CAE TOOL FOR PROGRAMMABLE LOGIC SLASHES ENGINEERING TIME

Fusible links mean logic design variety

Dave B. Pellerin, Software Design Engineer, Data I/O Corp.
Michael J. Holley, Staff Design Engineer, Data I/O Corp.

Traditionally, engineers have designed digital logic circuits by interconnecting off-the-shelf, function logic ICs on a PC board. The advantages of these discrete logic elements are low cost and easy availability. Alternatively, the engineer can use a custom integrated circuit for the same design, if minimizing space and power consumption is required. Unfortunately, the large design effort and long lead times restrict the use of custom ICs to high-volume applications.

A third alternative, programmable logic, combines many of the benefits of custom ICs with the off-the-shelf availability of fixed function ICs. Furthermore, new design tools allow you to design and produce custom logic circuits with programmable logic devices in much less time than traditional methods require. A programmable logic development workstation composed of a personal computer, logic programmer, and logic development software can be purchased for less than $10,000.

Programmable logic devices consist of an array of logic gates interconnected by fusible links that can be programmed to implement a wide variety of logic designs. Generally, the device architecture, as shown in Fig. 1, consists of inputs fed into AND gates, which are then fed into OR gates to provide the desired outputs. Additionally, many devices provide features such as output registers, feedback, and exclusive-OR gates for implementing more complex logic designs.

Fig. 2 shows a simple logic function. While this function would require several 74LS00-type devices, it can be implemented in a portion of a single programmable logic device, such as a PAL 14L4. When designing for programmable logic, the first step is to write a Boolean equation that describes the logic function. The Boolean equation below describes the logic function shown in Fig. 2.

X=((A $ B) # !(C & D)) & (!E # F # G)

The Boolean operators !, &, #, and $ stand for NOT, AND, OR, and exclusive-OR, respectively. These operators are standard in high-level logic design languages.

Before this logic equation can be mapped onto the device, it must be converted into a sum-of-products form and reduced. In the past, designers had to perform tedious hand conversions to achieve this form. Now, with the aid of high-level languages, the logic function can be expressed in its natual form—with sum-of-products conversion and logic reduction performed automatically.

The ABEL design language

One such high-level lanugage, Data I/O's "ABEL", incorporates a language processor that converts the Boolean equation into the form required for the device. ⇒

ABEL is also available as part of complete development system that allows the engineer to design, program and test more than 130 logic devices.

ABEL language processor

The ABEL language processor converts logic descriptions to industry-standard programmer load files that can be downloaded to a logic programmer for device programming. The language processor checks your logic description, performs logic reduction, simulates the operation of the programmed device, and creates design documentation. A diagram of the ABEL processing flow is shown in Fig. B.

Here are the six steps in processing an ABEL source file:

STEP/ACTION

1. PARSE: Read the source file, check for correct syntax, expand macros, act on directives.
2. TRANSFOR: Transform original equations with set notation into normal Boolean equations using only basic Boolean operators.
3. REDUCE: Perform DeMorgan conversions and logic reduction.
4. FUSEMAP: Create the programmer load file.
5. SIMULATE: Simulate the function of a programmed device.
6. DOCUMENT: Create comprehensive design documentation. □

History of programmable logic devices

The first field-programmable logic device was the bipolar PROM (programmable read-only memory). Today, EPROMs (erasable PROMs) as large as 64 kbytes are available, although the much smaller 32-byte PROM is still a popular logic element. The first field-programmable logic array was introduced by Signetics in 1975. A few years later, Monolithic Memories introduced the "PAL" (programmable array logic) and the first logic design language, PALASM. Today there are a dozen companies producing or developing programmable logic devices. The newest logic devices are equivalent to more than 1000 AND gates. □

ABEL converts the exclusive-OR expression into an AND-OR expression, which is then transformed into the following active low, sum-of-products equation.

```
![X] = A & B & C & D
     # !A & !B & C & D
     # E & !F & !G
```

Next, ABEL maps the equation onto the programmable logic device, as shown in Fig. 3. The fuses marked with an X are kept intact when the device is programmed, and the remaining fuses are blown. The programmed device will then perform the logic function resulting from this fuse pattern.

Design example: A microprocessor address decoder

Here is how ABEL can be used in a typical microprocessor-based controller. The system uses a MC6809 microprocessor, a 32-kbyte EPROM, three 8-kbyte static RAMs, an AD7528 digital-to-analog converter (DAC), an 8-channel analog to digital converter (ADC) and an SCN2651 serial interface. All decoding and control logic is contained in a single programmable logic device, a PAL20L10.

The PAL 20L10 monitors the microprocessor address bus and, based on the value of these address bits, selects the proper memory segment by placing a low (logical zero) on its appropriate output pin.

Using ABEL, the engineer can express this design clearly in Boolean equations, with the help of high-level language constructs such as set notation and relational operators. Fig. 6 shows a complete ABEL source file implementing this design.

In the first section of the source file, a module name and descriptive title are followed by declarations of the device to be used, and input and output pin names. Only the pins used for this design need to be declared. A 20L10 PAL device was selected for this design, although one of many other common devices would suffice. (Substituting a different device is a simple matter of changing the device name in the declaration and, if necessary, changing the pin numbers.)

Constants are also declared in this section. Constant declarations are a powerful way to make a design more meaningful to read and easier to write; frequently used expressions involving single constants, sets, and even entire logical expressions can be given symbolic names that are used in the remainder of the source file. These symbolic names simplify complex expressions.

The address decoder design is sim-

Although ABEL version shown here runs on personal computer, version that runs on DEC VAX is also available.

ABEL converts state diagrams into reduced Boolean equations for programming into device.

plified by grouping the nine address line inputs and seven "don't care" values into a set of 16 members named Address. A "don't care", indicated by the special constant ".X.", is a way of specifying that a signal can be either high or low. In this way, the true memory addresses may be used in the equations. The ABEL relational operators allow the designer to use a "natural" form for expressing the logic function, rather than restricting him to standard Boolean operators. For each output pin, the designer writes one equation that expresses the logical conditions which will produce the desired function for that output.

Simulation

The source file also includes a test vector section. Test vectors are an important part of any programmable logic design, and are used by the ABEL simlulator to verify that the design will produce the intended outputs for a given set of inputs.

Address Range	Device Selected
0000 - 1FFF	RAM1
2000 - 3FFF	RAM2
4000 - 5FFF	RAM3
6000 - 600F	Digital to Analog
6010 - 601F	Analog to Digital
6020 - 602F	Serial Interface
8000 - FFFF	EPROM

Fig. 5 - Memory map of microprocessor system.

The ability to simulate the function of the design with test vectors can result in significant savings of time and money, because the designer can be sure that the device will perform as expected before programming it. The same test vectors may also be used by the programming hardware so that the device functions properly after it is programmed. When writing the test vectors, the engineer can again use set notation, resulting in a clearly readable source file and reducing chances that incorrect test vectors will be written. In Fig. 6, three separate test vector sections are written for testing the memory and I/O decoding, and control signal functions.

ABEL processing

When the source file is processed by ABEL, set equations are expanded and the relational operators are converted to equivalent equations using standard Boolean operators. Logic reduction is then performed to minimize the number of product terms used to perform the desired logic function. Automatic logic reduction allows the designer to

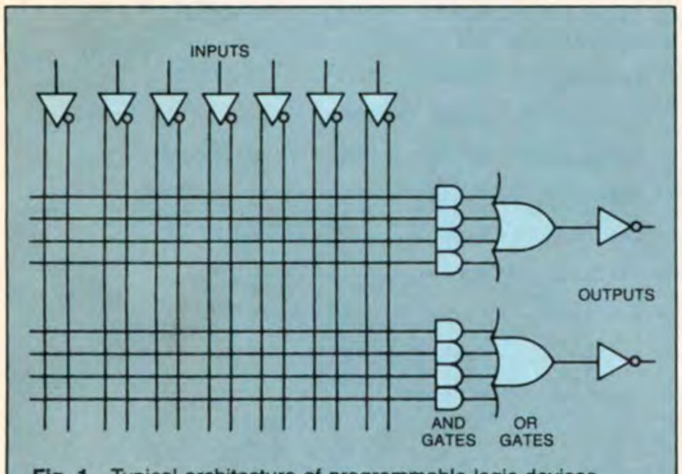

Fig. 1 - Typical architecture of programmable logic devices.

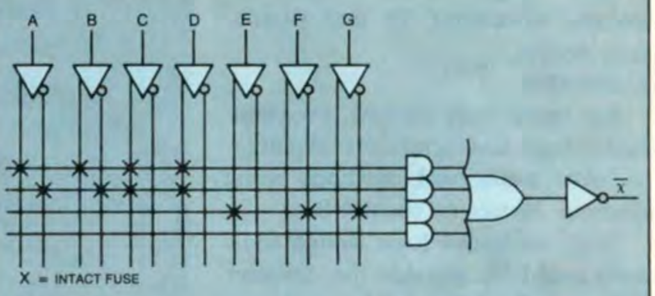

Fig. 3 - Logic function mapped onto programmable logic array. Fuse locations marked with 'X' are intact.

!X = A & B & C & D
!A & !B & C & D
# E & !F & !G

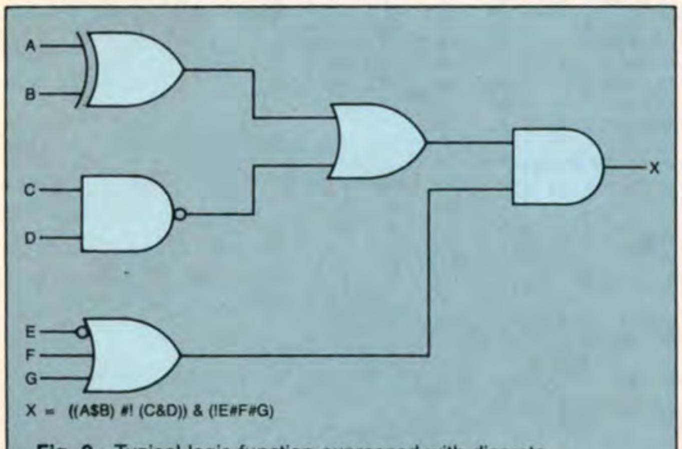

Fig. 2 - Typical logic function expressed with discrete logic elements.

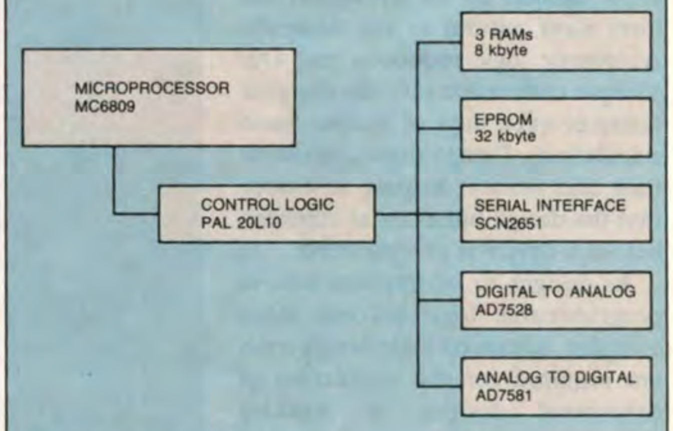

Fig. 4 - Block diagram of microprocessor-based controller.

express his design in a clear, straightforward manner—as opposed to the restrictive sum-of-products form required by the logic device. Fig. 7 shows the reduced equations as printed by the ABEL document generator. Also shown are the expanded test vectors produced by ABEL from the vectors in the source file. The expanded test vectors contain expected values for all input and output pins.

ABEL maps the reduced equations onto the fuse array of the specified device and simulates the design with the use of the test vectors provided. Simulation errors, if any, are reported by a listing of which vectors failed and what results were actually obtained on the outputs.

The final output includes an industry-standard format (JEDEC VStandard No. 3) download file for device programming, as well as a comprehensive document file and simulation results.

Conclusion

For many logic designs, programmable logic has significant advantages over traditional methods using discrete devices or custom ICs.

New, advanced logic design tools such as ABEL provide the designer with new flexibility in the way logic designs are expressed. Designs may be written as truth tables, state diagrams, or Boolean equations. The powerful set notation and a variety of logical and relational operators allow designs to be written in the form most natural to the designer. Automatic logic reduction and DeMorgan conversion save the designer hours or even days of tedious hand calculations. Design simulation saves time and money, helping to ensure that the design functions as expected before a device is programmed.

As designs to be implemented in programmable logic become more complex, advanced logic design tools are required for the conversion of conceptual designs to working devices. □

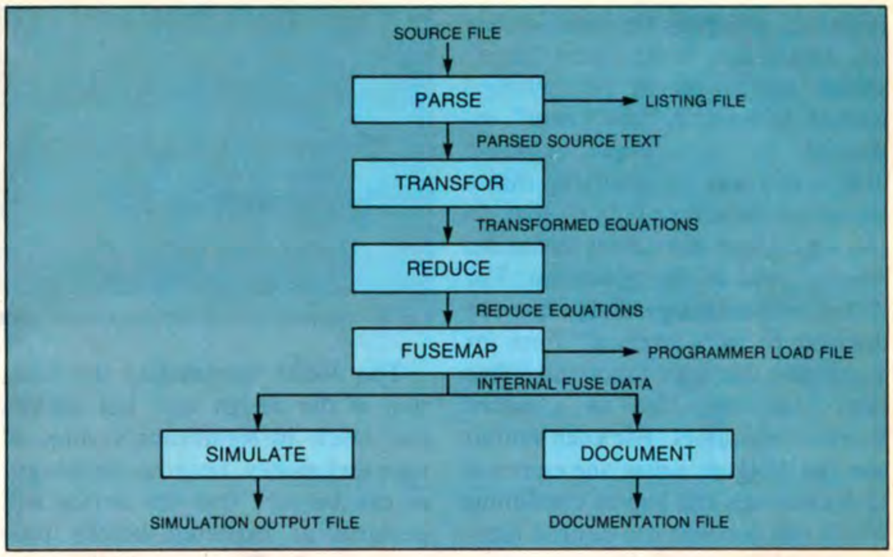

Fig. 8 - ABEL processing flow.

```
                 U1     device    'P20L10';
         A15,A14,A13,A12,A11,A10 pin 1,2,3,4,5,6;
         A6,A5,A4                pin 7,8,9;
         E,RW                    pin 10,11;
         RAM1,RAM2,RAM3,EPROM    pin 14,15,16,17;
         SERIAL,DAC,ADC          pin 18,19,20;
         SRW,WR                  pin 21,22;

     H,L,X     = 1,0,.X.;
     Address   = [A15,A14,A13,A12, A11,A10,X,X, X,A6,A5,A4, X,X,X,X];

 equations
         !RAM1   = (Address <= ^h1FFF);

         !RAM2   = (Address >= ^h2000) & (Address <= ^h3FFF);

         !RAM3   = (Address >= ^h4000) & (Address <= ^h5FFF);

         !DAC    = (Address >= ^h6000) & (Address <= ^h600F);

         !ADC    = (Address >= ^h6010) & (Address <= ^h601F);

         !SERIAL = (Address >= ^h6020) & (Address <= ^h602F);

         !EPROM  = (Address >= ^h8000);

         !WR     = E & !RW;      "write strobe for AD7528 DAC"

         !SRW    = RW;           "inverted RW for SCN2651 serial port"

 test_vectors 'test RAM and EPROM decode'
             ([Address] -> [RAM1,RAM2,RAM3,EPROM])
              [^h0000 ] -> [ L,   H,   H,    H  ];
              [^h1000 ] -> [ L,   H,   H,    H  ];
              [^h2000 ] -> [ H,   L,   H,    H  ];
              [^h3000 ] -> [ H,   L,   H,    H  ];
              [^h4000 ] -> [ H,   H,   L,    H  ];
              [^h5000 ] -> [ H,   H,   L,    H  ];
              [^h6000 ] -> [ H,   H,   H,    H  ];
              [^h8000 ] -> [ H,   H,   H,    L  ];
              [^hA000 ] -> [ H,   H,   H,    L  ];
              [^hC000 ] -> [ H,   H,   H,    L  ];
              [^hFFFF ] -> [ H,   H,   H,    L  ];

 test_vectors 'test I/O decode'
             ([Address] -> [DAC,ADC,SERIAL])
              [^h6000 ] -> [ L,  H,   H   ];
              [^h6008 ] -> [ L,  H,   H   ];
              [^h6010 ] -> [ H,  L,   H   ];
              [^h6020 ] -> [ H,  H,   L   ];
              [^h6030 ] -> [ H,  H,   H   ];

 test_vectors 'test control signals'
             ([E,RW] -> [WR,SRW])
              [0, 0] -> [ H, H ];
              [0, 1] -> [ H, L ];
              [1, 0] -> [ L, H ];
              [1, 1] -> [ H, L ];
 end System
```

Fig. 6 - Complete ABEL source file for address decoder and control logic design.

BISON GIVES YOU LESS FOR THE MONEY!

About 30% less in overall space yet it delivers 43% more power! The reason? Bison gearmotors are engineered smarter out of stronger materials! We include better bearings, oil filled boxes, and hardened steel gears. They're designed to AGMA standards to deliver more torque in less space.

And in OEM quantities, they do it more economically. So that chances are good you can replace your present gearmotors with built-to-last Bisons and save money. Because the other thing less about Bison gearmotors are their prices!

We make your products go.

BISON
Gear & Engineering Corp.
2424 Wisconsin Avenue
Downers Grove, IL 60515
312-968-6400
Circle No. 421

Did you find this article interesting? If so, let us know. Circle No. 855

Equations for Module System

```
Device U1
     Reduced Equations:

        RAM1 = !(!A13 & !A14 & !A15);

        RAM2 = !(A13 & !A14 & !A15);

        RAM3 = !(!A13 & A14 & !A15);

        DAC = !(!A10 & !A11 & !A12 & A13 & A14 & !A15 & !A4 & !A5 &
              !A6);

        ADC = !(!A10 & !A11 & !A12 & A13 & A14 & !A15 & A4 & !A5 &
              !A6);

        SERIAL = !(!A10 & !A11 & !A12 & A13 & A14 & !A15 & !A4 & A5
                 & !A6);

        EPROM = !(A15);

        WR = !(E & !RW);

        SRW = !(RW);
```

Chip diagram for Module System

```
Device U1
                             P20L10

                    ----------\       /----------
                    |          \     /          |
                    |           -----           |
              A15   |  1                    24  |  Vcc
                    |                           |
              A14   |  2                    23  |
                    |                           |
              A13   |  3                    22  |  WR
                    |                           |
              A12   |  4                    21  |  SRW
                    |                           |
              A11   |  5                    20  |  ADC
                    |                           |
              A10   |  6                    19  |  DAC
                    |                           |
               A6   |  7                    18  |  SERIAL
                    |                           |
               A5   |  8                    17  |  EPROM
                    |                           |
               A4   |  9                    16  |  RAM3
                    |                           |
                E   | 10                    15  |  RAM2
                    |                           |
               RW   | 11                    14  |  RAM1
                    |                           |
              GND   | 12                    13  |
                    |                           |
                    |                           |
                    -----------------------------
```

Test Vectors for Module System

```
Device U1
test RAM and EPROM decode

   1 [0000 0000 0--- ---- ---- ----] -> [---- ---- ---- -LHH H--- ----];
   2 [0001 0000 0--- ---- ---- ----] -> [---- ---- ---- -LHH H--- ----];
   3 [0010 0000 0--- ---- ---- ----] -> [---- ---- ---- -HLH H--- ----];
   4 [0011 0000 0--- ---- ---- ----] -> [---- ---- ---- -HLH H--- ----];
   5 [0100 0000 0--- ---- ---- ----] -> [---- ---- ---- -HHL H--- ----];
   6 [0101 0000 0--- ---- ---- ----] -> [---- ---- ---- -HHL H--- ----];
   7 [0110 0000 0--- ---- ---- ----] -> [---- ---- ---- -HHH H--- ----];
   8 [1000 0000 0--- ---- ---- ----] -> [---- ---- ---- -HHH L--- ----];
   9 [1010 0000 0--- ---- ---- ----] -> [---- ---- ---- -HHH L--- ----];
  10 [1100 0000 0--- ---- ---- ----] -> [---- ---- ---- -HHH L--- ----];
  11 [1111 1111 1--- ---- ---- ----] -> [---- ---- ---- -HHH L--- ----];

test I/O decode

  12 [0110 0000 0--- ---- ---- ----] -> [---- ---- ---- ---- -HLH ----];
  13 [0110 0000 0--- ---- ---- ----] -> [---- ---- ---- ---- -HLH ----];
  14 [0110 0000 1--- ---- ---- ----] -> [---- ---- ---- ---- -HHL ----];
  15 [0110 0001 0--- ---- ---- ----] -> [---- ---- ---- ---- -LHH ----];
  16 [0110 0001 1--- ---- ---- ----] -> [---- ---- ---- ---- -HHH ----];

test control signals

  17 [---- ---- -00- ---- ---- ----] -> [---- ---- ---- ---- ---- HH--];
  18 [---- ---- -01- ---- ---- ----] -> [---- ---- ---- ---- ---- LH--];
  19 [---- ---- -10- ---- ---- ----] -> [---- ---- ---- ---- ---- HL--];
  20 [---- ---- -11- ---- ---- ----] -> [---- ---- ---- ---- ---- LH--];
end of module System
```

Fig. 7 - ABEL documentation including reduced equations, chip diagram and test vectors.