54	Biput	
13	Biput		55	Biput	ip_d[11]
14	Biput		56	Biput	
15	Biput		57	Biput	ip_d[10]
16	Biput		58	Biput	
17	Biput		59	Biput	ip_d[9]
18	Biput		60	Biput	ad[3]
19	Biput		61	Biput	ip_d[8]
20	In/CLK	ip_clk	62	In/CLK	multclk
21	Vcc		63	Vcc	
22	GND		64	GND	
23	In/CLK	rp_extclk	65	In/CLK	
24	Biput		66	Biput	ad[2]
25	Biput		67	Biput	ad[1]
26	Biput		68	Biput	ip_d[7]
27	Biput		69	Biput	iosel
28	Biput	code_out	70	Biput	ip_d[6]
29	Biput	marker_out	71	Biput	ip_d[5]
30	Biput	clk_sel	72	Biput	ip_d[4]
31	Biput	code_sel	73	Biput	idsel
32	GND		74	GND	
33	Biput		75	Biput	ip_d[3]
34	Biput		76	Biput	rw
35	Biput		77	Biput	ip_d[2]
36	Biput		78	Biput	ip_reset
37	Biput		79	Biput	ip_d[1]
38	Biput		80	Biput	ip_d[0]
39	Biput		81	Biput	
40	Biput		82	Biput	trig
41	Input		83	Input	fp_extclk
42	Vcc		84	Vcc	

WIRELIST

Signal	Device	Pin
ip_clk	MACH435_1	20
iosel	MACH435_1	69
idsel	MACH435_1	73
ad[3]	MACH435_1	60
ad[2]	MACH435_1	66
ad[1]	MACH435_1	67
rw	MACH435_1	76
ip_reset	MACH435_1	78
ip_d[15]	MACH435_1	47
ip_d[14]	MACH435_1	48
ip_d[13]	MACH435_1	50
ip_d[12]	MACH435_1	52
ip_d[11]	MACH435_1	55
ip_d[10]	MACH435_1	57
ip_d[9]	MACH435_1	59
ip_d[8]	MACH435_1	61
ip_d[7]	MACH435_1	68
ip_d[6]	MACH435_1	70
ip_d[5]	MACH435_1	71
ip_d[4]	MACH435_1	72
ip_d[3]	MACH435_1	75
ip_d[2]	MACH435_1	77
ip_d[1]	MACH435_1	79
ip_d[0]	MACH435_1	80
ack	MACH435_1	51
fp_extclk	MACH435_1	83
rp_extclk	MACH435_1	23
clk_out	MACH435_1	49
multclk	MACH435_1	62
trig	MACH435_1	82
code_out	MACH435_1	28
marker_out	MACH435_1	29
clk_sel	MACH435_1	30
code_sel	MACH435_1	31