Interoffice Correspondence

☐ For Your Comments	☐ For Your Information
☐ For Your Approval	☐ Note & Return
☐ Please Take Action	☐ Note & File
☐ Call Me	☐ For Your Signature
☐ See Me	☐ Please Advise Status
☐ Per Our Conversation	☐ Per Your Request
☐ Read, Initial & Return	☐ Please Advise Appropriate Employees In Your Department

To: Vic Belmondo
George James
Wayne Paulson

Date: December 9, 1981

CC:

From: Russ dePina

Subject: Generalized Logic Development Language

Enclosed is a preliminary specification of the Data I/O Proprietary Logic Development Language. Feel free to make any comments or questions.

RMdP:lc
Attachment

SPECIFICATION FOR AN ADVANCED BOOLEAN EXPRESSION LANGUAGE (ABEL)

Prepared by Russell dePina
December 8, 1981

SPECIFICATION FOR AN ADVANCED BOOLEAN EXPRESSION LANGUAGE (ABEL)

1. APPLICATIONS

ABEL is being developed to allow users of programmable logic devices to have a universal language with which they may specify the logic structure for the devices regardless of the manufacturer. The ABEL processor will be resident within the Data I/O Logic Pak so that, once the logic structure is defined, the fuse pattern may immediately be programmed into the device.

2. MEMORY REQUIREMENTS

The ABEL processor will reside in the PROM space of the Logic Pak and shall occupy no more than 8K of memory. ABEL will also have the capability to use floppy disk storage when the Logic Pak is tied to a DCU.

3. DEVELOPMENT LANGUAGES

The language that will be used for the development of the ABEL processor should be a high level language capable of performing string operations, matrix manipulation as well as logic functions. The development language should also support I/O operations to and from peripherals such as floppy disk. The following are languages that could be considered for the development of ABEL:

PASCAL

C

FORTH

PL/I and derivatives

FORTH is an attractive possibility because it is an interpretive language and is very efficient in its machine code production. However, there may be problems installing FORTH on the PDP-11 and VAX. PASCAL, C, and PL/I are block-structured languages which have good support and can be implemented with almost no problems. One drawback to most of these languages is that there does not exist a cross-compiler for the 6800/6809 that runs on the PDP-11. The reasons why this is important are:

A. The object code from the cross-compiler will run on the target computer (6800/6809), thus eliminating the need for extra compilers in the development database.

B. Development time is decreased because of fewer errors in transporting to the development system. Also, the larger host computer will run the compiler faster than the smaller machine and a more powerful instruction set of the development language can be implemented.

4. OPERATION OF THE ABEL PROCESSOR

The operation of the ABEL processor should be kept as simple as possible. The typical ABEL program will include module specification, device specification, data definition sections, and a program section where the logic equations are defined. An end of module specification designates the end of the device module specification.

5. DATA DEFINITION SECTION

5.1 Program Specification

An ABEL module specification is an alphanumeric string used to identify a specific ABEL module. This feature is used especially when more than one module occupies a program unit. This feature allows for multiple device definition in the event a user wishes to define an entire network of logic devices. Up to 3 distinct modules can be specified in a single program unit. This option is selected when the ABEL processor is invoked.

The syntax for the program unit specification is as follows:

```
PROGRAM    <name>
```

$\texttt{<Name>}\ ::=\ \texttt{<letter><alphanumeric>}^{0:5}$

The PROGRAM specification is used only when the user wishes to define more than one device. When only one device is specified, a module specification is different. (Note: because of the extra memory required to store multiple fuse patterns, the PROGRAM feature can be implemented only on a system with some form of mass storage (floppy disk).

5.2 Module Specification

The ABEL module is where the device logic specification is made. An ABEL module consists of data definition sections and a logic definition section (boolean equations) from which the ABEL processor will create a device fuse pattern. The syntax of the module specification is shown below:

```
MODULE     <module name>
```

$\texttt{<Module name>}\ ::=\ \texttt{<letter><alphanumeric>}^{0:5}$

5.3 Device Specification

The Device Specification statement tells the ABEL processor how to set up the universal fuse matrix to conform to a particular device. It also allows the use of certain ABEL instructions, depending on the device. The Device Specification syntax is shown below:

```
DEVICE      <device spec>

<device spec>  ::=<mmi spec> <sig spec> <amd spec> <harris spec>

                  <national spec> <ti spec>

<mmi spec>      ::=  MMI20 <mmi series 20 number>

                     MMI24 <mmi series 24 number>

<mmi series 20 part number>  ::= 10H8.|.12H6.|.14H4.|.16H2

                                    |.16L2.|.16L8.|.16R8.|.16R4.

                                    |.16X4.|.16A4

<mmi series 24 number> ::=    12L10 | 14L8 | 16L6

                             |18L4   | 20L2 | 20C1

                             |20L10  | 20X10|20X8

                             |20X4

<sig spec>  ::=    <iFl-20 part number> <iFl-28 part number>

<ifl-20 part no> ::=    82S150 | 82S151 | 82S152 | 82S153 | 82S154

                        82S155 | 82S156 | 82S157 | 82S158 | 82S159

<ifl-28 part no> ::=    82S100 | 82S101 | 82S102 | 82S103 | 82S104

                        82S105 | 82S106 | 82S107

<amd spec>  ::=    <amd designator><pal part no.>

<amd designator>::= AMD

<pal part no> ::=  <mmi series 20 part no.>

                    mmi series 24 part no.>
```

5.4 Pin List Specification

The pin list is used to give symbolic representation to the pins of a particular device. These pin names are then equated by the ABEL processor to certain points in the master device array and are used in the definition of the device logic structure. Pin list specification syntax is shown below.

<pin list> ::= <PIN-LIST:><line skip>[<space><pin identifier>^{20}

<line skip> ::= <carriage return><line feed><

<pin identifier>::= <letter><alphanumeric>$^{1:3}$

5.5 Register Specification

Register specifications are made to define which device outputs are the outputs of a sequential logic structure. Only one device so far (82S104/5) has an internal set of registers. The 20 pin FPLS (82S154-9) has registered outputs which can be dynamically programmed to be Jk, D, or T type flip flops. The ABEL processor will require the user to specify one type of flip flop and will not allow dynamically programmable register types. The default flip flop type is D. In the case of the 82S104/5, a register specification <u>must</u> be made since the register type for that device is RS. The register names are declared in the pin list, and are typed as registered outputs in the register specification.

The syntax of a register specification is shown below:

<register specification> ::=

<register list><delimiter><register type>

<register list> ::= {register name<>delimiter>$^{0:1}$}$^{1:8}$

<register name> ::= <letter><alphanumeric>$^{0:3}$

<register type> ::= JK| RS| D| T

<delimiter> ::= ; | : | | , | .

6. EQUATION DEFINITION SECTION

The Boolean Equations processed by the ABEL processor fall into two categories: assignment equations and conditional equations.

Assignment equations usually will cause a device output or register to change its state. Conditional statements will change the device outputs if a specific condition is met.

The syntax for assignment equations follows:

<assignment stmt> ::= <expression>

<expression> ::= <expression><operator><expression> | <identifier>

<identifier> ::= <letter><alphanumeric>$^{0:3}$

<operator> ::= * | + | / | :+: | =

operator list

*	AND
+	OR (inclusive)
:+:	exclusive OR
/	NOT

The operator precedence structure is as follows:

\

*

+, :+:

=

The syntax for conditional statements is as follows:

<conditional stmt> ::= IF <condition> THEN <assignment stmt>

<condition> ::= (<expression>)