intel®

APPLICATION NOTE

AP-279

May 1986

Implementing an EPLD Design Using Intel's Programmable Logic Development System

LAKSHMI JAYANTHI
DSO APPLICATIONS

Order Number 280310-001

OVERVIEW

Welcome to the fascinating world of ERASABLE PROGRAMMABLE LOGIC DEVICES (EPLDs) and Intel's Programmable Logic Development System (iPLDS). This application note has been written for the newcomer to Intel's devices and design tools. It has been designed as a step-by-step guide through the tools but should also prove useful as a reference document for the experienced logic designer.

By the end of this application note you will have designed/solved multiple logic problems and be in a position to implement solutions to many of the digital design challenges you face today. It is anticipated that this application note will be used in conjunction with Intel's iPLS software. To increase the usefulness of this application note, Intel will supply a PCB card for you to experiment on and a sample diskette (see Appendix E for details).

This application note is divided into the following three sections:

1. An introduction to Erasable Programmable Logic Devices (EPLD)
2. An introduction to Intel's Programmable Logic Development System (iPLDS)
3. Implementation of EPLD and iPLDS using detailed examples to implement a logic design.

INTRODUCTION

Programmable logic in the form of PALs have been available for some time. They have become more complex as Large Scale Integration (LSI) techniques have been applied to this technology.

The benefits of Large Scale Integration circuits are many fold. These circuits offer lower manufacturing costs, since the use of customized LSI circuits reduces required printed circuit board space, thereby significantly reducing board costs. These circuits also consume lower power so less expensive power supplies are required and cooling fans are also eliminated. LSI circuits also have higher reliability than equivalent systems comprised of many low density standard components.

As end users of semiconductors moved into higher and higher levels of integration, chip designers found it more and more difficult to define larger and larger blocks of logic. These difficulties led to the emergence of the user-defined Application Specific Integrated Circuit (ASIC).

The options available for application specific logic are explained below and shown in Figure 1.

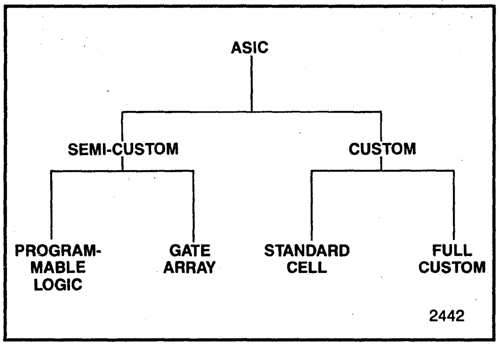


Figure 1. Logic Options

Full Custom: These circuits can be tailored to give the best functional performance with the highest level of integration, the smallest silicon area, the lowest power use, and be produced for the least cost at high production volumes.

Standard Cell Library: This approach represents an integrated circuit which is composed of predesigned and precharacterized cells chosen from a computer data base library of cells.

Gate Arrays: These are integrated circuits that contain a regular, usually square, matrix of predefined logic gates.

User Programmable Logic: The concept of user programmable logic is to provide the designer with the benefits of custom LSI chips from standard products.

A recent innovation in the programmable logic field has been Intel's introduction of an ERASABLE Programmable Logic Device. Using the same technology used in the manufacture of EPROMs, Intel now offers increased flexibility to the logic designer.

Intel has addressed the limitations of gate arrays and fuse programming logic with its EPLD products and development system support tools. The benefits to the system designer are:

- Greatly reduced lead times
- Low design costs
- Ease of design changes
- Low power dissipation from CHMOS technology
- Multiple programming facility
- Maximum flexibility in each chip and the ability to erase and reprogram
- High density products that maximize function, integration, and quality
- A self-contained, low-cost sophisticated development system based upon the industry standard IBM PC XT or AT.

Table 1. Intels EPLDs

EPLD	Gates	Pins	Dedicated Inputs	I/O
5C031	300	20	10	8
5C060	600	24	4	16
5C090	900	40	12	24
5C121	1200	40	13	24
5C180	1800	68	12	48

EPLDs are now a cost-effective solution to the problem of large scale logic integration. EPLDs are the simplest form of high density application-specific logic to implement. At present, the following logic devices are available from Intel as shown in Table 1.

Intel's EPLDs use the "Sum Of Products" architecture with programmable AND and fixed OR gates to drive a combinatorial or registered output. Each of the devices listed in Table 1 has different attributes and resources targeted at specific applications.

In general each device contains multiple sets of programmable MACROCELLS as shown in Figure 2.

Everything is programmable (and erasable if you need to make modifications). Product terms may be generated from any combination of input terms—any terms not used are considered a "don't-care" in the array. The output register is also programmable—you can choose D-type, Toggle, SR, or even JK FLIP-FLOPs; you can even choose no output register if you only require combinatorial outputs. The clock and output enables are also programmable.

Intel EPLD devices are available in many configurations to fit most applications. A complete listing of data sheet availability is covered in Appendix E.

DESIGN TECHNIQUES USING INTEL'S EPLDS

Designing with EPLDs is similar to designing with standard TTL logic circuits. The focus moves from "how can I configure this design with standard parts" to "what else could I replace using this EPLD". Remember, if you ever use all of an EPLDs resources you just move up the device chain to the next bigger component—all of the work you did is DIRECTLY PORTABLE to a larger device.

Any network, either combinatorial or registered, has an equivalent two level form. Any logic circuit consisting of AND, OR, NOR, NAND, XOR Logic can easily be converted into the corresponding truth table. Any Boolean expression, no matter how complex, may be written in Sum-Of-Products form. This Sum-Of-Products expression that has been derived from the truth table can be reduced until it has as few product terms as possible. This procedure can be repeated for any complex network.

Let us consider a very simple network as shown in Figure 3. This logic circuit consists of an AND gate, an OR gate and a NOT gate. The inputs are A, B, C, and the output is Y.

For this simple network, the truth table is shown in Table 2:

A Boolean expression can easily be written from the truth table in a Sum-Of-Products form. This expression contains the relationship between the inputs and the output.

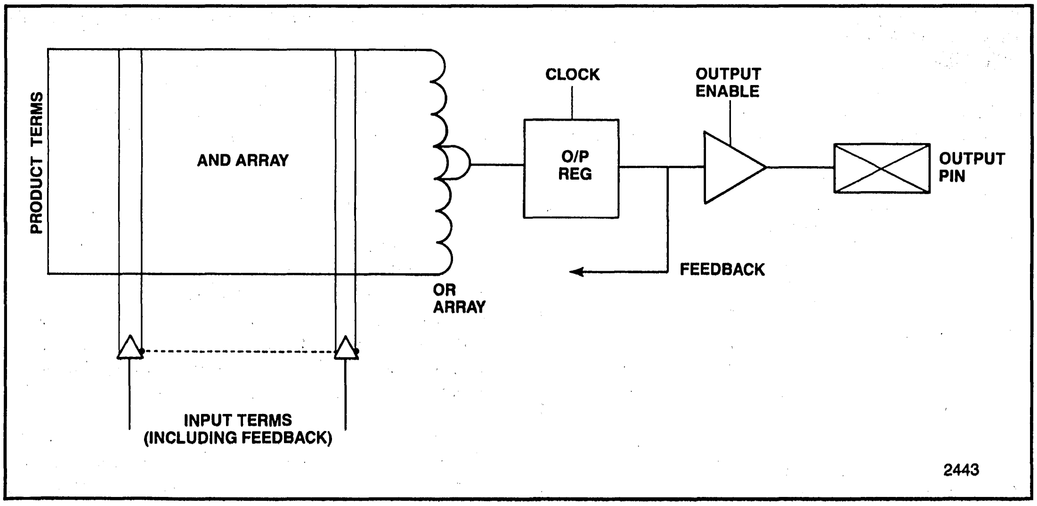


Figure 2. Macrocell Arch

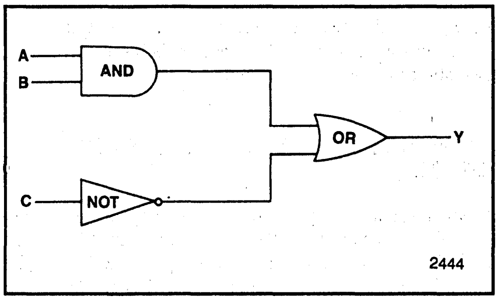


Figure 3. Simple Network

Note that the output Y is true in five of these eight states (0,2,4,6, and 7) so expressing Y in the form "Sum-Of-Products" by writing the ones in terms of A, B, and C yields:

Y = /A*/B*/C + /A*B*/C + A*/B*/C
+ A*B*/C + A*B*C

Hence, given any network, that network can be converted into its truth table. Next, a Sum-Of-Products expression that has the same truth table can be derived. If so desired, this Sum-Of-Products expression can be reduced using DeMorgan's theorem to simplify the circuit (you will see later that this will not be required).

DEVELOPMENT SUPPORT

Development tools are critical to the use of new technologies because tools allow you to control and use a new technology. Good tools help you, the designer, to work in familiar methods, then translate the design to the device.

Good tools broaden the applications by making it easy to use new technology in designs. They are not a barrier to using the technology, but encourage its use and applications.

Advanced and innovative technologies need similar advancements and innovations in the corresponding tools.

Table 2.

STATE	INPUT			OUTPUT
	A	B	C	Y
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

iPLDS, Intel's Programmable Logic Development System, provides a full spectrum of ways to design and use a variety of design tools with fast, easy-to-use entry software.

The iPLDS contains all the software, hardware, documentation and devices needed to program EPLDs. iPLDS are the most advanced PLD design tools available. It provides better utilization of device resources (more gates per chip) than any other development software. These versatile tools are for users with different skill levels and applications. iPLDS tools handle the details of converting your design to working silicon on the personal computer.

The iPLDS contains the three fundamental modules

- Logic Builder (LB)
- Logic Optimizimg Compiler (LOC)
- Logic Programmer Software (LPS)

To implement the logic design we will use the iPLDS modules in the order listed above.

The modules are essentially independant modules that use special data files to pass information as shown in Figure 4. These data files are the ADF, RPT, LEF, and JED files.

The Advanced Design File (*.ADF) is generated from the Logic Builder and contains the Inputs/outputs and all the primitive equations.

The Logic Equation File (*.LEF) contains the primitive equations that have been minimized by the Logic Optimizing Compiler.

The Utilization Report File (*.RPT) contains information on the macrocell and pin assignments.

The JEDEC File (*.JED) is the file generated by the Logic Optimizing Compiler used to program the device using the Logic Programmer.

Before implementing the logic design using the iPLDS, let us briefly discuss the iPLDS family of parts to be familiar with the iPLDS modules.

Logic Builder (LB)

The Logic Builder module guides you through the entire process of design entry by prompting for necessary information and showing a screen display (one primitive at a time) with input signals on the left side and output signals on the right side. The Logic Builder is used to generate an Advanced Design File (or ADF) by inputting the data in netlists or Boolean equations.

After all required data are entered, the Logic Builder module indicates whether the circuit is complete and properly connected. If any changes need to be made, the module enables you to edit the circuit design either by

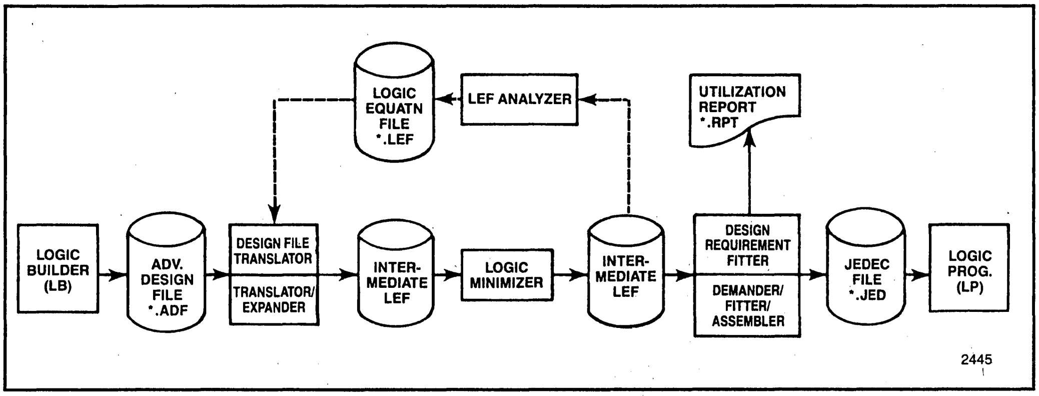


Figure 4. Block Diagram of iPLDS Modules

systematically scanning through the primitives in the Advanced Design File (ADF) or by directly finding a primitive by the name of a node connected to it.

Any circuit may be edited. The Logic Builder reads in the ADF and prompts you for changes. The Logic Builder also allows two or more partially complete ADF files to be MELDED together to form a more complex function. This concept is not discussed in this application note but will be a topic of a future application note.

Logic Optimizing Compiler (LOC)

The Logic Optimizing Compiler provides an easy-to-use interface to the Logic User System software. Regardless of the type of design entry method used, the LOC first translates an Advanced Design File (ADF) into internal logic equations; then it performs a Boolean reduction on the translated design, and finally produces a JEDEC Standard File, which is then used to program an Intel EPLD. In addition, you have the option of requesting an analysis of the Logic Equation File (LEF) as output by the Minimizer module.

The LOC performs the following functions:

- The TRANSLATOR translates the ADF into an intermediate Logic Equation File (LEF). (Most errors are detected and corrected).
- The EXPANDER expands the Boolean equations into Sum-Of-Products form, removes redundant factors from product terms, and produces another LEF.
- The MINIMIZER performs a sophisticated Boolean reduction on the translated design to maximize utilization of the EPLD.
- The LEF Analyzer converts the LEF output by the MINIMIZER into a human readable file to allow you to see your design. (*.LEF)
- The DEMANDER organizes the file output by the MINIMIZER.
- The FITTER matches your design requirements with the known resources of the Intel device.
- The ASSEMBLER converts the fitted requests into JEDEC file.

Logic Programmer Software (LPS)

The Logic Programmer Software provides a user interface to the JEDEC Standard File output of the Logic Optimizing Compiler and to the Logic Programmer Interface. You can use the Logic Programmer Software to view JEDEC files and to program your designs into EPLDs.

The Logic Programmer Software is used

- to program your designs into EPLDs
- to verify the validity of data in the device
- to read data from the device
- to display JEDEC data graphically
- to edit JEDEC data

HARDWARE REQUIREMENTS

The iPLDS requires an IBM PC XT, PC AT, or other compatible computer. A color monitor is preferred. The computer must have at least one 360K double-sided double-density disk drive, a second 360K floppy disk or hard disk, and at least 512K bytes of RAM memory.

The iPLDS consists of the Logic Programmer Interface card, and the programming unit needed to program and verify EPLDs. The Intel iUP 201 with a GUPI adapter may be used as an alternate system to program the EPLD devices.

SOFTWARE REQUIREMENTS

The personal computer should be capable of running DOS V3.0 or a higher version. The Intel Programmable Logic

Software (iPLS) that contains the software controlling the logic programmer interface and assisting in the design of Intel applications is shipped on floppy diskettes.

PROBLEM DEFINITION

We are going to use iPLDS to implement a medium complexity logic function. As a vehicle to show the usage of the tools and design techniques we will design a circuit that will roll and spin a pair of dice. The design has been split into multiple stages for illustration purposes.

This example has been chosen since it incorporates many of a typical logic design tradeoffs and also solves many of the typical problems a hardware logic designer will encounter.

Appendix A contains some basic definitions that may be useful when reading through the design and its implementation.

DESIGN SAMPLE

Problem Set-up

The circuit is designed to set both of the dice spinning when you push a switch and display a random set of numbers when you release the switch. The dice will spin at a rate that is visually pleasing and roll at the highest possible rate to ensure randomness.

You will implement the design in the following steps:

A. One dice that will roll out a number.

B. Add a switch that will control the roll/not roll action.

C. Add a second dice to roll a number.

D. Add a spinning option to both dice.

E. Retro-fit a power save feature to extend battery life.

Hence, at the end of the five design steps you will have a pair of dice spinning and showing a pair of numbers between 1 and 6 in a very random manner. At the end of the five design steps, you will have added a very realistic and practical feature to your design and that is extending the battery life by a power saving option. It is important to note that the five steps mentioned above are sequential steps in that step C can be achieved only after steps A and B etc. Let us describe the sample circuit for the dice rolling example. It is a very simple circuit allowing you to concentrate upon the design process. It illustrates the possible design stages and considerations in detail.

PART A

Four Outputs—1A, 1B, 1C, 1D are required to drive the LEDs arranged in a DICE pattern as shown in Figure 5.

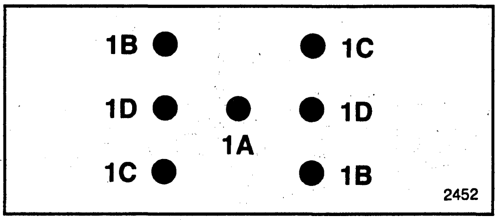


Figure 5. Dice Configuration

Operating sequence—Rolling dice from 1 to 6 and the block diagram of the circuit, both shown in Figure 6.

The total number of states that are possible is 16 since the four LED pairs generate a permutation of (2**4) = 16. The LEDs should be lit up such that any number between 1 and 6 inclusive is shown. Hence, out of the 16 possible states, only six states are valid. This leaves ten invalid states.

If the LEDs come up in a valid state upon power up, then a number between 1 and 6 will be displayed.

However, if the LEDs come up in an invalid state upon power up, then you have to design the circuit such that any one of the ten invalid states will fall into a valid state.

If the LEDs fall into any one of the ten invalid states, then you have designed the circuit to move into a state where 1A, 1B, 1C, 1D have zero logic values respectively on the next clock edge. Every time a zero logic value appears in the invalid states, then at the next clock edge, LED 1A gets lit up generating a valid state. Since 1 is a valid state, the numbers between 1 and 6 inclusive will be displayed at all subsequent clock edges.

Listed below are the steps involved in designing the logic circuit.

STEP 1. Generate the state diagram to clearly show the operating sequence including the status of the outputs for each state and the influence of the inputs on the next state transitions as shown in Figure 7. We have arbitrarily chosen that the states should count 1,2,3,4,5,6, and repeat. You could have implemented the design using any sequence but we chose the most obvious. Note how most of the invalid states move you to state 0 which then puts us into a valid state which then repeats forever.

STEP 2. Generate a truth table with entries for all available states and combinations of inputs, and use the next states resulting from these as shown in Table 3. The bracketed numbers, (3) etc., show the number being

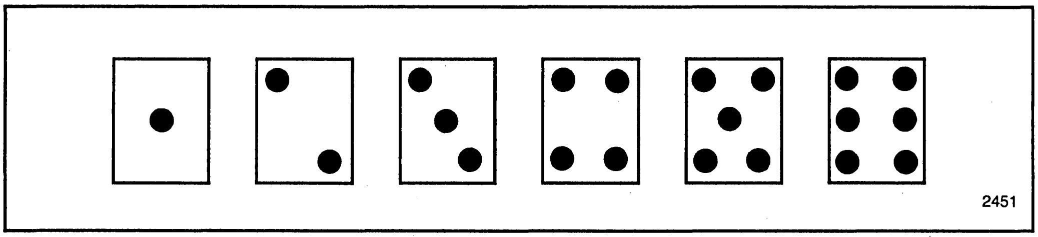


Figure 6. Rolling Sequence

displayed on the dice and the 0, 1 values of 1D, 1C, 1B, and 1A indicate which LEDs should be OFF/ON to display the required dice pattern.

STEP 3. Convert the truth table directly into Sum-Of-Products equations as shown below:

DICE1A has four entries; 3 from the valid states and one to control the invalid states

```
DICE1A = (/1A*1B*/1C*/1D + /1A*1B*/1C*/1D
         + /1A*1B*1C*1D + /1A*/1B*/1C*/1D)
```

DICE1B has five entries from valid states

```
DICE1B = (1A*/1B*/1C*/1D + /1A*1B*/1C*/1D
         + 1A*1B*/1C*/1D + /1A*1B*1C*/1D
         + 1A*1B*1C*/1D)
```

DICE1C has three entries from valid states

```
DICE1C = (1A*1B*/1C*/1D + /1A*1B*1C*/1D
         + 1A*1B*1C*/1D)
```

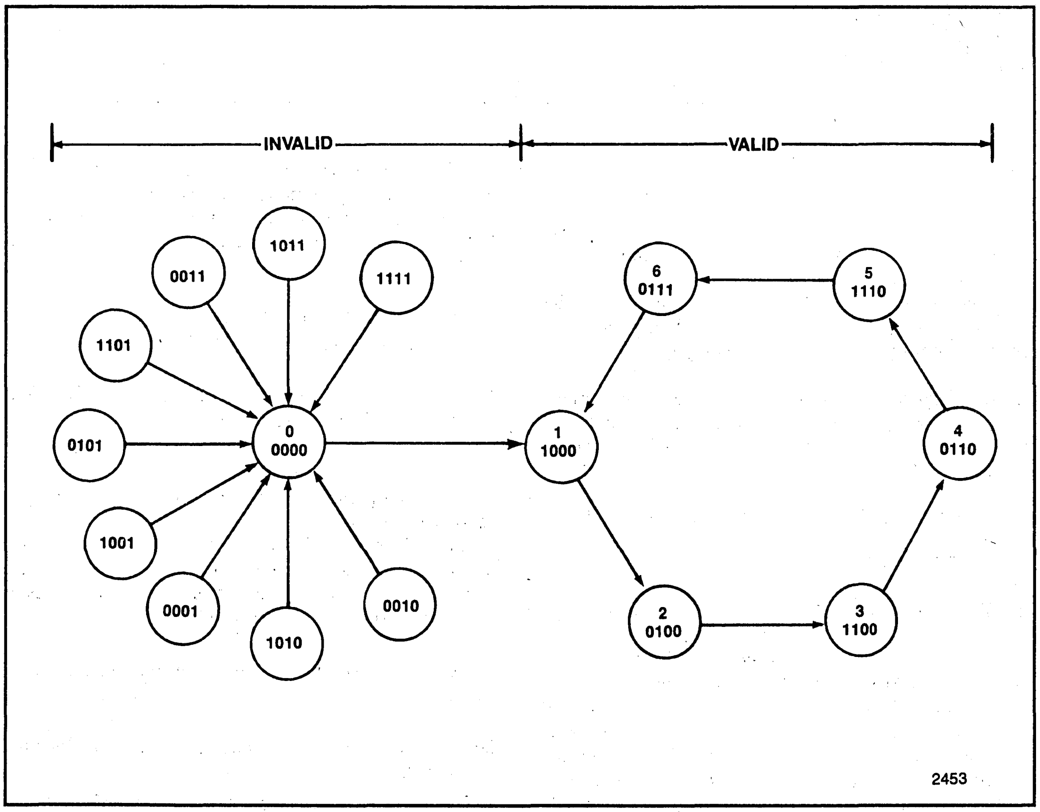


Figure 7.

Table 3. Truth Table for DICE1

Input State				Output State							
1A	1B	1C	1D	1A	1B	1C	1D	1A	1B	1C	1D
				Valid state				Invalid state			
				CHANGE TO THE NEXT VALID STATE							
1	0	0	0(1)	0	1	0	0(2)				
0	1	0	0(2)	1	1	0	0(3)				
1	1	0	0(3)	0	1	1	0(4)				
0	1	1	0(4)	1	1	1	0(5)				
1	1	1	0(5)	0	1	1	1(6)				
0	1	1	1(6)	1	0	0	0(1)				
				CONTROL THE INVALID STATES							
0	0	1	0					0	0	0	0
1	0	1	0					0	0	0	0
0	0	0	1					0	0	0	0
1	0	0	1					0	0	0	0
0	1	0	1					0	0	0	0
1	1	0	1					0	0	0	0
0	0	1	1					0	0	0	0
1	0	1	1					0	0	0	0
1	1	1	1					0	0	0	0
0	0	0	0					1	0	0	0(1)

DICE1D has one valid entry

DICE1D = (/1A*1B*1C*1D)

Note that no attempt has been made to minimize these equations - the iPLS software that you will use later contains reducing algorithms and other techniques to optimize the design. This allows you to focus upon the problem and not on tasks such as Karnaugh map reduction which a computer can often do better anyway.

Having designed part A of the circuit, you can now move on to tool usage to implement the design. Refer to the Intel Programmable Logic Software Manual if you have not installed the iPLS software.

In order to invoke iPLS type the following command

```
C:\IPLS>IPLS <Enter>
```

The iPLS menu will appear as shown in Screen 1.

The number to the left of each function allows you to select a function with a function key. Two kinds of function keys are available: toggle keys and field keys. <F3> and <F4> are toggle keys. All other keys are field keys. Functions beyond <F10> are executed by pressing the <Shift> key together with the function key. Press <F3> to invoke the Logic Builder and observe the Logic Builder menu as shown in Screen 2.

The first prompt asks for the file name. If the file already exists, its header information and primary inputs and outputs are displayed. If you enter a new file name, the Logic Builder module prompts for all the functions remaining on the screen.

```
Enter: DICE1 <Enter>
Create New Netlist(Y/N):Y
```

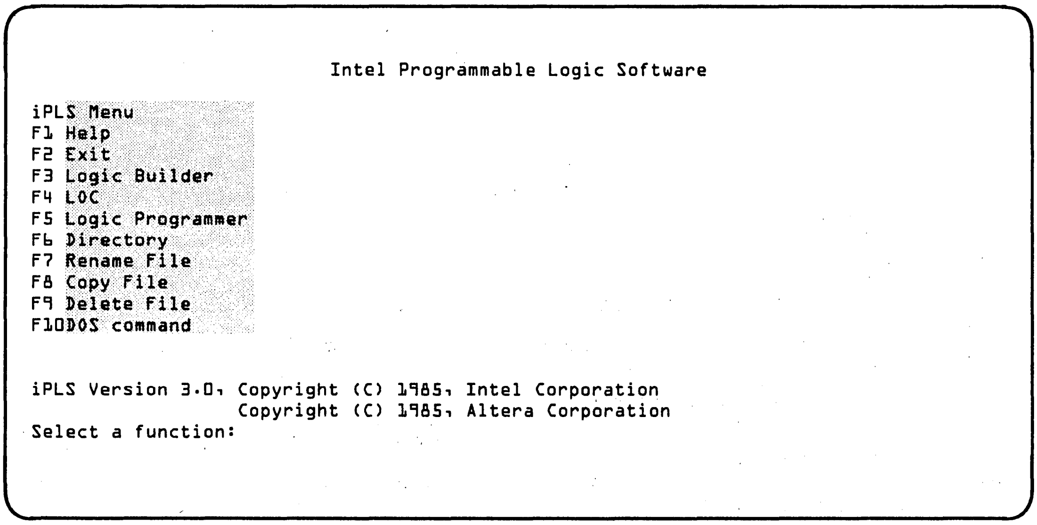
In this sample session, user entries are all in uppercase letters. Note: IPLS is case sensitive.

When initially invoked, the Logic Builder module displays its configuration menu. The Logic Builder configuration menu shows "5C121" as the default Intel part and, on the right side of the menu, displays those primitives that are legal for use with the 5C121. As soon as you enter another part (e.g. 5C060) the list of primitives changes to display the primitives applicable to that specific part.

Press <F6> and enter 5C060 when prompted for user entry.

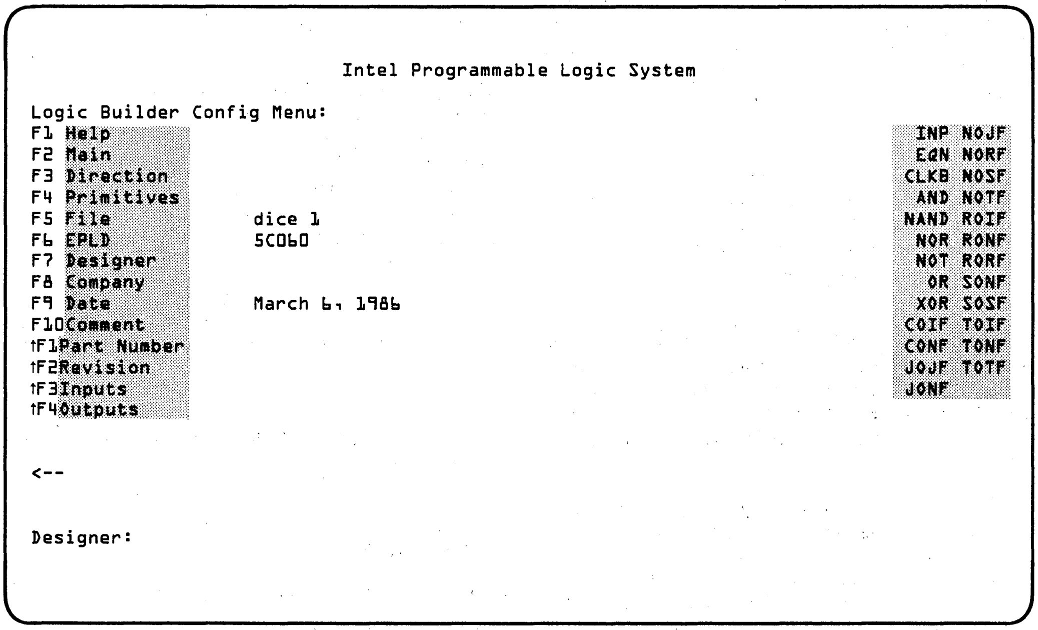
Screen 2 shows the Logic Builder Configuration Menu for the 5C060.

- The left side of the screen shows a menu of functions, each preceded by a function key number.



```
                        Intel Programmable Logic Software

iPLS Menu
F1 Help
F2 Exit
F3 Logic Builder
F4 LOC
F5 Logic Programmer
F6 Directory
F7 Rename File
F8 Copy File
F9 Delete File
F10DOS command


iPLS Version 3.0, Copyright (C) 1985, Intel Corporation
                  Copyright (C) 1985, Altera Corporation
Select a function:
```

Screen 1.



```
                        Intel Programmable Logic System

Logic Builder Config Menu:
F1 Help                                                          INP NOJF
F2 Main                                                          EQN NORF
F3 Direction                                                    CLKB NOSF
F4 Primitives                                                    AND NOTF
F5 File          dice 1                                         NAND ROIF
F6 EPLD          5C060                                           NOR RONF
F7 Designer                                                      NOT RORF
F8 Company                                                        OR SONF
F9 Date          March 6, 1986                                   XOR SOSF
F10Comment                                                      COIF TOIF
↑F1Part Number                                                  CONF TONF
↑F2Revision                                                     JOJF TOTF
↑F3Inputs                                                       JONF
↑F4Outputs


<--


Designer:
```

Screen 2.

Table 4.

Prompt	User Entry
F6 EPLD	5C060
F7 Designer	Your Name
F8 Company	Your Company
F9 Date	Present Date
F10 Comment	Our first design
↑F1 Part Number	0.1
↑F2 Revision	1.0
↑F3 Inputs	CLOCK@1
↑F4 Outputs	DICE1A@10,DICE1B@9,DICE1C@8,DICE1D@7

- The right side of the screen shows the list of available primitives (these are discussed in detail later).
- The two lines at the bottom of the screen are designated for comments (first line) and prompts (second line).
- The center of the screen is used to show a representation of the primitive; name and pictorial representation are in the middle, input signals are to the left, and output signals are to the right of the primitive.
- The direction of the arrow located on the left side of the screen below the list of functions determines the starting point and direction of design entry. If the arrow points to the left, entry is from output pins to input pins. If the arrow points to the right, entry is from input pins to output pins.

NOTE

We have assigned pin numbers to pin names by using the "@" symbol within the name of the logic variable. Specific pin numbers need not be assigned if not desired. In that case, the Logic Builder will assign its pin numbers for you.

Type in the information as given in Table 4 in the Logic Builder Config Menu. The information is also shown in Screen 3. After entering all of this required information, iPLDS will automatically prompt you through defining the circuit, starting with a primitive to drive the last output specified.

Once in the Logic Builder main menu, you are guided with prompts to enter information as follows:

Enter the name of the primitive to connect to the first node. The name may be entered by typing the name of the primitive, which highlights the appropriate primitive on the right side of the menu, then pressing <Enter>.

Subsequently, a representation of the primitive is displayed in the center of the screen surrounded by input and output signals. You are prompted for names of nodes to connect to each of the signals. The Design Primitives library contains approximately 80 basic functional blocks needed for designing circuits in programmable logic products.

Design Primitives are divided into the following groups:

- Input Primitives (INP,LINP)
- Logic Primitives (AND,GND,CLKB,NOT,VCC,OR,NAND,NOR,XOR)
- Equation Primitives (EQN)
- I/O Primitives (JOJF, NOJF, NORF, RORF, etc)

Refer to Appendix A for an explanation of the Primitives used in this example.

The logic is based on input clock transitions. At the rising edge of the clock we want the LEDs to generate a particular state depending on the input state. You want the output of the LEDs to follow the input, which is basically a D-TYPE FLIP-FLOP. You also require the feedback to generate the next state, which means that you should use a D-TYPE FLIP-FLOP with FEEDBACK or RORF as shown in Screen 4.

NOTE

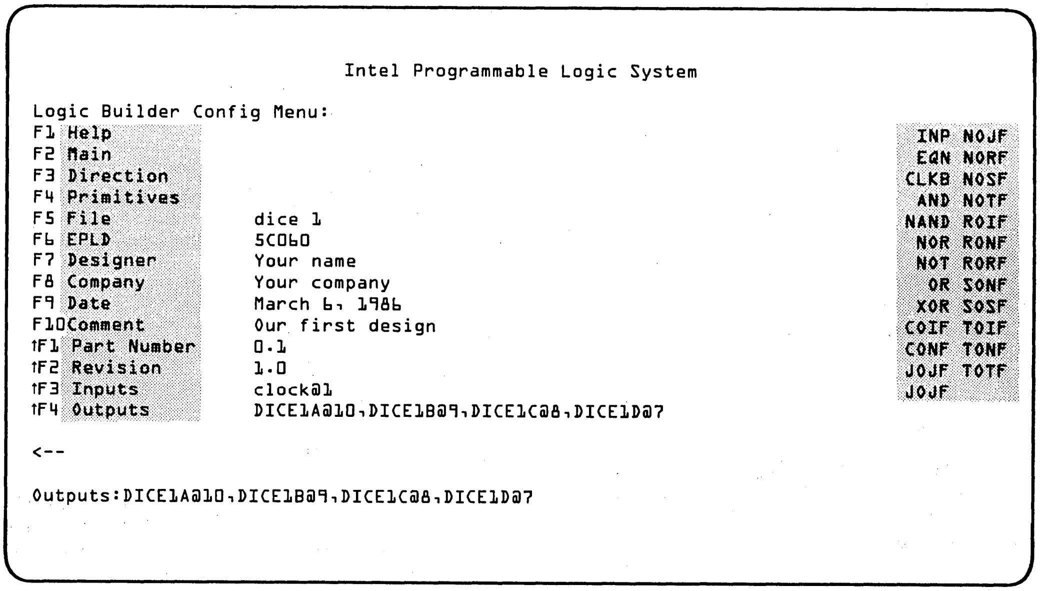
The Logic Builder module starts with the last output entered.

When you are prompted to select a primitive to drive DICE1D enter:

```
Select a primitive to drive DICE1D@7:
RORF <Enter>
```

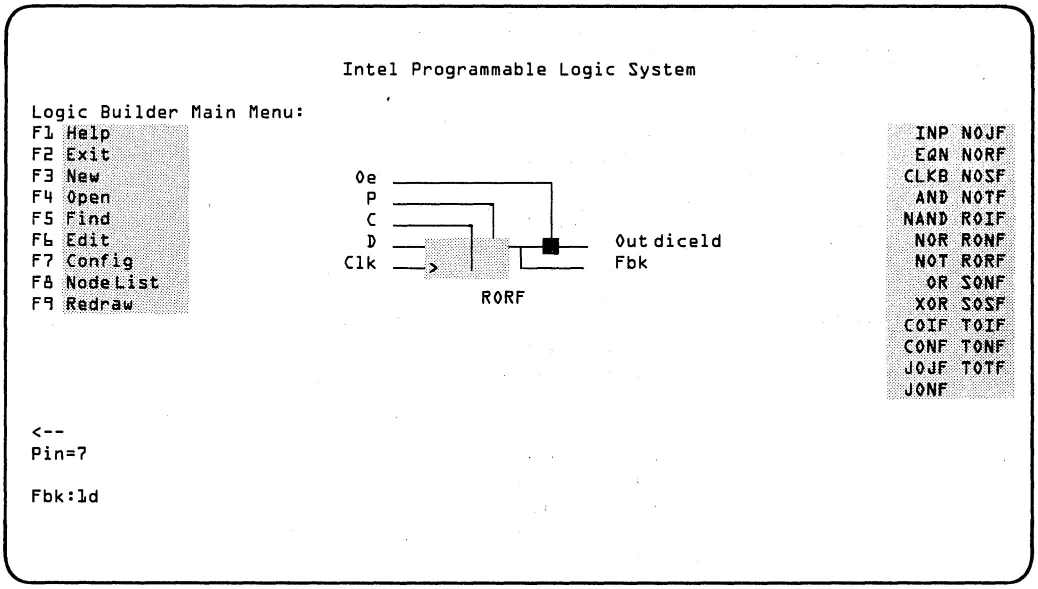
Now you are prompted for the remaining connections:

```
For FBK: 1D <Enter>

For OE, P, C: Press <Enter> (VCC, GND are
the defaults).

For D: IN1D <Enter>

For CLK: CLOCK <Enter>

Select a primitive to drive CLOCK: INP
<Enter>
```



```
                         Intel Programmable Logic System

Logic Builder Config Menu:
F1 Help                                                                INP  NOJF
F2 Main                                                                EQN  NORF
F3 Direction                                                          CLKB  NOSF
F4 Primitives                                                          AND  NOTF
F5 File                   dice 1                                      NAND  ROIF
F6 EPLD                   5C060                                        NOR  RONF
F7 Designer               Your name                                    NOT  RORF
F8 Company                Your company                                  OR  SONF
F9 Date                   March 6, 1986                                XOR  SOSF
F10Comment                Our first design                            COIF  TOIF
↑F1 Part Number           0.1                                         CONF  TONF
↑F2 Revision              1.0                                         JOJF  TOTF
↑F3 Inputs                clock@1                                     JOJF
↑F4 Outputs               DICE1A@10,DICE1B@9,DICE1C@8,DICE1D@7

<--

Outputs:DICE1A@10,DICE1B@9,DICE1C@8,DICE1D@7
```

Screen 3.



```
                         Intel Programmable Logic System

Logic Builder Main Menu:
F1 Help                                                                INP  NOJF
F2 Exit                                                                EQN  NORF
F3 New                                                                CLKB  NOSF
F4 Open                                                                AND  NOTF
F5 Find                                                               NAND  ROIF
F6 Edit                                                                NOR  RONF
F7 Config                                                              NOT  RORF
F8 NodeList                                                             OR  SONF
F9 Redraw                                                              XOR  SOSF
                                                                      COIF  TOIF
                                                                      CONF  TONF
                                                                      JOJF  TOTF
                                                                      JONF

<--
Pin=7

Fbk:1d
```

Screen 4.

```
In: CLOCK <Enter>
```

```
Select a primitive to drive IN1D: EQN
<Enter>
```

After you are prompted for the equation, type it in as derived in the Problem Set-up section. Please note that "/" indicates a logical "NOT", "*" indicates a logical "AND", and " + " indicates a logical "OR". The equation is terminated by a ";" as shown in Screen 5.

```
IN1D = (1A * 1B * 1C * /1D); <Enter>
```

The following prompts and design entries, as shown in Table 5, are needed to complete the design entries for DICE1C, DICE1B, and DICE1A respectively.

The Logic Builder will stop prompting for primitives once you have entered the complete design.

Press <F8> to show the design so far as shown in Screen 6.

Press <F2> to exit.

The Logic Builder main menu is cleared, replaced by the Logic Builder exit menu.

To save the configuration and return to iPLS menu you must press <F6> (Save-Exit).

Note that you are saving the Advanced Design File (ADF) that is generated by the Logic Builder.

You can print the ADF file that has been created at the end of this session if you so desire. You can use <F10> when in the iPLS main menu to print the ADF file for a listing. You can verify your file with the DICE1.ADF file given in Appendix D. If you desire a listing, while you are in the iPLS main menu, type the following:

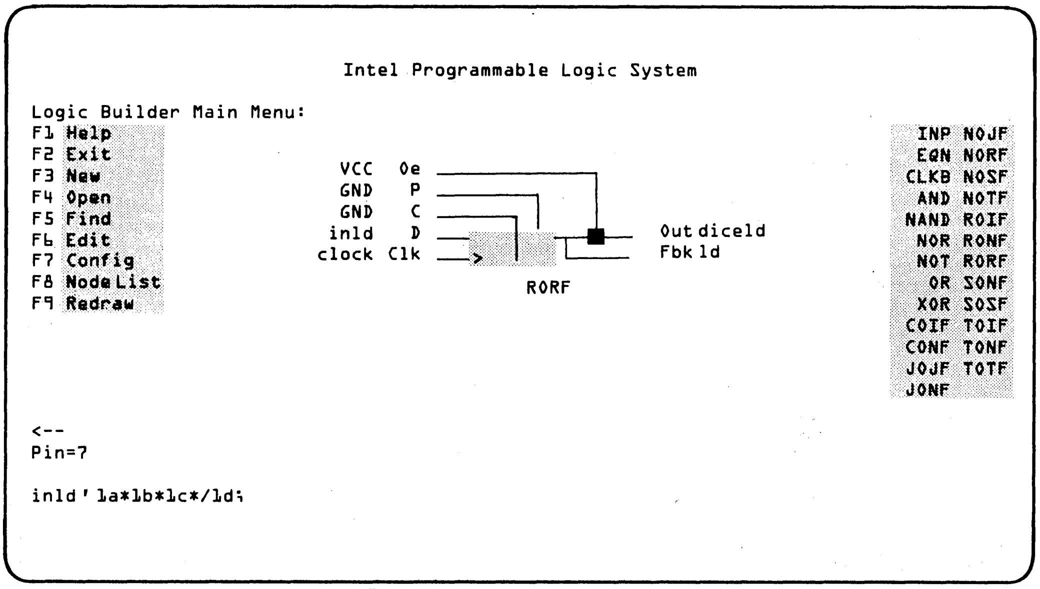
```
<F10> <Enter>

PRINT DICE1.ADF <Enter>
```

Submitting the ADF to the LOC

This ADF file is now compiled using the Logic Optimizing Compiler. To enter the ADF created with the Logic Builder module into the Logic Optimizing Compiler (LOC), press <F4> to access the LOC menu.

Table 5.

PROMPT	USER ENTRY
Select a primitive to drive 1C:	RORF <Enter>
Out:	DICE1C <Enter>
Oe:	VCC<Enter>
P:	GND <Enter>
C:	GND <Enter>
D:	IN1C <Enter>
Select a primitive to drive IN1C:	EQN <Enter>
IN1C:	(1A*1B*/1C*/1D)+(/1A*1B*1C*/1D)+(1A*1B*1C*/1D);
	<Enter>
Select a primitive to drive 1B:	RORF <Enter>
Out:	DICE1B <Enter>
Oe:	VCC <Enter>
P:	GND <Enter>
C:	GND <Enter>
D:	IN1B <Enter>
Select a primitive to drive IN1B:	EQN <Enter>
IN1B:	(1A*/1B*/1C*/1D)+(/1A*1B*/1C*/1D)+(1A*1B*/1C*/1D)
	+(/1A*1B*1C*/1D)+(1A*1B*1C*/1D);<Enter>
Select a primitive to drive 1A:	RORF <Enter>
Out:	DICE1A <Enter>
Oe:	VCC <Enter>
P:	GND <Enter>
C:	GND <Enter>
D:	IN1A <Enter>
Select a primitive to drive IN1A:	EQN <Enter>
IN1A:	(/1A*1B*/1C*/1D)+(/1A*1B*1C*/1D)+(/1A*1B*1C*1D)
	+(/1A*/1B*/1C*/1D);<Enter>



```
<--
Pin=7

inld ' 1a*1b*1c*/1d;
```

Screen 5.

Once the LOC menu is displayed, you are prompted through the LOC menu functions as follows:

The Input Format prompts you to specify your form of input: If input is in the form of a pinlist as output by DASH-2, enter P, if input is an Advanced Design File, enter an ADF or press <Enter> (ADF is the default). If output is a component list from PCAD, enter C.

```
INPUT FORMAT: A <Enter>

FILE NAME: DICE1 <Enter>

MINIMIZATION: <Enter to select default>

INVERSION CONTROL: <Enter to select
default>

LEF ANALYSIS: <Enter to select default>
```

After you have answered all the prompts, you are asked if you wish to run under the above conditions as shown in Screen 7.

```
DO YOU WISH TO RUN UNDER THE ABOVE CONDI-
TIONS [Y/N]?
```

Enter: Y

Finally you are prompted with:

```
WOULD YOU LIKE TO IMPLEMENT ANOTHER DE-
SIGN [Y/N]?
```

Enter: N

Note that the LOC generates a synopsis of its progress as shown in Screen 8. You are returned to the iPLS menu.

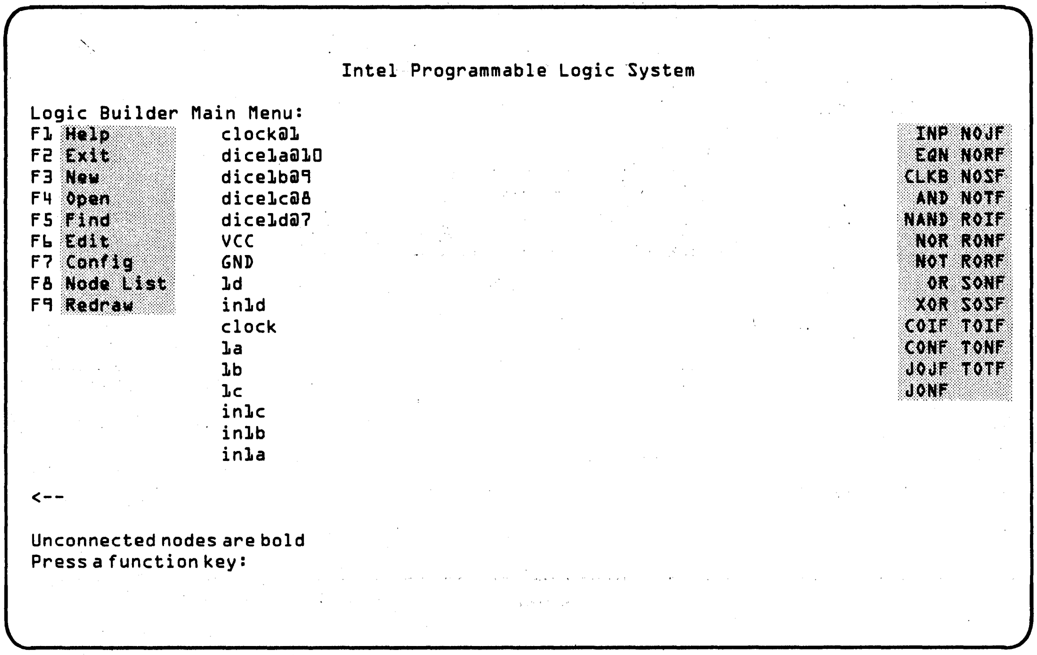
At the end of the LOC a JEDEC Standard File has been created which we will use in the Logic Programmer, DICE1.JED.

Also at the end of the LOC a report file is created, DICE1.RPT, which gives the pin configuration menu of the device. The DICE1.RPT file is given in Appendix D.

Programming the EPLD

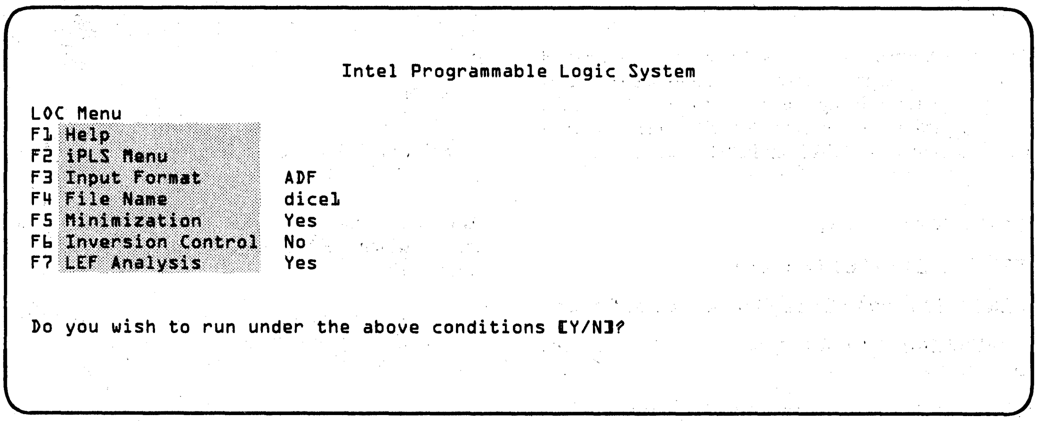
Finally, you submit your design to the Logic Programmer. In order for you to use the Logic Programmer, you must have the programming card plugged in. Please refer to the Intel Programmable Logic Software User Manual for installation instructions.

Alternatively you can use Intel's GUPI (Generic Universal Programmer Interface) to program your device.



```
                         Intel Programmable Logic System

Logic Builder Main Menu:
F1 Help          clock@1                                              INP  NOJF
F2 Exit          dice1a@10                                            EQN  NORF
F3 New           dice1b@9                                            CLKB  NOSF
F4 Open          dice1c@8                                             AND  NOTF
F5 Find          dice1d@7                                            NAND  ROIF
F6 Edit          VCC                                                  NOR  RONF
F7 Config        GND                                                  NOT  RORF
F8 Node List     1d                                                    OR  SONF
F9 Redraw        in1d                                                 XOR  SOSF
                 clock                                               COIF  TOIF
                 1a                                                  CONF  TONF
                 1b                                                  JOJF  TOTF
                 1c                                                  JONF
                 in1c
                 in1b
                 in1a

<--

Unconnected nodes are bold
Press a function key:
```

Screen 6.

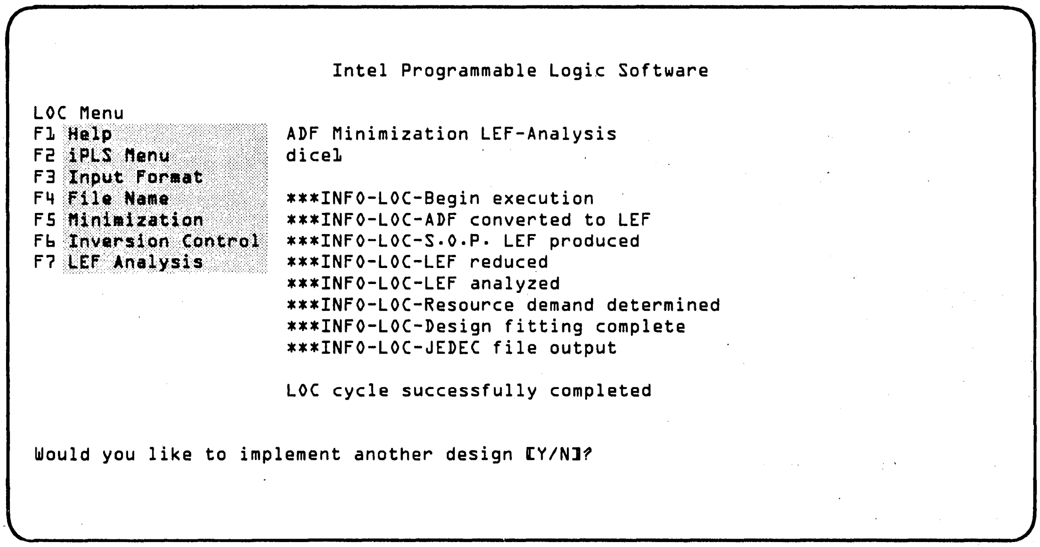


```
                         Intel Programmable Logic System

LOC Menu
F1 Help
F2 iPLS Menu
F3 Input Format        ADF
F4 File Name           dice1
F5 Minimization        Yes
F6 Inversion Control   No
F7 LEF Analysis        Yes

Do you wish to run under the above conditions [Y/N]?
```

Screen 7.

The iUP-GUPI and assorted GUPI LOGIC adaptors provide an alternative programming solution for Intel's H-series and EPLD devices, when purchased with the iPLS. This complete set of software is available without the Logic Programmer pod and the IBM interface card.

While you are still in the iPLS menu, press <F5>. This function allows you to access the Logic Programmer Software. The Logic Programmer will now come up as shown in Screen 9.



```
                         Intel Programmable Logic Software

LOC Menu
F1 Help                 ADF Minimization LEF-Analysis
F2 iPLS Menu            dice1
F3 Input Format
F4 File Name            ***INFO-LOC-Begin execution
F5 Minimization         ***INFO-LOC-ADF converted to LEF
F6 Inversion Control    ***INFO-LOC-S.O.P. LEF produced
F7 LEF Analysis         ***INFO-LOC-LEF reduced
                        ***INFO-LOC-LEF analyzed
                        ***INFO-LOC-Resource demand determined
                        ***INFO-LOC-Design fitting complete
                        ***INFO-LOC-JEDEC file output

                        LOC cycle successfully completed

Would you like to implement another design [Y/N]?
```

Screen 8.

Use the cursor keys to select "Program Device" option.

When you are prompted

Enter JEDEC file name

Enter: `DICE1.JED <Enter>`

When you are prompted for:

Select Device For Programming

Enter: `5C060 <Enter>`

When you are prompted for:

Do you wish to enable verify protection? [Y/N]?

Enter: N

When you are prompted for:

Do you wish to enable turbo-bit? [Y/N]?

Enter: N

Once you have answered all the prompts, the device is programmed and ready to be used in an actual circuit, as shown in Screen 10.

Exit from the Logic Programmer after saving the JEDEC file by using the "EXIT" option.

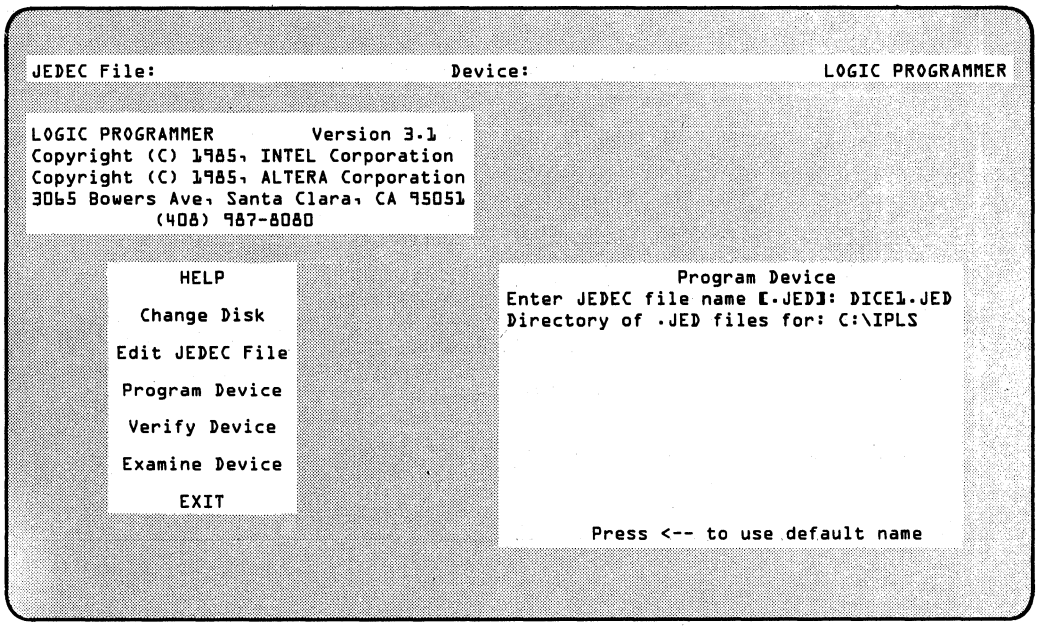
This completes part A of the design, which was to roll a single dice. The programmed device can be tested as described in Appendix C.

PART B

Now that you have a good understanding of the manner in which a circuit is designed and also a good understanding of how the programming tools are used to program the device, you can proceed to the next step in the five stages of the dice design. According to the truth table generated in part A, the dice will roll a number between 1 and 6 inclusive as long as you supply a power source. When you disconnect the power source, all the LEDs will turn off. This will not be much help since you can only see the dice roll, but not actually see a number displayed.

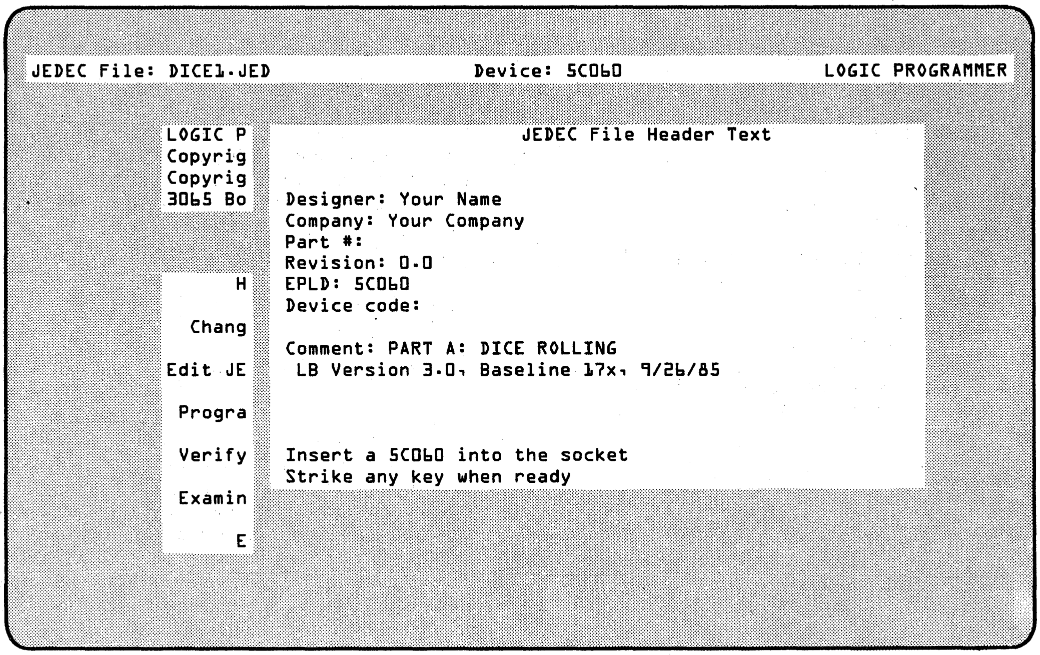
Let us include an additional feature into the rolling dice. Let us include a switch to control the rolling and display of the dice.

You could choose to gate the clock of the dice or add the necessary inputs to the product terms to effect this design. If you were to stop after this step, then gating the clock would be a simpler choice, however, you will require the dice to roll during part D of the design; so we will choose to add product terms at this stage. This also results in a better engineering solution since gated clocks often cause problems in large systems, and it has been shown that synchronous systems are more reliable.



```
JEDEC File:                           Device:                       LOGIC PROGRAMMER

LOGIC PROGRAMMER        Version 3.1
Copyright (C) 1985, INTEL Corporation
Copyright (C) 1985, ALTERA Corporation
3065 Bowers Ave, Santa Clara, CA 95051
          (408) 987-8080

        HELP                                    Program Device
     Change Disk                  Enter JEDEC file name [.JED]: DICE1.JED
   Edit JEDEC File                Directory of .JED files for: C:\IPLS
    Program Device
    Verify Device
    Examine Device
        EXIT
                                          Press <-- to use default name
```

Screen 9.



```
JEDEC File: DICE1.JED                   Device: 5C060               LOGIC PROGRAMMER

LOGIC P                        JEDEC File Header Text
Copyrig
Copyrig
3065 Bo  Designer: Your Name
         Company: Your Company
         Part #:
         Revision: 0.0
      H  EPLD: 5C060
         Device code:
  Chang
         Comment: PART A: DICE ROLLING
Edit JE   LB Version 3.0, Baseline 17x, 9/26/85

 Progra

 Verify  Insert a 5C060 into the socket
         Strike any key when ready
 Examin

      E
```

Screen 10.

Since you already have a proven design of a rolling dice from part A, we shall use the Logic Builder and edit that design. You may wish to save the original design at this stage. You can do this by using the <F10> key in the Main Menu. Press <F10> and issue the following command before re-entering the iPLS menu:

```
COPY DICE1.* DICE1A.*
```

The truth table is shown in Table 6.

Now you can use the iPLDS to design and program the device.

Go through the same steps to program the device as in Part A of the design example. Use the Logic builder, the Logic Optimizing Compiler, and the Logic Programmer respectively. The Logic Optimizing Compiler and the Logic Programmer steps are identical to the corresponding steps explained in part A of the design example. However, the Logic Builder will be used to edit the existing file, DICE1, to include the switch feature as follows:

Invoke the Logic Builder Menu from the iPLS main menu by pressing the <F3> key. Once you obtain the Logic Builder Configuration Menu, type in DICE1 as your input file name.

Use (Shift)(F3) to get the Inputs option and then add switch at pin #2 to it.

Inputs: `CLOCK, SWITCH@2 <Enter>`

Now press <F2> to exit to the Logic Builder Main Menu and answer the prompts as given in Table 7.

All that is left to do now is to edit the four equations, IN1A, IN1B, IN1C, IN1D to add the SWITCH option to it. Edit the four equations as follows:

Edit Function

When you press the "Edit" function key, <F6>, while in the main menu, the edit menu is displayed on the left side of the screen as shown in Screen 11. If you wish to edit an EQN Primitive displayed on the screen, press <F6>. Then the equation is moved to the prompt line where it can be edited.

Hence, the Boolean expressions for this case would consider the situations of when the switch was ON as well as OFF. The Boolean equations would contain the expression for the switch as follows.

```
DICE1A = ((1A*/1B*/1C*/1D)+(1A**1B*/1C*/1D)
         +(1A*1B*1C*/1D)
         +(/1A*/1B*/1C*/1D))*/SWITCH
         +(((/1A*1B*/1C*/1D)+(/1A*1B*1C*/1D)
         +(/1A*1B*1C*1D)
         +(/1A*/1B*/1C*/1D))*SWITCH

DICE1B = (((/1A*1B*/1C*/1D)+(1A*1B*/1C*/1D)
         +(/1A*1B*1C*/1D)+(1A*1B*1C*/1D)
         +(/1A*1B*1C*1D))*/SWITCH
         +((1A*/1B*/1C*/1D)
         +(/1A*1B*/1C*/1D)+(1A*1B*/1C*/1D)
         +(/1A*1B*1C*/1D)
         +(1A*1B*1C*/1D))*SWITCH

DICE1C = (((/1A*1B*1C*/1D)
         +(1A*1B*1C*/1D)
         +(/1A*1B*1C*1D))*/SWITCH
         +((1A*1B*/1C*/1D)+(/1A*1B*1C*/1D)
         +(1A*1B*1C*/1D))*SWITCH

DICE1D = (/1A*1B*1C*1D)*/SWITCH
         +(1A*1B*1C*/1D)*SWITCH
```

The equation primitive must be displayed on the screen in order to edit that equation. In order to display the equation on the screen, use the "Find" command, <F5>, to find it.

The "Find" command prompts for a node name: then searches the design for that node and displays it. If the direction arrow points to the left, the primitive on the output side of the node is shown. If the direction arrow points to the right, the first primitive on the input side is shown.

After you have modified all four equations to include the SWITCH feature, return to the iPLDS main menu using the <F5> key and save the design using the <F6> key. You can verify your ADF file with the ADF file for part B given in Appendix D.

The file is ready to be compiled using the LOC, and the device is ready to be programmed using the LP.

The steps required to use the LOC and the LP are identical to the steps in part A.

Now the device that has been programmed is ready to be tested. At this stage in the design, you have completed part B of the design which is to add a switch to give the roll/no-roll option.

The programmed device can be tested as described in Appendix C.

Let us summarize before moving on to the next part of the design.

Table 6. Truth Table for DICE1

	Input State				Output State							
SWITCH	1A	1B	1C	1D	1A	1B	1C	1D	1A	1B	1C	1D
					Valid state				Invalid state			
REMAIN IN THE SAME STATE												
0	1	0	0	0	1	0	0	0(1)				
0	0	1	0	0	0	1	0	0(2)				
0	1	1	0	0	1	1	0	0(3)				
0	0	1	1	0	0	1	1	0(4)				
0	1	1	1	0	1	1	1	0(5)				
0	0	1	1	1	0	1	1	1(6)				
CONTROL THE INVALID STATES												
0	0	0	1	0					0	0	0	0
0	1	0	1	0					0	0	0	0
0	0	0	0	1					0	0	0	0
0	1	0	0	1					0	0	0	0
0	0	1	0	1					0	0	0	0
0	1	1	0	1					0	0	0	0
0	0	0	1	1					0	0	0	0
0	1	1	1	1					0	0	0	0
0	0	0	0	0					1	0	0	0(1)
CHANGE TO THE NEXT VALID STATE*												
1	1	0	0	0(1)	0	1	0	0(2)				
1	0	1	0	0(2)	1	1	0	0(3)				
1	1	1	0	0(3)	0	1	1	0(4)				
1	0	1	1	0(4)	1	1	1	0(5)				
1	1	1	1	0(5)	0	1	1	1(6)				
1	0	1	1	1(6)	1	0	0	0(1)				
CONTROL THE INVALID STATES												
1	0	0	1	0				0	0	0	0	
1	1	0	1	0				0	0	0	0	
1	0	0	0	1				0	0	0	0	
1	1	0	0	1				0	0	0	0	
1	0	1	0	1				0	0	0	0	
1	1	1	0	1				0	0	0	0	
1	0	0	1	1				0	0	0	0	
1	1	0	1	1				0	0	0	0	
1	1	1	1	1				0	0	0	0	
1	0	0	0	0				1	0	0	0(1)	

Note: This part of the truth table is identical to Table 3.

We have briefly discussed the EPLD and the IPLDS family of parts. We have also defined the design problem. We have implemented the design using the state equations and the truth table, edited an existing design to add features, and actually programmed a device using the Logic Builder, Logic Optimizing Compiler, and the Logic Programmer.

Our logic in implementing the dice example is to use the LED pairs in outputs 1A, 1B, 1C, and 1D respectively as

Table 7.

Prompts	User Entry
Select a primitive for switch@2 to drive:	INP <Enter>
Out:	SWITCH<Enter>
Select a primitive for switch to drive:	EQN<Enter>

shown in Figure 8. These LEDs are lit up to generate numbers between 1 and 6 inclusive. We are using a D-TYPE FLIP-FLOP to implement the truth table. The clock is a free running clock. A push button switch is also supplied to give the roll/no-roll option. Whenever the switch is ON, the LEDs roll, and when the switch is OFF, the LEDs display a number between 1 and 6, as long as the clock is supplied to the device.

After seeing the dice roll and display a number, you can either quit or move onto parts C, D and E of the design process. The following three parts describe a versatile use of the EPLD concept.

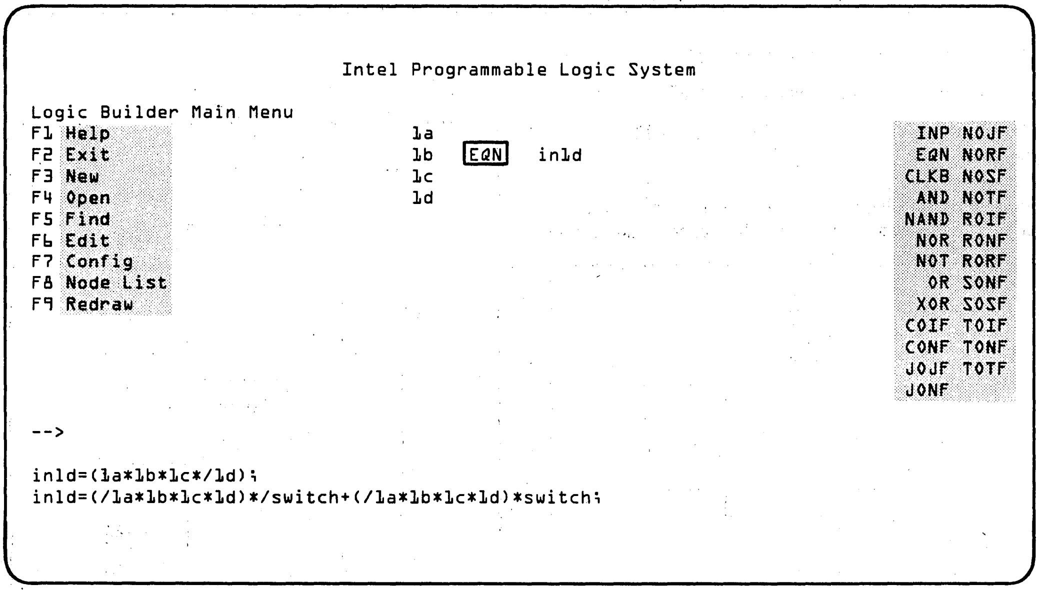
PART C

We are using an EPLD 5C060 which is a 24 pin, 600 gate device. It has four dedicated input pins and 16 input/output pins. Up to this point you have used only one input pin which is the switch and only four input/output pins for the four LEDs 1A, 1B, 1C, 1D.

Part C of the design is to include a second dice with the first dice. This is a step towards real-world application since dice are usually rolled in pairs. At the end of this section, you will have a pair of dice rolling and displaying a pair of numbers. All the conditions and truth tables and Boolean expressions that were designed for part B, hold good for DICE1. The equations for DICE2 would change slightly as explained below.

You have designed a 6 state counter and can define a carry out (fortunately you can use state 6 and do not require extra logic). You can use the carry out as an enable input to form two cascaded counters.

The carry out of 1D is used as an enable input to DICE2. Hence, 1D performs the same function as the push button switch performed in dice 1. Therefore, whenever 1D is enabled or logic high, DICE2 is enabled and rolls a number. DICE2 displays the number when 1D is disabled or logic is low. This configuration is shown in Figure 9.



```
-->

in1d=(1a*1b*1c*/1d);
in1d=(/1a*1b*1c*1d)*/switch+(/1a*1b*1c*1d)*switch;
```

Screen 11.

Table 8.

PROMPTS	USER ENTRY
Find:	IN1D <Enter>
(Now use the <cursor left> key to obtain the EQN Primitive.)	
Edit:	
IN1D = (/1A*1B*1C*1D)*/SWITCH+(1A*1B*1C*/1D)*SWITCH;<Enter>	
Find:	IN1C <Enter>
(Now use the <cursor left> key to obtain the EQN Primitive.)	
Edit:	
IN1C = ((/1A*1B*1C*/1D)+(1A*1B*1C*/1D)+(/1A*1B*1C*1D))*/SWITCH	
+((1A*1B*/1C*/1D)+(/1A*1B*1C*/1D)+(1A*1B*1C*/1D))*SWITCH;	
<Enter>	
Find:	IN1B <Enter>
(Now use the <cursor left> key to obtain the EQN Primitive.)	
Edit:	
IN1B = ((/1A*1B*/1C*/1D)+(1A*1B*/1C*/1D)+(/1A*1B*1C*1D)+(1A*1B*1C*/1D)	
+(/1A*1B*1C*1D))*/SWITCH	
+((1A*/1B*/1C*/1D)+(/1A*1B*/1C*/1D)+(1A*1B*/1C*/1D)+	
(/1A*1B*1C*/1D)+(1A*1B*1C*/1D))*SWITCH;<Enter>	
Find:	IN1A <Enter>
(Now use the <cursor left> key to obtain the EQN Primitive.)	
Edit:	
IN1A=((1A*/1B*/1C*/1D)+(1A*1B*/1C*/1D)+(1A*1B*1C*/1D)+	
(/1A*/1B*/1C*/1D))*/SWITCH+((/1A*1B*/1C*/1D)+(/1A*1B*1C*/1D)	
+(/1A*1B*1C*1D)+(/1A*/1B*/1C*/1D))*SWITCH;<Enter>	

The two conditions obtained are as follows:

When power is ON and 1D is enabled, DICE2 will roll.

When power is ON and 1D is disabled, DICE2 will display.

For DICE1, the logic conditions remain the same as in part A. Just as you used the switch to enable and disable

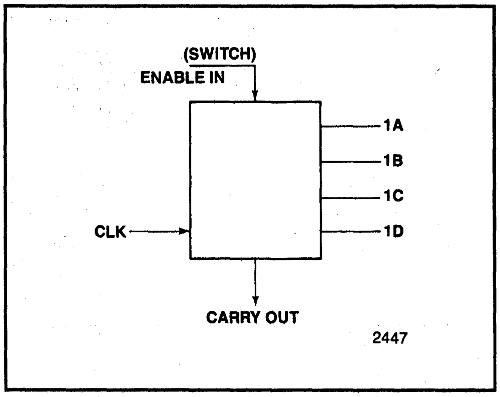


Figure 8.

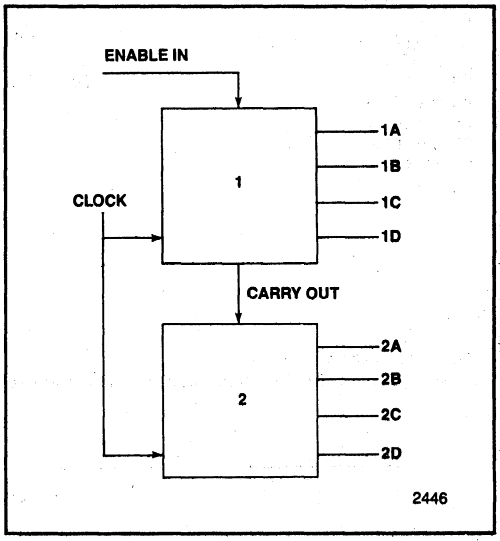


Figure 9.

DICE1, you will use the switch as well as the output of LED 1D to enable and disable DICE2; because the number on DICE2 is a function of both the switch and the present state of LED 1D, as explained above.

Now write down the truth table since the state diagrams can easily be inferred from the truth table. Please note that the truth table is identical to the one for DICE1 except for the switch input. For DICE2, you will have the combination of the switch and the 1D, as shown in Table 9.

The Boolean expressions for part C will consider the situation when the switch is ON as well as OFF and also 1D enabled or disabled respectively. The Boolean equations will contain the expression for the switch and LED 1D, as shown below.

```
DICE2A = ((2A*/2B*/2C*/2D)+(2A*2B*/2C*/2D)
         +(2A*2B*2C*/2D)+(/2A*/2B*/2C*/2D))
         *(/SWITCH*/1D)
         +((/2A*2B*/2C*/2D)+(/2A*2B*2C*/2D)
         +(/2A*2B*2C*2D)+(/2A*/2B*/2C*/2D))
         *(SWITCH*1D)

DICE2B = ((/2A*2B*/2C*/2D)+(2A*2B*/2C*/2D)
         +(/2A*2B*2C*/2D)+(2A*2B*2C*/2D)
         +(/2A*2B*2C*2D))*(/SWITCH*/1D)
         +((2A*/2B*/2C*/2D)
         +(/2A*2B*/2C*/2D)+(2A*2B*/2C*/2D)
         +(/2A*2B*2C*/2D)
         +(2A*2B*2C*/2D))*(SWITCH*1D)

DICE2C = ((/2A*2B*2C*/2D)+(2A*2B*2C*/2D)
         +(/2A*2B*2C*2D))*(/SWITCH*/1D)
         +((2A*2B*/2C*/2D)
         +(/2A*2B*2C*/2D)+(2A*2B*2C*/2D))
         *(SWITCH*1D)

DICE2D = (/2A*2B*2C*2D)*(/SWITCH*/1D)
         +(2A*2B*2C*/2D)*(SWITCH*1D)
```

Now you can use the iPLDS to program and test the device as explained in appendix C. At this stage in design, you have completed part C of the design which is to add a second DICE to the first one giving the the roll/no-roll option.

In part C of the design process, you have used one dedicated input which is the switch, and a total of eight output pins for the two pairs of LEDs, 1A, 1B, 1C, 1D and 2A, 2B, 2C, 2D respectively. You have also used the RORF primitive, since the design logic was the same for DICE2 as it was for DICE1. This leaves 3 dedicated inputs and 8 I/O pins on the 5C060 device.

You can stop the design now or go onto part D which gives the next option, which is adding the spin.

PART D

This is the fourth step in our design process and adds the spin option to the two dice that are rolling when the switch is pushed and display a number when the switch is released. The logic used to implement the spin concept is as follows:

When the power is ON and the switch is OFF, DICE1 and DICE2 display a random number according to the logic defined in parts B and C respectively.

But, when power is ON and the switch is ON, the two dice spin by lighting the LEDs B, C, and D. That is, DICE1 will light LEDs 1B, 1C, 1D while DICE2 will light LEDs 2B, 2C, and 2D. This pattern on the LEDs will generate the spinning pattern. The logic is shown in the truth table in Table 10. The schematic is shown in Figure 10.

As you can see from the truth table, when the present state is any of the three valid states, then the two dice will spin. The dice will also spin if the present state is an invalid state, because all the invalid states go to "0 0 0 0" in the next state. But from the truth table in Table 10, you see that this particular state is a valid state lighting LED C.

The spin frequency should be chosen to be visually appealing and should be high enough to ensure randomness of the dice. If we use the "carry out" state of DICE2, then the spin pattern will only change once for every combination of the two dice. This will ensure randomness. The "carry out" of DICE2 is signal 2d; we do not need extra terms to derive it.

Thus we have achieved our objective of adding the spinning option to the two dice.

The Boolean equations that are obtained from the above truth table are as follows:

```
SPIN1B = (SWITCH*2d*/S1D*/S1C*/S1B*S1A)

SPIN1C = (SWITCH*2d*/S1D*/S1C*/S1B*/S1A)

SPIN1D = (SWITCH*2d*/S1D*S1C*/S1B*/S1A)

SPIN2B = (SWITCH*2d*/S2D*/S2C*/S2B*S2A)

SPIN2C = (SWITCH*2d*/S2D*/S2C*/S2B*/S2A)

SPIN2D = (SWITCH*2d*/S2D*S2C*/S2B*/S2A)
```

Please note in the above equations that A, B, C, and D refer to both DICE1 and DICE2. For DICE1 the above set of equations would be 1A, 1B, 1C, and 1D. For DICE2 the above set of equations would be 2A, 2B, 2C, and 2D respectively. SD is the feedback obtained from IN D of both DICE1 and DICE2 respectively. If the switch is not ON, the dice will not spin and a random pair of numbers will be displayed by the two dice; but, if the switch is ON, then the two dice will spin according to the truth table and Boolean expression given in Table 10.

Table 9. Truth Table for DICE2

Input State					Output State							
(SWITCH*1D)	2A	2B	1C	2D	2A	2B	2C	2D	2A	2B	2C	2D
					Valid state				Invalid state			
REMAIN IN THE SAME STATE												
0	1	0	0	0	1	0	0	0(1)				
0	0	1	0	0	0	1	0	0(2)				
0	1	1	0	0	1	1	0	0(3)				
0	0	1	1	0	0	1	1	0(4)				
0	1	1	1	0	1	1	1	0(5)				
0	0	1	1	1	0	1	1	1(6)				
CONTROL THE INVALID STATES												
0	0	0	1	0					0	0	0	0
0	1	0	1	0					0	0	0	0
0	0	0	0	1					0	0	0	0
0	1	0	0	1					0	0	0	0
0	0	1	0	1					0	0	0	0
0	1	1	0	1					0	0	0	0
0	0	0	1	1					0	0	0	0
0	1	1	1	1					0	0	0	0
0	0	0	0	0					1	0	0	0(1)
CHANGE TO THE NEXT VALID STATE*												
1	1	0	0	0(1)	0	1	0	0(2)				
1	0	1	0	0(2)	1	1	0	0(3)				
1	1	1	0	0(3)	0	1	1	0(4)				
1	0	1	1	0(4)	1	1	1	0(5)				
1	1	1	1	0(5)	0	1	1	1(6)				
1	0	1	1	1(6)	1	0	0	0(1)				
CONTROL THE INVALID STATES												
1	0	0	1	0					0	0	0	0
1	1	0	1	0					0	0	0	0
1	0	0	0	1					0	0	0	0
1	1	0	0	1					0	0	0	0
1	0	1	0	1					0	0	0	0
1	1	1	0	1					0	0	0	0
1	0	0	1	1					0	0	0	0
1	1	0	1	1					0	0	0	0
1	1	1	1	1					0	0	0	0
1	0	0	0	0					1	0	0	0(1)

Note the extreme similarity between this truth table and the one given in Table 3.

Table 10. Truth Table to Spin Two Dice

Input State					Output State			
SWITCH	A	B	C	D	A	B	C	D
CHANGE TO THE NEXT VALID STATE								
1	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
1	0	0	1	0	0	0	0	1
1	0	1	0	0	0	0	0	0
ROLLING INTO A VALID STATE								
1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0
1	0	1	0	1	0	0	0	0
1	1	1	0	1	0	0	0	0
1	0	0	1	1	0	0	0	0
1	1	0	1	1	0	0	0	0
1	0	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0	0

We have chosen the following two primitives for part D:

Registered Output Registered Feedback (RORF)

No output JK Feedback (NOJF)

For the dice spinning option you will use the RORF and for the dice not spinning option you will use the NOJF, while using the Logic Builder.

When you add the spinning option to the pair of rolling dice, you obtain the following boolean equations. (These Boolean equations satisfy the requirements of the two dice spinning when the switch is on and displaying a number when the switch is off).

```
SPIN1A = (/SWITCH*1A)
SPIN1B = (/SWITCH*1B)
         + (SWITCH*2d*/S1D*/S1C*/S1B*S1A)
SPIN1C = (/SWITCH*1C)
         + (SWITCH*2d*/S1D*/S1C*/S1B*/S1A)
SPIN1D = (/SWITCH*1D)
         +(SWITCH*2d*/S1D*S1C*/S1B*/S1A)
SPIN2A = (/SWITCH*2A)
SPIN2B = (/SWITCH*2B)
         + (SWITCH*2d*/S2D*/S2C*/S2B*S2A)
SPIN2C = (/SWITCH*2C)
         + (SWITCH*2d*/S2D*/S2C*/S2B*/S2A)
SPIN2D = (/SWITCH*2D)
         +(SWITCH*2d*/S2D*S2C*/S2B*/S2A)
```

At the end of the design step, you have completed all the design steps. You can now program the device using iPLDS.

The correct ADF file is included in Appendix D for your reference. You can refer to it to verify the ADF file you have created.

The programmed device can be tested on:

- A PCB with slow clock

For information on this board and on testing your design, please refer to Appendix C.

It works!

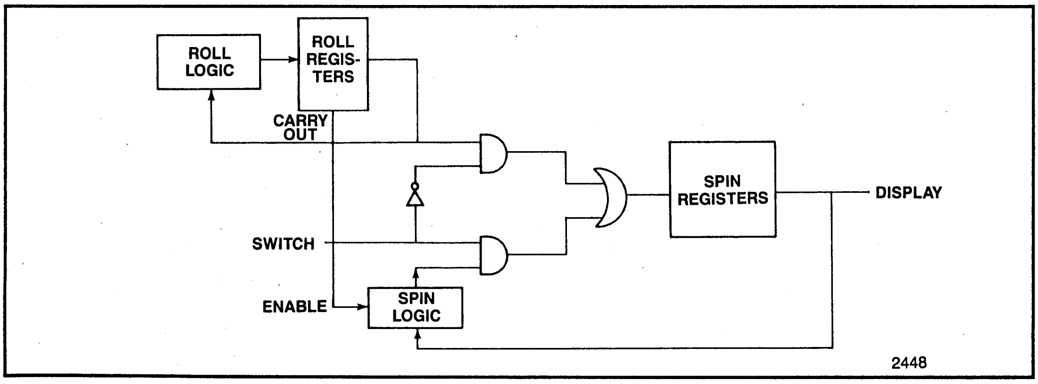


Figure 10.

LATE NEWS FLASH

The PCBs have been made and we have units in the field. Now Marketing wants the design updated! Field trials of the dice showed that the battery needed to last longer. A simple mod to the design, chop the drive to the LEDs, extends the battery life.

This is very simple using the EPLDs. Reprogram the EPLD and test it. Imagine how difficult it would have been without using EPLDs.

PART E

This step of the design process is to modify the existing circuit to add the power save feature which will extend the battery life. This can easily be done by chopping the drive to the LEDs. Chopping the drive to the LEDs can be done as follows:

When you designed the circuit and implemented it using the iPLS, you have set the output enable (Oe) to VCC supply. This means that the LEDs are enabled 100% of the time. You can "chop" the drive to the LEDs with a conveniant high (above 50Hz) signal that will not be visible to the human eye.

Next set Output enable (Oe) to the clock signal. Thus, depending on the clock input the LEDs will only be on 50% of the time and battery life is extended as required. You can easily modify the ADF file to change the Oe input from VCC to CLOCK and then test the design using the PCB as explained in Appendix C.

CONCLUSION

You should now have a comprehensive knowledge of Intel's EPLD and iPLDS family of devices.

With this knowledge you will be able to implement designs using the iPLDS tools.

Good Luck!

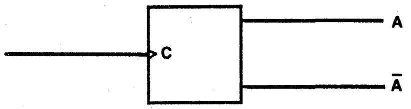
APPENDIX A:
BASIC DEFINITIONS

BASIC DEFINITIONS

Logic Design — A systematic procedure for realizing specified terminal characterisitics of digital networks, at either the device or system level.

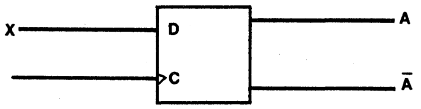
CLOCKED FLIP-FLOP — Output determined by the leading or trailing edge of clock pulse.

T FLIP-FLOP — Output changes value with every input clock pulse.



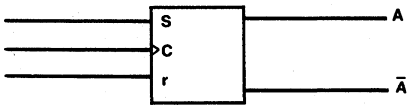
T FLIP-FLOP

D FLIP-FLOP — Output determined by the input signal when clock pulse present.



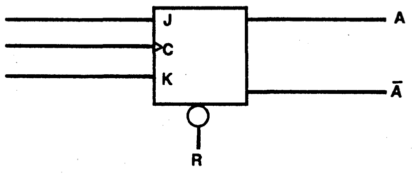
D FLIP-FLOP

S-R FLIP-FLOP — Output states synchronized with the clock pulse and controlled by the input signals, S and R.



S-R FLIP-FLOP

J-K FLIP-FLOP — Output states synchronized with the clock pulse and controlled by the input signals, J and K.



J-K FLIP-FLOP

COMBINATORIAL CIRCUIT — Output determined by current value of input signal.

REGISTERED CIRCUIT — Output determined by sequence of input signals.

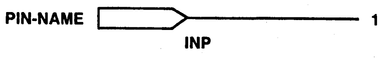
Intel Schematic Primitive — One of the basic functional blocks needed to design circuits for Intel programmable logic products.

Truth Table — A list of all the input-output possibilities of a logic circuit.

Boolean Logic — Describes logic that obeys the theorems of Boolean algebra. The Boolean portion of a design is that portion which can be implemented in the AND-OR matrix.

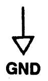
State Diagram — A diagram that shows the succession of output states through which the circuit passes as its input signals vary.

INP — Input



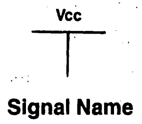
Input Primitive

GND — Ground.

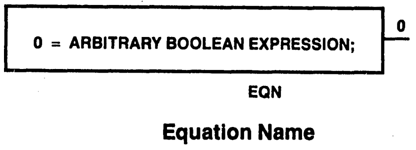


Ground Signal Name

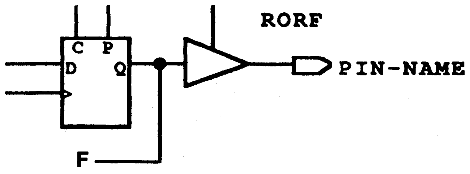
VCC — Signal



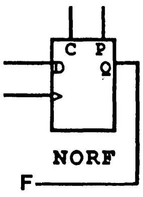
EQN — Equation



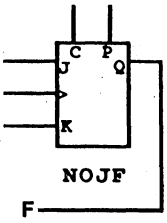
Registered Output Registered Feedback (RORF)



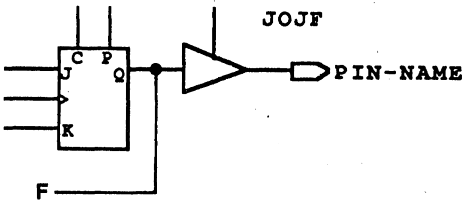
No Output Registered Feedback (NORF)



No Output JK Feedback (NOJF)



JK Output JK Feedback (JOJF)



Security Bit — A feature that prevents the device from being interrogated or being accidentally programmed.

Turbo-bit — A control bit that allows you to choose the speed and power characteristics of the device. If the inputs are static for approximately 50 ns and the Turbo-bit is not programmed, the device will enter power down mode. When the input changes, the device will take an extra 3-5 ns to wake-up and react to the change. Programming the Turbo-bit inhibits the power down.

Macrocell — A basic building block of Intel's programmable logic devices. A macrocell consists of two sections: combinatorial logic and output logic. The combinatorial logic allows a wide variety of logic functions. The output logic has two data paths: one leads to the other macrocells or feeds back to the macrocell itself: the other is configured as a pin configuration acting as input, output, or bi-directional I/O port on the chip.

Node — A wire connecting two or more primitives in a schematic.

Pin — A node that is connected to an input or I/O primitive on one end and a pin of the chip on the other end.

Product tem (P-Term) — Two or more factors in a boolean expression combined with the AND operator consitutes a logic product term.

JEDEC Standard File — An industry-wide standard for the transfer of information between a data preparation system and a logic device programmer.

EPLD PROGRAMMING TECHNIQUES

You can enter your design in the following ways

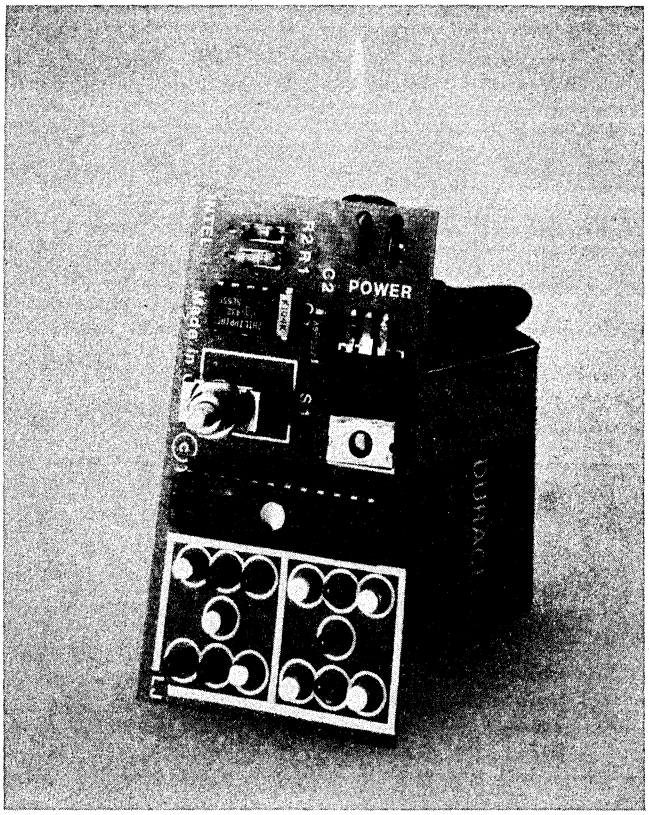
1. BOOLEAN EQUATION — entering the design in BOOLEAN equations or expressions.
2. NETLIST CAPTURE — selecting components and specifying interconnections until all elements are specified.
3. SCHEMATIC CAPTURE — using a mouse and menu driven environment.
4. STATE MACHINE — specifying states and conditional branches and also inputs/outputs to the state machines.

APPENDIX B: COMPONENTS LIST

COMPONENTS USED IN DESIGN

In order to implement the EPLD program, you should use the following:

- An 5C060 EPLD
- A pair of seven discrete LEDs (Dice 1, Dice 2)
- A timer to generate a clock signal (NE555)
- A voltage regulator to generate a fixed voltage of 5 volts (7805)
- A push button switch to control the spinning mechanism
- A 9-Volt DC battery source to generate the power supply
- Capacitors C1 = 0.1 MF, C2 = 0.01 MF
- Resistors R1 = 390K, R2 = 100K
- A PCB as explained in Appendix C



APPENDIX C: PCB DESCRIPTION

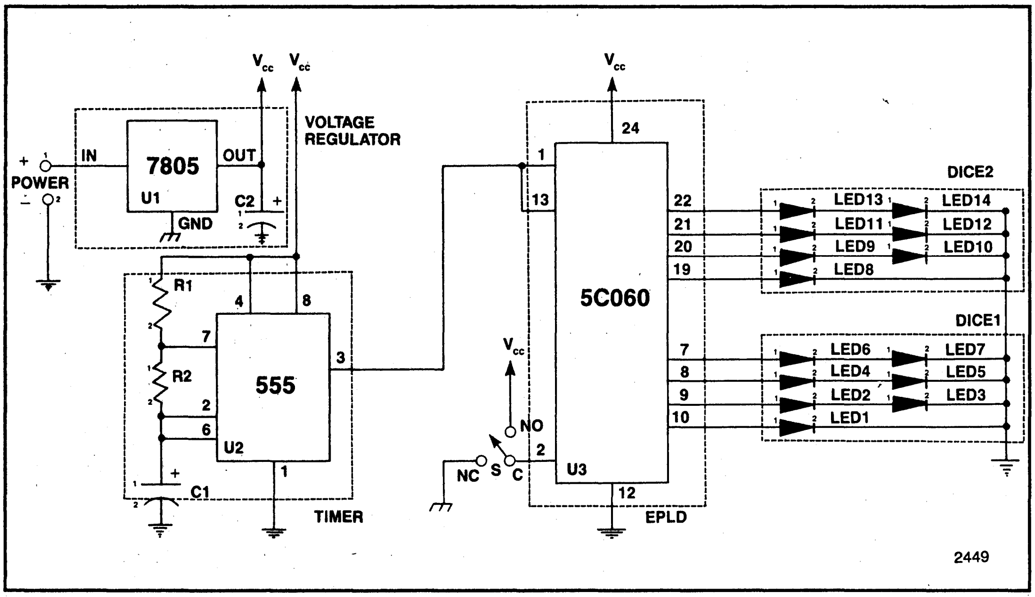


Figure C-1

You can test each part of your design using the PCB with a slow clock on it.

The PCB is a board that is very specific to the dice example. The PCB is portable, approximately 2″ × 3″. All the components except for the EPLD are easily available commercially. A complete list of all the components that are required for the PCB is given in Appendix B. The circuit can easily be connected and tested using the circuit diagram given below. After the four steps of the design are completed, the PCB can be used to throw a pair of dice in any home games such as Monopoly etc.

After the EPLD is programmed using the Logic Programmer, it can be inserted into the PCB. For design steps B, C, and D the push button switch can be used to generate the roll/no-roll or the spin/no spin option.

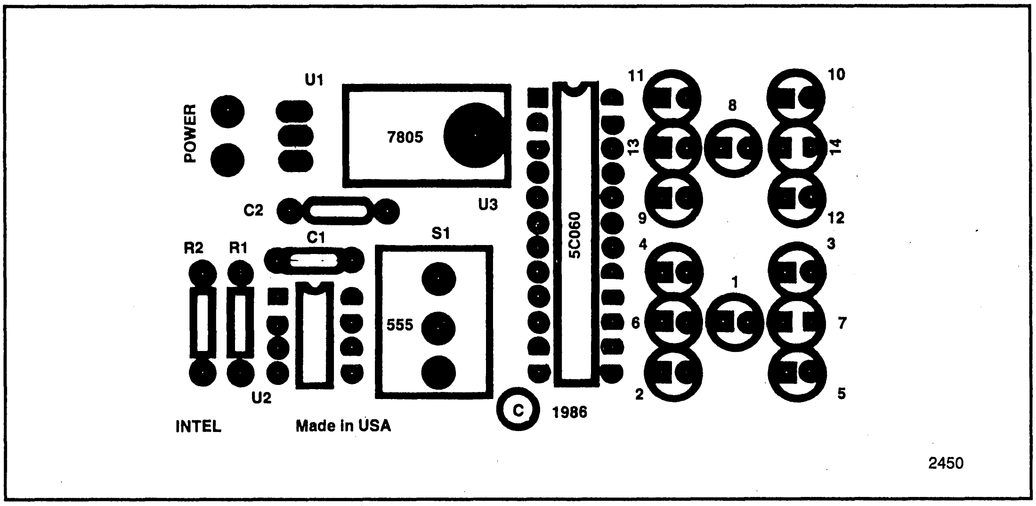

U1
POWER
7805
C2
U3
R2
R1
C1
S1
555
U2
5C060
INTEL
Made in USA
C 1986
11
10
8
13
14
9
12
4
3
1
6
7
2
5
2450

Figure C-2

APPENDIX D

```
ADF FOR PART A: SINGLE DICE ROLLING
------------------------------------

Lakshmi Jayanthi
DSO Applications
February 19, 1986

5C060

Part A:   DICE ROLLING

LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C060

INPUTS: clock1

OUTPUTS: dice1a@10,dice1b@9,dice1c@8,dice1d@7

NETWORK:

dice1a,1a = RORF (in1a,clock1,GND,GND,VCC)
dice1b,1b = RORF (in1b,clock1,GND,GND,VCC)
dice1c,1c = RORF (in1c,clock1,GND,GND,VCC)
dice1d,1d = RORF (in1d,clock1,GND,GND,VCC)

clock1 = INP (clock1)

EQUATIONS:

in1a =(/1a*1b*/1c*/1d)
         +(/1a*1b*1c*/1d)
         +(/1a*1b*1c*1d)
         +(/1a*/1b*/1c*/1d);
in1b =(1a*/1b*/1c*/1d)
         +(/1a*1b*/1c*/1d)
         +(1a*1b*/1c*/1d)
         +(/1a*1b*1c*/1d)
         +(1a*1b*1c*/1d);
in1c =(1a*1b*/1c*/1d)
         +(/1a*1b*1c*/1d)
         +(1a*1b*1c*/1d);
in1d =(1a*1b*1c*/1d);

END$
```

```
RPT FOR PART A: SINGLE DICE ROLLING
-----------------------------------------

Logic Optimizing Compiler Utilization Report

 ***** Design implemented successfully

Lakshmi Jayanthi
DSO Applications
February 19, 1986

5C060

Part A:    DICE ROLLING

LB Version 3.0, Baseline 17x, 9/26/85

                     5C060
                  - - - - -
    clock1  -| 1  .  24|- Vcc
       GND  -| 2     23|- GND
       GND  -| 3     22|- GND
       GND  -| 4     21|- GND
       GND  -| 5     20|- GND
       GND  -| 6     19|- GND
    dice1d  -| 7     18|- GND
    dice1c  -| 8     17|- GND
    dice1b  -| 9     16|- GND
    dice1a  -|10     15|- GND
       GND  -|11     14|- GND
       GND  -|12     13|- GND
                  - - - - -


**INPUTS**

                                                  |      Feeds:
        Name   Pin   Resource   MCell #    PTerms |   MCells    OE   Clear   Clock

      clock1    1        INP        -          -  |      -       -      -    CLK1

**OUTPUTS**

                                                  |      Feeds:
        Name   Pin   Resource   MCell #    PTerms |   MCells    OE   Clear   Clock

      dice1d    7       RORF       13       1/ 8           13     -      -       -
                                                           14
                                                           15
                                                           16

      dice1c    8       RORF       14       2/ 8           13     -      -       -
                                                           14
                                                           15
                                                           16

      dice1b    9       RORF       15       2/ 8           13     -      -       -
                                                           14
                                                           15
                                                           16

      dice1a   10       RORF       16       2/ 8           13     -      -       -
                                                           14
                                                           15
                                                           16
```

```
**UNUSED RESOURCES**

        Name   Pin   Resource      MCell     PTerms

         -      2        -           -          -
         -      3        -           9          8
         -      4        -          10          8
         -      5        -          11          8
         -      6        -          12          8
         -     11        -           -          -
         -     13        -           -          -
         -     14        -           -          -
         -     15        -           8          8
         -     16        -           7          8
         -     17        -           6          8
         -     18        -           5          8
         -     19        -           4          8
         -     20        -           3          8
         -     21        -           2          8
         -     22        -           1          8
         -     23        -           -          -

**PART UTILIZATION**

22%     Pins
25%     MacroCells
5%      Pterms
```

NOTE: Since part A is a simple design, the part utilization is very low.

```
ADF FOR PART B: SINGLE DICE ROLL/NOT ROLL
-----------------------------------------

Lakshmi Jayanthi
DSO Applications
February 19, 1986


5C060

PART B: DICE ROLL AND NOT ROLL

LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C060

INPUTS: clock1,switch@2

OUTPUTS: dice1a@10,dice1b@9,dice1c@8,dice1d@7

NETWORK:

dice1a,1a = RORF (in1a,clock1,GND,GND,VCC)
dice1b,1b = RORF (in1b,clock1,GND,GND,VCC)
dice1c,1c = RORF (in1c,clock1,GND,GND,VCC)
dice1d,1d = RORF (in1d,clock1,GND,GND,VCC)

clock1 = INP (clock1)

switch = INP(switch)

EQUATIONS:

in1a =(/1a*/1b*/1c*/1d*/switch)
        +(1a*/1b*/1c*/1d*/switch)
        +(1a*1b*/1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*/1b*/1c*/1d*switch)
        +(/1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(/1a*1b*1c*1d*switch);
in1b =(/1a*1b*/1c*/1d*/switch)
        +(1a*1b*/1c*/1d*/switch)
        +(/1a*1b*1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*1b*1c*1d*/switch)
        +(1a*/1b*/1c*/1d*switch)
        +(/1a*1b*/1c*/1d*switch)
        +(1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(1a*1b*1c*/1d*switch);
in1c =(/1a*1b*1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*1b*1c*1d*/switch)
        +(1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(1a*1b*1c*/1d*switch);
in1d =(/1a*1b*1c*1d*/switch)
        +(1a*1b*1c*/1d*switch);

END$
```

```
RPT FOR PART B: SINGLE DICE ROLL/NOT ROLL
------------------------------------------


Logic Optimizing Compiler Utilization Report

 ***** Design implemented successfully

Lakshmi Jayanthi
DSO Applications
February 19, 1986


5C060

PART B: DICE ROLL AND NOT ROLL

LB Version 3.0, Baseline 17x, 9/26/85

               5C060
            - - - - -
 clock1 -| 1    24|- Vcc
 switch -| 2    23|- GND
    GND -| 3    22|- GND
    GND -| 4    21|- GND
    GND -| 5    20|- GND
    GND -| 6    19|- GND
 dice1d -| 7    18|- GND
 dice1c -| 8    17|- GND
 dice1b -| 9    16|- GND
 dice1a -|10    15|- GND
    GND -|11    14|- GND
    GND -|12    13|- GND
            - - - - -

**INPUTS**

                                                 |            Feeds:
     Name   Pin   Resource    MCell #    PTerms  |   MCells      OE   Clear   Clock

   clock1    1        INP          -         -          -        -       -     CLK1

   switch    2        INP          -         -         13        -       -       -
                                                       14
                                                       15
                                                       16


**OUTPUTS**

                                                 |            Feeds:
     Name   Pin   Resource    MCell #    PTerms  |   MCells      OE   Clear   Clock

   dice1d    7       RORF         13      2/ 8         13        -       -       -
                                                       14
                                                       15
                                                       16

   dice1c    8       RORF         14      3/ 8         13        -       -       -
                                                       14
                                                       15
                                                       16
    dice1b     9        RORF          15       3/ 8           13      -       -
                                                              14
                                                              15
                                                              16

    dice1a     10       RORF          16       5/ 8           13      -       -
                                                              14
                                                              15
                                                              16


**UNUSED RESOURCES**

         Name   Pin   Resource      MCell     PTerms

          -      3       -            9          8
          -      4       -           10          8
          -      5       -           11          8
          -      6       -           12          8
          -     11       -            -          -
          -     13       -            -          -
          -     14       -            -          -
          -     15       -            8          8
          -     16       -            7          8
          -     17       -            6          8
          -     18       -            5          8
          -     19       -            4          8
          -     20       -            3          8
          -     21       -            2          8
          -     22       -            1          8
          -     23       -            -          -


**PART UTILIZATION**

27%       Pins
25%       MacroCells
10%       Pterms
```

NOTE: Part B of the design gets more complicated, hence the part utilization of the pins, macrocells and the Pterms is higher.

```
ADF FOR PART C: TWO DICE ROLLING
--------------------------------

Lakshmi Jayanthi
DSO Applications
February 19, 1986

5C060

PART C:  TWO DICE ROLL AND NOT ROLL

B Version 3.0, Baseline 17x, 9/26/85
PART: 5C060

INPUTS: clock1,clock2,switch@2

OUTPUTS: dice1a@10,dice1b@9,dice1c@8,dice1d@7,dice2a@19,dice2b@20,dice2c@21,dice
2d@22

NETWORK:

dice1a,1a = RORF (in1a,clock1,GND,GND,VCC)
dice1b,1b = RORF (in1b,clock1,GND,GND,VCC)
dice1c,1c = RORF (in1c,clock1,GND,GND,VCC)
dice1d,1d = RORF (in1d,clock1,GND,GND,VCC)

dice2a,2a = RORF (in2a,clock2,GND,GND,VCC)
dice2b,2b = RORF (in2b,clock2,GND,GND,VCC)
dice2c,2c = RORF (in2c,clock2,GND,GND,VCC)
dice2d,2d = RORF (in2d,clock2,GND,GND,VCC)

clock1 = INP (clock1)
clock2 = INP (clock2)

switch = INP (switch)

EQUATIONS:

in1a =(/1a*/1b*/1c*/1d*/switch)
        +(1a*/1b*/1c*/1d*/switch)
        +(1a*1b*/1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*/1b*/1c*/1d*switch)
        +(/1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(/1a*1b*1c*1d*switch);


in1b =(/1a*1b*/1c*/1d*/switch)
        +(1a*1b*/1c*/1d*/switch)
        +(/1a*1b*1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*1b*1c*1d*/switch)
        +(1a*/1b*/1c*/1d*switch)
        +(/1a*1b*/1c*/1d*switch)
        +(1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(1a*1b*1c*/1d*switch);
in1c =(/1a*1b*1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*1b*1c*1d*/switch)
        +(1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(1a*1b*1c*/1d*switch);
in1d =(/1a*1b*1c*1d*/switch)
        +(1a*1b*1c*/1d*switch);
```

```
in2a =(/2a*/2b*/2c*/2d*/(1d*switch))
        +(2a*/2b*/2c*/2d*/(1d*switch))
        +(2a*2b*/2c*/2d*/(1d*switch))
        +(2a*2b*2c*/2d*/(1d*switch))
        +(/2a*/2b*/2c*/2d*(1d*switch))
        +(/2a*2b*/2c*/2d*(1d*switch))
        +(/2a*2b*2c*/2d*(1d*switch))
        +(/2a*2b*2c*2d*(1d*switch));
in2b =(/2a*2b*/2c*/2d*/(1d*switch))
        +(2a*2b*/2c*/2d*/(1d*switch))
        +(/2a*2b*2c*/2d*/(1d*switch))
        +(2a*2b*2c*/2d*/(1d*switch))
        +(/2a*2b*2c*2d*/(1d*switch))
        +(2a*/2b*/2c*/2d*(1d*switch))
        +(/2a*2b*/2c*/2d*(1d*switch))
        +(2a*2b*/2c*/2d*(1d*switch))
        +(/2a*2b*2c*/2d*(1d*switch))
        +(2a*2b*2c*/2d*(1d*switch));
in2c =(/2a*2b*2c*/2d*/(1d*switch))
        +(2a*2b*2c*/2d*/(1d*switch))
        +(/2a*2b*2c*2d*/(1d*switch))
        +(2a*2b*/2c*/2d*(1d*switch))
        +(/2a*2b*2c*/2d*(1d*switch))
        +(2a*2b*2c*/2d*(1d*switch));
in2d =(/2a*2b*2c*2d*/(1d*switch))
        +(2a*2b*2c*/2d*(1d*switch));

END$
```

```
RPT FOR PART C: TWO DICE ROLLING
--------------------------------

Logic Optimizing Compiler Utilization Report

 ***** Design implemented successfully

Lakshmi Jayanthi
DSO Applications
February 19, 1986

5C060

PART C:  TWO DICE ROLL AND NOT ROLL

B Version 3.0, Baseline 17x, 9/26/85

                    5C060
                  - - - - -
     clock1  -| 1    24|- Vcc
     switch  -| 2    23|- GND
        GND  -| 3    22|- dice2d
        GND  -| 4    21|- dice2c
        GND  -| 5    20|- dice2b
        GND  -| 6    19|- dice2a
     dice1d  -| 7    18|- GND
     dice1c  -| 8    17|- GND
     dice1b  -| 9    16|- GND
     dice1a  -|10    15|- GND
        GND  -|11    14|- GND
        GND  -|12    13|- clock2
                  - - - - -

**INPUTS**

                                                   |        Feeds:
       Name   Pin   Resource    MCell #    PTerms  |   MCells      OE   Clear   Clock

     clock1    1        INP          -         -           -        -       -    CLK1

     switch    2        INP          -         -           1        -       -       -
                                                           2
                                                           3
                                                           4
                                                          13
                                                          14
                                                          15
                                                          16

     clock2   13        INP          -         -           -        -       -    CLK2


**OUTPUTS**

                                                   |        Feeds:
       Name   Pin   Resource    MCell #    PTerms  |   MCells      OE   Clear   Clock

     dice1d    7       RORF         13      2/ 8           1        -       -       -
                                                           2
                                                           3
                                                           4
                                                          13
                                                          14
                                                          15
                                                          16
```

```
dice1c    8      RORF       14      3/ 8        13     -      -      -
                                                14
                                                15
                                                16

dice1b    9      RORF       15      3/ 8        13     -      -      -
                                                14
                                                15
                                                16

dice1a    10     RORF       16      5/ 8        13     -      -      -
                                                14
                                                15
                                                16

dice2a    19     RORF        4      7/ 8         1     -      -      -
                                                 2
                                                 3
                                                 4

dice2b    20     RORF        3      4/ 8         1     -      -      -
                                                 2
                                                 3
                                                 4

dice2c    21     RORF        2      4/ 8         1     -      -      -
                                                 2
                                                 3
                                                 4

dice2d    22     RORF        1      3/ 8         1     -
                                                 2
                                                 3
                                                 4


**UNUSED RESOURCES**

      Name   Pin   Resource     MCell    PTerms

       -      3        -           9        8
       -      4        -          10        8
       -      5        -          11        8
       -      6        -          12        8
       -     11        -           -        -
       -     14        -           -        -
       -     15        -           8        8
       -     16        -           7        8
       -     17        -           6        8
       -     18        -           5        8
       -     23        -           -        -

**PART UTILIZATION**

50%     Pins
50%     MacroCells
24%     Pterms
```

NOTE: in part C of the design you have added the second dice. Hence you can see that fifty percent of the device has been used.

```
ADF FOR PART D: TWO DICE SPINNING
-----------------------------------

Lakshmi Jayanthi
DSO Applications
February 19, 1986


5C060

PART D: TWO DICE SPINNING

B Version 3.0, Baseline 17x, 9/26/85
PART: 5C060

INPUTS: clock1,clock2,switch@2

OUTPUTS: spin1a@10,spin1b@9,spin1c@8,spin1d@7,spin2a@19,spin2b@20,spin2c@21,spir
2d@22

NETWORK:

1a = NOJF (in1a,clock1,in11a,GND,GND)
1b = NOJF (in1b,clock1,in11b,GND,GND)
1c = NOJF (in1c,clock1,in11c,GND,GND)
1d = NOJF (in1d,clock1,in11d,GND,GND)

2a = NOJF (in2a,clock2,in22a,GND,GND)
2b = NOJF (in2b,clock2,in22b,GND,GND)
2c = NOJF (in2c,clock2,in22c,GND,GND)
2d = NOJF (in2d,clock2,in22d,GND,GND)

in11a = NOT(in1a)
in11b = NOT(in1b)
in11c = NOT(in1c)
in11d = NOT(in1d)

in22a = NOT(in2a)
in22b = NOT(in2b)
in22c = NOT(in2c)
in22d = NOT(in2d)

spin1a,s1a = RORF (ins1a,clock1,GND,GND,VCC)
spin1b,s1b = RORF (ins1b,clock1,GND,GND,VCC)
spin1c,s1c = RORF (ins1c,clock1,GND,GND,VCC)
spin1d,s1d = RORF (ins1d,clock1,GND,GND,VCC)

spin2a,s2a = RORF (ins2a,clock2,GND,GND,VCC)
spin2b,s2b = RORF (ins2b,clock2,GND,GND,VCC)
spin2c,s2c = RORF (ins2c,clock2,GND,GND,VCC)
spin2d,s2d = RORF (ins2d,clock2,GND,GND,VCC)

clock1 = INP (clock1)
clock2 = INP (clock2)

switch = INP (switch)

EQUATIONS:

in1a =(/1a*/1b*/1c*/1d*/switch)
        +(1a*/1b*/1c*/1d*/switch)
        +(1a*1b*/1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
```

```
          +(/1a*/1b*/1c*/1d*switch)
          +(/1a*1b*/1c*/1d*switch)
          +(/1a*1b*1c*/1d*switch)
          +(/1a*1b*1c*1d*switch);
in1b =(/1a*1b*/1c*/1d*/switch)
          +(1a*1b*/1c*/1d*/switch)
          +(/1a*1b*1c*/1d*/switch)
          +(1a*1b*1c*/1d*/switch)
          +(/1a*1b*1c*1d*/switch)
          +(1a*/1b*/1c*/1d*switch)
          +(/1a*1b*/1c*/1d*switch)
          +(1a*1b*/1c*/1d*switch)
          +(/1a*1b*1c*/1d*switch)
          +(1a*1b*1c*/1d*switch);
in1c =(/1a*1b*1c*/1d*/switch)
          +(1a*1b*1c*/1d*/switch)
          +(/1a*1b*1c*1d*/switch)
          +(1a*1b*/1c*/1d*switch)
          +(/1a*1b*1c*/1d*switch)
          +(1a*1b*1c*/1d*switch);
in1d =(/1a*1b*1c*1d*/switch)
          +(1a*1b*1c*/1d*switch);

in2a =(/2a*/2b*/2c*/2d*/(1d*switch))
          +(2a*/2b*/2c*/2d*/(1d*switch))
          +(2a*2b*/2c*/2d*/(1d*switch))
          +(2a*2b*2c*/2d*/(1d*switch))
          +(/2a*/2b*/2c*/2d*(1d*switch))
          +(/2a*2b*/2c*/2d*(1d*switch))
          +(/2a*2b*2c*/2d*(1d*switch))
          +(/2a*2b*2c*2d*(1d*switch));
in2b =(/2a*2b*/2c*/2d*/(1d*switch))
          +(2a*2b*/2c*/2d*/(1d*switch))
          +(/2a*2b*2c*/2d*/(1d*switch))
          +(2a*2b*2c*/2d*/(1d*switch))
          +(/2a*2b*2c*2d*/(1d*switch))
          +(2a*/2b*/2c*/2d*(1d*switch))
          +(/2a*2b*/2c*/2d*(1d*switch))
          +(2a*2b*/2c*/2d*(1d*switch))
          +(/2a*2b*2c*/2d*(1d*switch))
          +(2a*2b*2c*/2d*(1d*switch));
in2c =(/2a*2b*2c*/2d*/(1d*switch))
          +(2a*2b*2c*/2d*/(1d*switch))
          +(/2a*2b*2c*2d*/(1d*switch))
          +(2a*2b*/2c*/2d*(1d*switch))
          +(/2a*2b*2c*/2d*(1d*switch))
          +(2a*2b*2c*/2d*(1d*switch));
in2d =(/2a*2b*2c*2d*/(1d*switch))
          +(2a*2b*2c*/2d*(1d*switch));

ins1a = (/switch*1a);
ins1b = (/switch*1b)
          +((2d*switch)*s1d*/s1c*/s1b*/s1a);
ins1c = (/switch*1c)
          +((2d*switch)*/s1a*/s1b*/s1c*/s1d);
ins1d = (/switch*1d)
          +((2d*switch)*/s1a*/s1b*s1c*/s1d);

ins2a = (/switch*2a);
ins2b = (/switch*2b)
          +((2d*switch)*s2d*/s2c*/s2b*/s2a);
ins2c = (/switch*2c)
          +((2d*switch)*/s2a*/s2b*/s2c*/s2d);
ins2d = (/switch*2d)
          +((2d*switch)*/s2a*/s2b*s2c*/s2d);

END$
```

```
LEF FOR PART D: TWO DICE SPINNING
---------------------------------

Lakshmi Jayanthi
DSO Applications
February 19, 1986


5C060

PART:
          5C060

INPUTS:

          clock1, clock2, switch@2


OUTPUTS:

          spin1a@10, spin1b@9, spin1c@8, spin1d@7, spin2a@19, spin2b@20,
          spin2c@21, spin2d@22

NETWORK:

          clock1 = INP(clock1)
          clock2 = INP(clock2)

          switch = INP(switch)

          spin1a, s1a = RORF(ins1a, clock1, GND, GND, VCC)
          spin1b, s1b = RORF(ins1b, clock1, GND, GND, VCC)
          spin1c, s1c = RORF(ins1c, clock1, GND, GND, VCC)
          spin1d, s1d = RORF(ins1d, clock1, GND, GND, VCC)

          spin2a, s2a = RORF(ins2a, clock2, GND, GND, VCC)
          spin2b, s2b = RORF(ins2b, clock2, GND, GND, VCC)
          spin2c, s2c = RORF(ins2c, clock2, GND, GND, VCC)
          spin2d, s2d = RORF(ins2d, clock2, GND, GND, VCC)

          %  ***  Resource, NOJF, was minimized to NORF  ***  %

          2d = NORF(..SG007D, clock2, GND, GND)

          %  ***  Resource, NOJF, was minimized to NOTF  ***  %

          2c = NOTF(..SG006D, clock2, GND, GND)

          %  ***  Resource, NOJF, was minimized to NORF  ***  %

          2b = NORF(..SG005D, clock2, GND, GND)

          %  ***  Resource, NOJF, was minimized to NORF  ***  %

          2a = NORF(..SG004D, clock2, GND, GND)

          %  ***  Resource, NOJF, was minimized to NORF  ***  %

          1d = NORF(..SG003D, clock1, GND, GND)

          %  ***  Resource, NOJF, was minimized to NORF  ***  %

          1c = NORF(..SG002D, clock1, GND, GND)
```

```
% *** Resource, NOJF, was minimized to NORF *** %

1b = NORF(..SG001D, clock1, GND, GND)

% *** Resource, NOJF, was minimized to NORF *** %

1a = NORF(..SG000D, clock1, GND, GND)
```

EQUATIONS:

```
ins2d = switch' * 2d
      + 2d * switch * s2a' * s2b' * s2c * s2d';

ins2c = switch' * 2c
      + 2d * switch * s2a' * s2b' * s2c' * s2d';

ins2b = switch' * 2b
      + 2d * switch * s2d * s2c' * s2b' * s2a';

ins2a = switch' * 2a

ins1d = switch' * 1d
      + 2d * switch * s1a' * s1b' * s1c * s1d';

ins1c = switch' * 1c
      + 2d * switch * s1a' * s1b' * s1c' * s1d';

ins1b = switch' * 1b
      + 2d * switch * s1d * s1c' * s1b' * s1a';

ins1a = switch' * 1a;

..SG000D = 1a' * 1b' * 1c' * 1d'
         + 1a * 1c' * 1d' * switch'
         + 1a' * 1b * 1d' * switch
         + 1a * 1b * 1d' * switch'
         + 1a' * 1b * 1c * switch;

..SG001D = 1b * 1d'
         + 1b * 1a' * 1c * switch'
         + 1a * 1c' * 1d' * switch;

..SG002D = 1c * 1b * 1d'
         + 1c * 1a' * 1b * switch'
         + 1a * 1b * 1d' * switch;

..SG003D = 1d * 1a' * 1b * 1c * switch'
         + 1d' * 1a * 1b * 1c * switch;

..SG004D = 2a' * 2b' * 2c' * 2d'
         + 2a * 2c' * 2d' * 1d'
         + 2a * 2c' * 2d' * switch'
         + 2a * 2b * 2d' * 1d'
         + 2a * 2b * 2d' * switch'
         + 2a' * 2b * 2d' * 1d * switch
         + 2a' * 2b * 2c * 1d * switch;

..SG005D = 2b * 2d'
         + 2b * 2a' * 2c * 1d'
         + 2b * 2a' * 2c * switch'
         + 2a * 2c' * 2d' * 1d * switch;
```

```
    ..SG006D = 2c * 2b'
             + 2c * 2a * 2d
             + 2c * 2d * 1d * switch
             + 2c' * 2a * 2b * 2d' * 1d * switch;

    ..SG007D = 2d * 2a' * 2b * 2c * switch'
             + 2d * 2a' * 2b * 2c * 1d'
             + 2d' * 2a * 2b * 2c * 1d * switch;

END$
```

NOTE: PLease note how the IPLS software has simplified the equations for you. You need not worry about minimization. The complicated Boolean expressions have been minimized to a great extent.

```
RPT FOR PART D: TWO DICE SPINNING
--------------------------------------------

Logic Optimizing Compiler Utilization Report

 ***** Design implemented successfully

Lakshmi Jayanthi
DSO Applications
February 19, 1986


5C060


PART D: TWO DICE SPINNING

B Version 3.0, Baseline 17x, 9/26/85

                 5C060
               - - - - -
    clock1 -| 1    24|- Vcc
    switch -| 2    23|- GND
  RESERVED -| 3    22|- spin2d
  RESERVED -| 4    21|- spin2c
  RESERVED -| 5    20|- spin2b
  RESERVED -| 6    19|- spin2a
    spin1d -| 7    18|- RESERVED
    spin1c -| 8    17|- RESERVED
    spin1b -| 9    16|- RESERVED
    spin1a -|10    15|- RESERVED
       GND -|11    14|- GND
       GND -|12    13|- clock2
               - - - - -


**INPUTS**
                                                       |          Feeds:
    Name   Pin   Resource    MCell #      PTerms     |   MCells      OE   Clear   Clock

    clock1   1        INP          -           -              -        -       -   CLK1

    switch   2        INP          -           -              1        -       -      -
                                                                2
                                                                3
                                                                4
                                                                5
                                                                6
                                                                7
                                                                8
                                                                9
                                                               10
                                                               11
                                                               12
                                                               13
                                                               14
                                                               15
                                                               16

    clock2  13        INP          -           -              -        -       -   CLK2
```

```
**OUTPUTS**

                                                  |          Feeds:
  Name    Pin    Resource     MCell #     PTerms  |   MCells        OE    Clear   Clock

spin1d     7        RORF          13       2/ 8          13          -      -       -
                                                         14
                                                         15

spin1c     8        RORF          14       2/ 8          13          -      -       -
                                                         14
                                                         15

spin1b     9        RORF          15       2/ 8          13          -      -       -
                                                         14
                                                         15

spin1a    10        RORF          16       1/ 8          13          -      -       -
                                                         14
                                                         15

spin2a    19        RORF           4       1/ 8           1          -      -       -
                                                          2
                                                          3

spin2b    20        RORF           3       2/ 8           1          -      -       -
                                                          2
                                                          3

spin2c    21        RORF           2       2/ 8           1          -      -       -
                                                          2
                                                          3

spin2d    22        RORF           1       2/ 8           1          -      -       -
                                                          2
                                                          3


**BURIED REGISTERS**

                                                  |          Feeds:
  Name    Pin    Resource     MCell #     PTerms  |   MCells        OE    Clear   Clock


          18        NORF           5       3/ 8           1          -      -       -
                                                          5
                                                          6
                                                          7
                                                          8

          17        NORF           6       4/ 8           2          -      -       -
                                                          5
                                                          6
                                                          7
                                                          8

          16        NORF           7       4/ 8           3          -      -       -
                                                          5
                                                          6
                                                          7
                                                          8
          15        NORF        8       7/ 8         4       -       -       -
                                                     5
                                                     6
                                                     7
                                                     8

           3        NORF        9       2/ 8         5       -       -       -
                                                     6
                                                     7
                                                     8
                                                     9
                                                    10
                                                    11
                                                    12
                                                    13

           4        NORF       10       3/ 8         9       -       -       -
                                                    10
                                                    11
                                                    12
                                                    14

           5        NORF       11       3/ 8         9       -       -       -
                                                    10
                                                    11
                                                    12
                                                    15

           6        NORF       12       5/ 8         9       -
                                                    10
                                                    11
                                                    12
                                                    16

**UNUSED RESOURCES**

      Name   Pin   Resource      MCell     PTerms

        -     11       -           -          -
        -     14       -           -          -
        -     23       -           -          -

**PART UTILIZATION**

86%       Pins
100%      MacroCells
35%       Pterms
```

NOTE: Part D of the design example utilizes the device in a very optimum manner. You have utilized all the macrocells and also 86% of the pins but only 35% of the product terms.

You have not used three of the input pins.

Consider this:

Make these three pins a mode select on this dice example — if all of these three additional inputs are high then the dice will function as described (this condition must be added to each product term). You now have seven other modes in which to operate this DICE. Anyone want to "load" the odds for "boxcars" or "snake-eyes"? You have 65% more product terms to use so you can be very creative. What else could you add to this EPLD?