DDL-P COMMAND LANGUAGE MANUAL

by

W.E. Cory
J.R. Duley
W.M. vanCleemput

Technical Report No. 164

March 1979

This work was supported by the Joint Services Electronics Program
under Contract N-00014-85-C-0601.

DDL-P COMMAND LANGUAGE MANUAL

W.E. Cory, J.R. Duley, W.M. vanCleemput

Technical Report No. 164

March 1979

Computer Systems Laboratory
Stanford University
Stanford, California 94305

ABSTRACT

This report describes the command language for the simulator, associated with DDL (Digital Design Language).

INDEX TERMS: Design automation, computer-aided design, hardware description languages, DDL, digital design language.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

This report is one of two manuals describing a compiler and simulator for DDL-P, a subset of DDL (Digital Design Language). DDL is a language for describing the behavior of digital systems at the Boolean equation, register transfer, and algorithmic levels. It uses a finite state machine notation and it may be used to describe systems over a wide range of levels.

DDL was originally formulated by Duley at the University of Wisconsin in 1967 [5, 6, 3]. A translator and simulator for a subset of DDL were implemented in FORTRAN [1, 7, 4, 8, 9, 10]. In 1971-73, J. Duley, B. Clark, and J. Welsch implemented an interactive simulation system for a subset of DDL (with modified syntax) on the HP 2100 system in HP-Algol at Hewlett-Packard Laboratories. The DDL-P language, compiler, and simulator are based on this HP implementation. In order to enhance portability the system was rewritten in PASCAL on the DEC-20 system under the TOPS-20 Operating System at Stanford University. Small changes were made to the

syntax, mainly to enhance the readability. The system will still accept the input format of the original HP-Algol version.

This report describes the usage of the DDL-P simulator as it was implemented at Stanford. The command language is described in detail and several examples are given to clarify the constructs. The appendices give a list of error messages as well as a formal BNF definition of the command language. A companion report describes the DDL-P language and the DDL-P compiler system [2].

Chapter 2

BOOLEAN EXPRESSIONS

```
boolean-exp ::= minterm { + minterm }***

minterm ::= product { [+] product }***

product ::= complement { * complement }***

complement ::= {-} relation

relation ::= arithmetic-exp
             | arithmetic-exp (=) arithmetic-exp
             | arithmetic-exp  #  arithmetic-exp
             | arithmetic-exp  <  arithmetic-exp
             | arithmetic-exp  >  arithmetic-exp
             | arithmetic-exp  >=  arithmetic-exp
             | arithmetic-exp  <=  arithmetic-exp
             | AT  identifier

arithmetic-exp  ::= { (-) } term
                  I arithmetic-exp (+) term
                  | arithmetic-exp (-) term

term ::= constant
       | reference
       | TIME
       | (  boolean-exp  )
```

In the command language, Boolean expressions may appear in SET, DISPLAY, PRINT, and conditional commands. The syntax for comments, constants, identifiers, and facility re-

ferences in the command language is identical to that in DDL-P descriptions. This syntax is shown in Appendix B for reference.

The syntax for Boolean expressions in the command language is similar to the syntax for Boolean expressions in a DDL-P description. The following operators and constructs are NOT available in the command language:

```
INPUT function
Conditional expression constructs
   CASE x DO a DO b . . ENDCASE
   IF x THEN a ELSE b ENDIF
   ¢x¢ a; b; . . .
CON
+ RED      * RED      [+] RED   (+) RED
EXT        TAIL       HEAD
```

The remaining operators are available as shown in the BNF above. In addition, two new constructs are permitted:

1. TIME

   is a string of length 16 bits with value equal to the number of units of time the DDL-P system has been simulated. TIME is initially zero and is incremented at the end of each state in the lowest level finite state machine. Recall that the size of this increment may be specified in the DDL-P description by the TIME action.

2. AT state-name

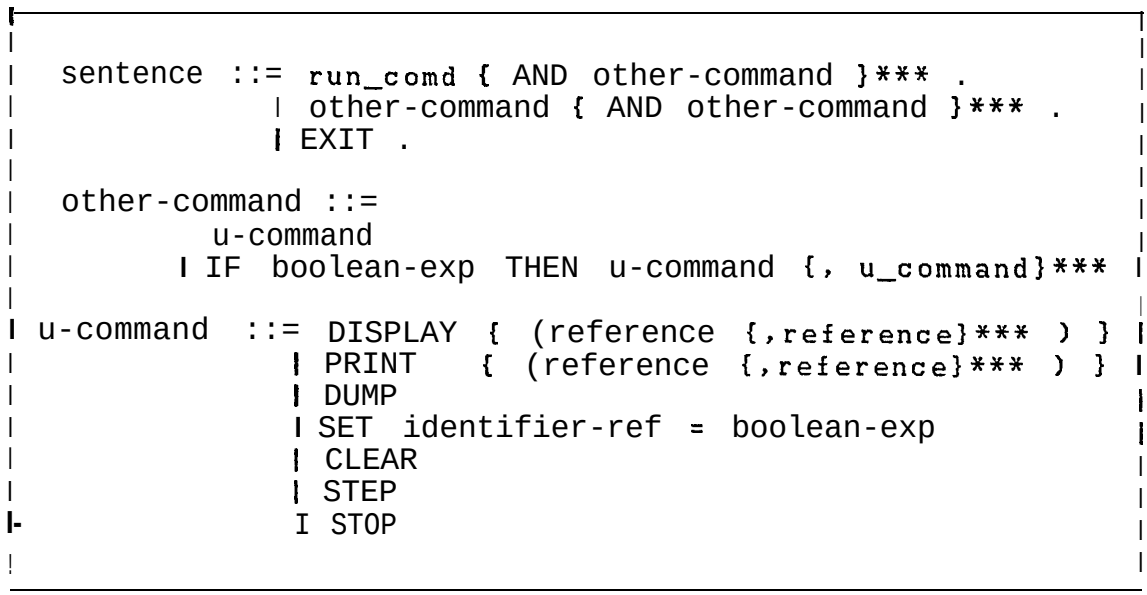
is a string of length one bit with value 1B1 if the state given by "state-name" is the current state, and value 1B0 otherwise.

Examples of Boolean Expressions in the Command Language

```
16D17
(A*B)(=)7 + X>Y + PROD(X,2)>25
MEM[IR[9:0](+)R[IR[12:10]]]
-AT P
M[MAR](=)HALT + AT ERR + TIME>100
```

Chapter 3

COMMAND SENTENCES



After a DDL-P description has been compiled with no fatal errors and the designer has selected the number base for use in output, then the simulator indicates its readiness to accept commands by typing ">" at the teletype. The designer may then use command sentences to examine or modify facility contents and simulate the DDL-P system.

The individual commands are discussed first, followed by a description of the constructs for combining several commands into a single command sentence.

3.1 COMMANDS

3.1.1 DISPLAY and PRINT

The DISPLAY command types the current values of the named facilities on the teletype, along with the current total state (location) and time. Any valid register, memory, or terminal reference may appear in the DISPLAY list. This list may be omitted altogether, in which case the current total state and time are typed.

Examples

```
>DISPLAY(LEFT,RIGHT).
*TIME=1110 STATE=P:              LEFT=0000000000000010
     RIGHT=0000000000000101
>DISPLAY(SCORE[2:4], CARD[LEFT:RIGHT], SUM(2,2)).
*TIME=1110 STATE=P:              SCORE[10:100]=110
     CARD[10:101]=1001           SUM=0000000000000100
>DISPLAY.
*TIME=1110 STATE=P:
>
```

The PRINT command is identical to the DISPLAY command, except that the output goes to the listing file on disk.

3.1.2 DUMP

The DUMP command prints in the listing file the current values of all registers, memories, and terminals which weren't assigned functions in the TERMINAL declaration. A DUMP is equivalent to a PRINT with all these facilities in the output list.

3.1.3 SET

The designer may use the SET command to set the values of registers, memories, and terminals which weren't assigned functions in the TERMINAL declaration. The SET command works like the "=" immediate store action in an OPERATION; when SET is executed in a command sentence, the facility referenced on the left side is set immediately to the current value of the Boolean expression on the right side.

Examples

```
SET CARDBUF=5D6
SET SCORE[1:2] = A[3:4] + SCORE[1:2]
```

If the facility written is a register for which a delayed store has not been finished, then the simulator will issue a SIMULTANEOUS STORE warning and cancel the delayed store.

3.1.4 CLEAR

The CLEAR command resets all registers, memories, and terminals to zero. This action occurs immediately and unconditionally whenever CLEAR appears anywhere in a command sentence, even if the CLEAR is in a conditional command. HENCE, WHEN USED, THIS COMMAND SHOULD APPEAR BY ITSELF IN THE COMMAND SENTENCE, even though this restriction is not enforced by the simulator.

All unfinished delayed stores will be canceled when a CLEAR is executed.

3.1.5 RUN

```
run_comd ::= RUN { FROM identifier {:identifier}*** }
                 {   TO identifier {:identifier}*** }
```

The RUN command initiates simulation of the DDL-P system. The designer may specify the initial and/or final locations of the simulation in the command. A location is a list of states separated by colons ":", where the list contains zero or one state from each finite state machine in

the DDL-P description. State sequencing registers will be initialized automatically before the simulation begins.

The RUN command has four forms, as follow:

1. RUN FROM location

   The simulation starts in the indicated location. If some finite state machine has no state listed in the location given, then the initial state for that machine will be the first state in the corresponding CONTROL declaration. The simulation will begin in the top level machine.

2. RUN

   Simulation starts in the current state, proceeding from the point where simulation was last suspended. If there was no previous simulation, then the initial location is the first state of each finite state machine.

   The exception is that if this command follows a RUN FROM command which did not execute because of an error in the command sentence, then the initial location will be the location given in the previous RUN FROM.

3. RUN TO location

   Simulation begins in the current state as above (the exception noted above also applies here), and proceeds until the system is in the indicated location. The state actions for this last location will be executed before the simulation ends.

   If the initial state is the same as the final state, then no simulation will take place.

4. RUN FROM location1 to location2

   Simulation begins in location1 and proceeds until the system is in location2.

During a simulation, the command sentence initiating the simulation will be rescanned and executed at the end of each state in the lowest level finite state machine. Hence, commands for conditionally halting simulation or for setting, displaying, or printing facility values during the simulation may be included in the command sentence with the RUN command. When the simulation is finished, the simulator will type ">" at the teletype, indicating its readiness to accept a new command sentence.

Examples

```
RUN
RUN FROM P1:Z1
RUN TO JK
RUN FROM A TO JK
```

After a run-time error, the initial location in a RUN command must be given explicitly by "FROM location**. The simulator variable TIME is reset to zero when a "RUN FROM.." command is issued.

When issuing a "RUN FROM" command, the designer should be careful to ensure that the system is properly initialized. In particular, beware that previously scheduled delayed stores may be carried out before the simulation is restarted (these may be canceled by CLEAR or SET).

3.1.6 STEP

The STEP command effects simulation of the system for one state. If the system contains several finite state machines, then one state in the lowest level machine is simulated; states from higher level machines may be simulated first, depending on the starting point of the simulation and/or the presence of LEVEL actions in previously simulated states.

The STEP command may appear in a sentence with a RUN command, in which case the RUN command determines the starting location of the simulation. Otherwise, simulation starts from the current location.

3.1.7 STOP

The STOP command halts the simulation in progress. This command typically appears in a conditional command (Section 3.21, so that the simulation will end when the command sentence is rescanned and some condition is satisfied.

A STOP only ends the simulation; it does not prevent execution of the remainder of the command sentence. The STOP, when executed, overrides a RUN or STEP command in the same command sentence.

3.1.8 Halt by <ESCAPE>

During a simulation, the designer may press the <ESCAPE> key to cause the simulator to halt and accept a new command sentence. Note that the <ESCAPE> is not a command one includes in a command sentence; rather, it is a way of interrupting a simulation, forcing a halt.

After an <ESCAPE>, the simulator may halt in any of three ways:

1. Normally, the simulator will halt at the end of the current state in the lowest level finite state machine. The halt will occur just after the last command sentence has been rescanned in the normal way.

2. In case of a run-time error, the simulator will halt immediately after issuing a run-time warning or error message if the designer has pressed <ESCAPE>.

3. If the simulator has executed more than 1000 goto actions in operations in the current state, then it will issue a warning that the DDL-P system is "probably in infinite loop." After this point, the simulator halts immediately when <ESCAPE> is pressed, rather than waiting until the end of the state.

3.1.9 EXIT

The EXIT command causes an exit from the simulator, returning control to the TOPS-20 Executive. When used, EXIT must be the only command in the command sentence. Note that STOP and EXIT are entirely different commands.

3.2 CONDITIONAL COMMAND

The conditional command allows the designer to make the execution of a command list dependent on some condition. The form of the command is

IF Boolean-exp THEN command, command,...

Each time the conditional command is scanned, the commands in the list will be executed only if the value of the Boolean expression is 1; otherwise the command list will be skipped.

Any combination of DISPLAY, PRINT, DUMP, SET, STEP, or STOP commands may appear in the conditional command list. Conditional commands may not be nested.

Examples

```
IF AT END + AT ERR THEN DUMP,STOP
IF TIME>=20 THEN PRINT(SCORE)
```

3.3 SENTENCE

A command sentence is a list of commands separated by "AND". All sentences are terminated by period ".", including those with just one command. Any combination of commands may appear in a sentence, subject to the following restrictions:

1. A sentence may include at most one RUN command, which (if present) must be the first command in the sentence.

2. When used, EXIT must be the only command in the sentence.

3. When used, CLEAR should be the only command in the sentence.

Command sentences may take up more than one line on the teletype; the simulator will prompt for additional lines by typing "=" until the designer terminates the sentence with period ".". If the designer enters end-of-file (CTRL-Z) from the teletype, the simulator will type a message and exit to the TOPS-20 Executive.

Examples

```
RUN AND STEP AND DUMP.
RUN FROM A AND PRINT(SCORE,CARDBUF,FF).
RUN AND IF AT JK THEN STOP,DUMP AND DISPLAY.
SET CARDBUF=5D1 AND SET FF=1B1.
STEP.
CLEAR.
EXIT   "return to TOPS-20".
```

In the third example above, note that STOP and DUMP are dependent on the condition AT JK, while DISPLAY is unconditional.

If a command sentence contains syntax errors, an error message will be typed and the entire sentence must be reentered. The simulator executes an error-free sentence immediately; if the sentence contains a RUN command, then the sentence will be executed again after each lowest-level state, as discussed in Section 3.1.5.

Note that during a simulation, the simulator is controlled only by the last command sentence entered. The sequence

```
>IF AT P + AT S + AT J + AT T THEN STOP.
>RUN FROM L AND DISPLAY.
```

will not have the desired effect of stopping when the simulation reaches state P, S, J, or T, because the simulator is controlled only by the second sentence. Instead, the designer could type

```
>RUN FROM L TO P AND DISPLAY
=  AND IF AT S + AT J + AT T THEN STOP.
```

Two sample simulations are shown in Chapter 5.

Chapter 4

SIMULATION-TIME ERRORS

As the designer carries out the simulation, errors may occur in four distinct areas:

1. A command sentence may have a syntax error.

2. An error condition may arise during execution of a command sentence.

3. A number entered during simulation of an INPUT action may have a syntax error.

4. An error condition may arise during simulation of the DDL-P system,

When an error occurs, the simulator types an error message at the teletype. The error message will also appear in the listing file unless the problem was a syntax error in a command sentence. In most cases, if the error severity is FATAL or ABORT, the simulator will suspend activity immediately and prompt for a new command sentence with ">".

Errors have one other important effect on simulator activity. Recall that the simulator must save delayed-store values in temporary buffers until the store is carried out. The simulator must also remember which terminals are to be cleared at the end of a state. The disposition of this information in case of error depends on the type of error and the error severity.

Error messages issued by the simulator are listed in Appendix A. The four types of errors are now discussed in more detail.

4.1 ERROR IN COMMAND SENTENCE SYNTAX

Syntax errors of severity FATAL or ABORT cause the entire sentence (except for any CLEAR) to be ignored; the entire sentence must be reentered. Delayed-store values will never be lost as a result of such errors; however, all terminals will be cleared and delayed stores completed if an invalid sentence contains a RUN FROM command.

Syntax errors of severity WARNING do not terminate command processing.

4.2 ERROR DURING COMMAND EXECUTION

Errors may arise in the evaluation of Boolean Expressions in DISPLAY, PRINT, SET, or conditional commands. In such cases, the simulator will stop immediately after issuing an error message if the severity is FATAL or ABORT or if the designer has pressed <ESCAPE>.

When the simulator stops after such an error, all delayed stores to registers are canceled. If the error severity is ABORT, then all terminals are cleared before the simulator will accept a new command; otherwise, all terminals will be cleared just before simulation is resumed (following a RUN command).

4.3 ERROR DURING SIMULATION-TIME INPUT

If a number entered during an INPUT action has a syntax error of severity WARNING or FATAL, then the simulator will type an error message and ask the designer to reenter the number. Such errors will not otherwise halt or interrupt simulation.

If an error of severity ABORT appears, then the simulator will halt simulation, cancel all delayed stores, clear all terminals, and prompt for a new command sentence.

4.4 ERROR DURING SIMULATION

When an error occurs during simulation, the simulator will print an error message followed by a value for SIMADDR, an internal location counter. The designer may then look at the program listing generated by the DDL-P compiler to determine the exact location of the error; the listing includes values of SIMADDR at the left hand margin.

The simulator will stop immediately after a simulation-time error if the severity is FATAL or ABORT or if the designer has pressed <ESCAPE>. After such a stop, all delayed stores are canceled. If the error severity is ABORT, then all terminals are cleared before the simulator will accept a new command. Otherwise, all terminals will be cleared just before simulation is resumed (following a RUN command).

Chapter 5

SAMPLE SIMULATION

Sample simulation sessions are now presented for two examples from An Introduction to the DDL-P Lansuase [2]. The examples are reprinted here for reference.

In the first example, the designer displays all the registers in each state, eventually halting the simulation by <ESCAPE>. After single-stepping twice from state D, the designer then restarts the simulation, displaying results only at the end.

```
" B L A C K J A C K     M A C H   I   N E ."
REGISTER  SCORE[5], CARDBUF[5], FF.
TERMINAL  HIT, BROKE, STAND,
   VALUE[1:5] = INPUT(1,VALUE),
   YCRD = INPUT(1,YCRD),
   YL17 = SCORE<17, YL22 = SCORE<22,
   NACE = CARDBUF#1.
OPERATION
   TPT = [CARDBUF <- 5D10],
   TMT = [CARDBUF <- 5D22],
   TVC = [CARDBUF <- VALUE],
   IHIT = [HIT=1B1],
   ISTD = [STAND=1B1], IBRK  = [BROKE=1B1],
   CLS = [SCORE <- 5D0],
   ADD=[SCORE <-(SCORE(+)CARDBUF)TAIL 51,
   KFF = [FF<-1D0], JFF  = [FF <- 1D1 1
CONTROL
   A:  CLS, KFF, ->B/
   B:  IHIT, TVC,
       IF YCRD THEN ->C ELSE ->B ENDIF/
   C:  IF YCRD THEN ->C ELSE ->D ENDIF/
   D:  ADD,  IF NACE+FF THEN ->F
                        ELSE ->E ENDIF/
   E:  JFF, TPT, ->D/
   F:  IF YL17 THEN ->B ELSE ->G ENDIF/
   G:  IF YL22 THEN ->K ELSE ->H ENDIF/
   H:  KFF, TMT,
       IF FF THEN ->D ELSE ->J ENDIF/
   J:  IBRK,
       IF YCRD THEN ->A ELSE ->J ENDIF/
   K:  ISTD,
       IF YCRD THEN ->A ELSE ->K ENDIF/.$

**********************************************************

@<sources.ddl>ddl
INPUT      = black.jak
OUTPUT     = black.lst
DDLINI     = ddl.ini[4,1550]

TO CONTINUE,  HIT THE RETURN KEY *

 END OF TRANSLATION,    0 FATAL ERROR(S).

 DDL SIMULATION MONITOR.  VERSION 16 NOVEMBER 1978

PLEASE ENTER RADIX (FOR USE IN ALL OUTPUT)
(BASE 2,  4,  8,  10,  OR 16): 10
```

```
>run and display(score,cardbuf,f f).
*TIME=0 STATE=A:          SCORE=0          CARDBUF=0
     FF=0
*TIME=0 STATE=A:          SCORE=0          CARDBUF=0
     FF=0
*TIME=1 STATE=B:
VALUE:=5d12
*TIME=1 STATE=B:
YCRD:=lbO
*TIME=1 STATE=B:          SCORE=0          CARDBUF=0
     FF=0
*TIME=2 STATE=B:
VALUE:=5d1
*TIME=2 STATE=B:
YCRD:=1b1
*TIME=2 STATE=B:          SCORE=0          CARDBUF=12
     FF=0
*TIME=3 STATE=C:
YCRD:=1b1
*TIME=3 STATE=C:          SCORE=0          CARDBUF=l
     FF=0
*TIME=4 STATE=C:
YCRD:=lbO
*TIME=4 STATE=C:          SCORE=0          CARDBUF=1
     FF=0
*TIME=5 STATE=D:          SCORE=0          CARDBUF=1
     FF=0
*TIME=6 STATE=E:          SCORE=1          CARDBUF=l
     FF=0
*TIME=7 STATE=D:          SCORE=1          CARDBUF=10
     FF=1
*TIME=8 STATE=F:          SCORE=11         CARDBUF=10
     FF=1
*TIME=9 STATE=B:
VALUE:=5d5
*TIME=9 STATE=B:
YCRD:=lbO
*TIME=9 STATE=B:          SCORE=11         CARDBUF=10
     FF=1
*TIME=10 STATE=B:
VALUE:=5d5
*TIME=10 STATE=B:
YCRD:=1b1
*TIME=10 STATE=B:         SCORE=11         CARDBUF=5
     FF=1
*TIME=11 STATE=C:
YCRD:=1b1
*TIME=11 STATE=C:         SCORE=11         CARDBUF=5
```

```
     FF=1
*TIME=12  STATE=C:
YCRD:=lbO
*TIME=12  STATE=C:          SCORE=11          CARDBUF=5
     FF=1
*TIME=13  STATE=D:          SCORE=11          CARDBUF=5
     FF=1
*TIME=14  STATE=F:          SCORE=16          CARDBUF=5
     FF=1
*TIME=15  STATE=B:
VALUE:=5d10
*TIME=15  STATE=B:
YCRD:=1b1
*TIME=15  STATE=B:          SCORE=16          CARDBUF=5
     FF=1
*TIME=16  STATE=C:
YCRD:=lbO
*TIME=16  STATE=C:          SCORE=16          CARDBUF=10
     FF=1
*TIME=17  STATE=D:          SCORE=16          CARDBUF=10
     FF=1
*TIME=18  STATE=F:          SCORE=26          CARDBUF=10
     FF=1
*TIME=19  STATE=G:          SCORE=26          CARDBUF=lO
     FF=1
*TIME=20  STATE=H:          SCORE=26          CARDBUF=10
     FF=1
*TIME=21  STATE=D:          SCORE=26          CARDBUF=22
     FF=0
*TIME=22  STATE=F:          SCORE=16          CARDBUF=22
     FF=0
*TIME=23  STATE=B:
VALUE:=5d6
*TIME=23  STATE=B:
YCRD:=1b1
*TIME=23  STATE=B:          SCORE=16          CARDBUF=22
     FF=0
*TIME=24  STATE=C:
YCRD:=lbO
*TIME=24  STATE=C:          SCORE=16          CARDBUF=6
     FF=0
*TIME=25  STATE=D:          SCORE=16          CARDBUF=6
     FF=0
*TIME=26  STATE=F:          SCORE=22          CARDBUF=6
     FF=0
*TIME=27  STATE=G:          SCORE=22          CARDBUF=6
     FF=0
*TIME=28  STATE=H:          SCORE=22          CARDBUF=6
```

```
     FF=0
*TIME=29 STATE=J:
YCRD:=lbO
*TIME=29 STATE=J:          SCORE=22          CARDBUF=22
     FF=0
STOP BY <ESCAPE>
*TIME=29 STATE=J:
>display(stand,
=         broke>.
*TIME=29 STATE=J:          STAND=0           BROKE= 1
>set cardbuf=5d1 and set ff=1b1.
>run from d and step.
*TIME=0 STATE=D:
>step.
*TIME=1 STATE=F:
>run from a and if at j+at k then display(stand,broke),stop.
*TIME=1 STATE=B:
VALUE:=5d7
*TIME=1 STATE=B:
YCRD:=1b1
*TIME=2 STATE=C:
YCRD:=lbO
*TIME=5 STATE=B:
VALUE:=5d4
*TIME=5 STATE=B:
YCRD:=1b1
*TIME=6 STATE=C:
YCRD:=lbO
*TIME=9 STATE=B:
VALUE:=5d1
*TIME=9 STATE=B:
YCRD:=1b1
*TIME=10 STATE=C:
YCRD:=lbO
*TIME=19 STATE=B:
VALUE:=5d8
*TIME=19 STATE=B:
YCRD:=1b1
*TIME=20 STATE=C:
YCRD:=lbO
*TIME=24 STATE=K:
YCRD:=lbO
*TIME=24 STATE=K:          STAND= 1          BROKE=0
*TIME=24 STATE=K:
>display(cardbuf,score).
*TIME=24 STATE=K:          CARDBUF=8         SCORE=20
```

```
>exit.

EXIT
a

*************************************************************
```

In the second example, the designer runs the memory summation machine. The accumulating sum is displayed only when a non-zero word has been added. Note that base 16 is used for output.

```
" EXAMPLE OF INTERPRETIVELY LINKED MACHINE "
REGISTER   A[16], B[ 161, ADDR[8].
MEMORY     MEM[0:255, 161.
TERMINAL   YCLR, YINC, YADD, YREAD,
           NDONE = ADDR # 255,
           START = INPUT (1, START).
OPERATION  CLEAR = [A <- 16D0, ADDR <- 8D0],
           INC = [ADDR <- ADDR(+)8D1 TAIL 81,
           ADD = [A <- A(+)B  TAIL  161,
           READ = [B <- MEM[ADDR],
                    MEM[ADDR] = 16D0],
           RESTORE = [MEM[ADDR] = B].
CONTROL
   P1: IF START THEN YCLR @, YREAD @, ->P2
                ELSE ->P1 ENDIF/
   P2: YADD @,
       IF NDONE THEN YINC @, YREAD @, ->P2
                ELSE ->P1 ENDIF/
CONTROL
   Q1: IF YINC THEN INC ENDIF,
       IF YCLR THEN CLEAR ENDIF,
       IF YADD THEN ADD ENDIF,
       IF YREAD THEN ->Q2
                ELSE ->Q1, LEVEL ENDIF/
   Q2: READ, ->Q3/
   Q3: RESTORE, LEVEL, ->Q1/.$

*************************************************************
```

```
@<sources.ddl>ddl
INPUT      = sum.mem
OUTPUT     = sum.lst
DDLINI     = ddl.ini[4,1550]

TO CONTINUE,  HIT THE RETURN KEY *

 END OF TRANSLATION,    0 FATAL ERROR(S).

 DDL SIMULATION MONITOR.   VERSION 16 NOVEMBER 1978

PLEASE ENTER RADIX (FOR USE IN ALL OUTPUT)
(BASE 2, 4, 8, 10, OR 16): 16
>set mem[10]=10 and set mem[20]=20.
>set mem[30]=30 and set mem[40]=40.
>set mem[50]=50 and set mem[60]=60.
>set mem[70]=70 and set mem[80]=80.
>set mem[90]=90 and set mem[99]=99.
>set addr=8d0.
>run and if at q2 * b#0 then display (addr,b,a)
=    and if at p1 * time>100 then display (a),stop.
*TIME=0 STATE=P1:Q1:
START:=1b0
*TIME=1 STATE=P1:Q1:
START:=lbl
*TIME=23 STATE=P2:Q2:      ADDR=0B              B=000A
     A=000A
*TIME=41 STATE=P2:Q2:      ADDR=15              B=0014
     A=001E
*TIME=5F STATE=P2:Q2:      ADDR=1F              B=001E
     A=003C
*TIME=7D STATE=P2:Q2:      ADDR=29              B=0028
     A=0064
*TIME=9B STATE=P2:Q2:      ADDR=33              B=0032
     A=0096
*TIME=B9 STATE=P2:Q2:      ADDR=3D              B=003C
     A=00D2
*TIME=D7 STATE=P2:Q2:      ADDR=47              B=0046
     A=0118
*TIME=F5 STATE=P2:Q2:      ADDR=51              B=0050
     A=0168
*TIME=113 STATE=P2:Q2:     ADDR=5B              B=005A
     A=01C2
*TIME=12E STATE=P2:Q2:     ADDR=64              B=0063
     A=0225
*TIME=302 STATE=P1:Q1:
START:=1b0
*TIME=302 STATE=P1:Q1:     A=0225
```

```
*TIME=302 STATE=Pl:Ql:
>exit.

EXIT
@
```

Appendix A

ERROR MESSAGES

The error messages issued by the DDL-P simulator are listed below, along with their severity codes. When an error condition arises, the simulator lists the appropriate message at the teletype and, in some cases, in the listing file. The significance of the error severity codes is explained in Chapter 4.

The characters in the first three columns of each line below indicate when the error on that line may occur. A "C" in column one denotes that the error may occur as the simulator is examining a command sentence. An "I" in column two denotes that the error may occur when the user types an input constant during simulation. An "S" in column three indicates that the error may occur while a command is being executed or during simulation.

The term "predefined terminal" refers to a terminal for which a function was specified in the TERMINAL declarations. An "argument" is the same as an "actual parameter," and

"SSR" stands for "state sequencing register." The string "<ID>" will be replaced by the appropriate facility name when an error message is printed.

The error

STRING TOO BIG FOR DECIMAL DISPLAY

refers to the fact that DDL-P cannot print a decimal number longer than 99 digits. The error message

SYNTAX ERROR

flags many kinds of errors in command sentences. In case of such an error, the designer should refer to the DDL-P Command Sentence BNF to determine the proper syntax.

C..	fatal	Syntax error
CI.	warning	Illegal character
CI.	warning	Input line longer than 132 characters
CI.	fatal	Constant too large
.I.	fatal	Binary string must start with decimal digit
.I.	fatal	Illegal character in binary string
CI.	fatal	Illegal number length spec. (zero or >256)
CI.	fatal	Decimal number may not be left-justified
CI.	fatal	Illegal char. or digit of wrong radix in no.
CI.	fatal	Digit is of improper radix
C..	fatal	Undeclared identifier
C..	fatal	This identifier may not be subscripted
C..	fatal	Two-dimensional array requires subscript
C..	fatal	This identifier may only have 1 subscript
C..	fatal	Field can't be used to denote range of words
C..	fatal	Subscripting nested too deeply (>10 levels)
C..	fatal	Improper field or access to non-existent bits
C..	fatal	Too many dimensions (>2) or invalid field
C..	fatal	Predef ined terminal subscripted
C..	fatal	Missing argument list
C..	fatal	Wrong number of arguments
C..	fatal	This identifier may not have arguments
C..	fatal	This identifier not allowed in expression
C..	fatal	Operation identifier not allowed in expr.

C..	fatal	Assignment to identifier of wrong type
C..	fatal	Identifier must be a state
C..	fatal	"RUN FROM. . " required after error
C..	fatal	"EXIT" must appear in command by itself
..S	fatal	Operand too long (>256 bits)
..S	fatal	String or CON or EXT result is too long
..S	fatal	Head or tail length too long
..S	fatal	String or field range is too big (>256 bits)
..S	warning	String too big for decimal display
..S	fatal	Reference to non-existent word of <ID>
..S	fatal	Store into non-existent word of <ID>
..S	fatal	Reference to non-existent bit of <ID>
..S	fatal	Store into non-existent bit of <ID>
..S	fatal	Improper field or non-existent bits of <ID>
..S	warning	Incompatible lengths for store into <ID>
..S	warning	Incompatible lengths for operation
..S	warning	Simultaneous stores into a flip-flop of <ID>
..S	warning	Illegal store into lower level SSR
..S	fatal	Two next states specified
..S	fatal	Two "=>" states specified
..S	fatal	No state corresponds to this SSR value
..S	fatal	No next-state indicated
..S	fatal	No place to return
..S	warning	***Probably in infinite loop
..S	abort	Internal error: Illegal instruction
..S	abort	Internal error: stack overflow
C..	fatal	Unexpected end of input
CI.	fatal	Unexpected end of file
C..	fatal	Unexpected end of command
C..	fatal	Internal error: parse stack overflow
CIS	abort	Internal error: memory overflow

Appendix B

DDL-P COMMAND SENTENCE BNF

The complete Backus-Naur Form for DDL-P simulator command sentences is listed below. Non-terminals are written in lower-case letters and underscore; "ddl_description" is a non-terminal, e.g. All other symbols are terminals except for the following special symbols ("meta-symbols"):

::= REPLACEMENT SYMBOL - Left-hand side may be replaced by right-hand side.

{ } OPTIONAL STRING SYMBOL - String of symbols enclosed in braces is optional.

{ }*** REPETITION SYMBOL - String of symbols enclosed may appear zero or more times in succession.

|| CONCATENATION SYMBOL - Symbol on left must be concatenated with symbol on right (i.e., with no intervening blanks or end-of-line).

| OR SYMBOL - This separates several right-hand sides of productions.

The simulator prompts for the first line of a command sentence by typing ">" at the beginning of a line. A sentence may take up several lines; the simulator prompts for additional lines with "=". In a sentence, a comment is any string of symbols except double-quote (") enclosed in double-quotes and contained on one line, e.g.,

"THIS IS A COMMENT" .

A comment may appear anywhere a blank is permitted.

```
letter : := A|B|C|D|E|F|G|H|I|J|K|L|M|
           N|O|P|Q|R|S|T|U|V|W|X|Y|Z|
           a|b|c|d|e|f|g|h|i|j|k|l|m|
           n|o|p|q|r|s|t|u|v|w|x|y|z

digit  ::= 0|1|2|3|4|5|6|7|8|9

hex-digit      := digit | A | B | C | D | E | F

. octal-digit  := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

quartal_digit ::= 0 | 1 | 2 | 3

bit            := 0 | 1

decimal-constant : : = digit { || digit }***

constant ::= decimal-constant
       | decimal-constant | | B { || . } | | bit
                                { | | bit }***
       I  decimal-constant  || Q { || . }
                            || quartal_digit
                          {  || quartal_digit }***
       | decimal-constant  | | @ { II . }
                            || octal-digit
                          {  || octal-digit }***
       | decimal-constant  || D I I decimal-constant
       | decimal-constant  | | H { || . }
                            || hex-digit
                          {  || hex-digit }***
```

```
letter-or-digit ::= letter | digit

identifier ::= letter { || letter-or-digit }***

field ::= boolean-exp   boolean-exp

identifier-ref ::= identifier
        | identifier || decimal-constant
        | identifier [ boolean-exp ]
        | identifier || decimal-constant
                                    [ boolean-exp ]
        | identifier [ boolean-exp ][ boolean-exp ]
        | identifier [ boolean-exp , boolean-exp ]
        | identifier [ field ]
        | identifier || decimal-constant [ field ]
        | identifier [ boolean-exp ][ field ]
        | identifier [ boolean-exp , field ]

terminal-ref ::= identifier ( boolean-exp
                          {, boolean_exp}*** )

reference ::= identifier-ref | terminal-ref

boolean-exp ::= minterm { + minterm }***

minterm ::= product { [+] product }***

product ::= complement { * complement }***

complement ::= {-} relation

relation ::= arithmetic-exp
           | arithmetic-exp (=) arithmetic-exp
           | arithmetic-exp #  arithmetic-exp
           | arithmetic-exp <  arithmetic-exp
           | arithmetic-exp >  arithmetic-exp
           | arithmetic-exp >= arithmetic-exp
           | arithmetic-exp <= arithmetic-exp
           | AT identifier

arithmetic-exp ::= { (-) } term
                 | arithmetic-exp (+) term
                 | arithmetic-exp (-) term

term ::= constant
       | reference
       | TIME
       | ( boolean-exp )
```

```
sentence ::= run-comd { AND other-command }*** .
           | other-command { AND other-command }*** .
           | EXIT .

run-comd ::= RUN { FROM identifier {:identifier}*** }
                 {   TO identifier {:identifier}*** }

other-command ::=
        u-command
      | IF boolean-exp THEN u-command {, u-command}***

u-command ::= DISPLAY { (reference {,reference}*** ) }
            | PRINT   { (reference {,reference}*** ) }
            | DUMP
            | SET identifier-ref = boolean-exp
            | CLEAR
            | STEP
            | STOP
```

REFERENCES

1. Arndt, R.L. and Dietmeyer, D.L. "DDLSIM--A Digital Design Language Simulator," Proc. National Electronics Conf., Vol. 26, pp. 116-118, December 1970.

2. Cory, W.E., Duley, J.R., and vanCleemput, W.M. An Introduction to the DDL-P Lansuase. Palo Alto: Stanford University, Computer Systems Laboratory, March 1979, 97 pp.

3. Dietmeyer, D.L. and Duley, J.R. "Register Transfer Languages and Their Translation" in Disital System Desisn Automation: Lanquages, Simulation and Data Base. Woodland Hills, CA.: Computer Science Press, Inc., 1975, pp. 117-218.

4. Dietmeyer, D.L. Translation of DDL Descriptions of Digital Systems, ECE-77-13. Madison: U. of Wisconsin, Dept. of Electrical and Computer Engineering, September 1977, 46 pp.

5. Duley, J.R. DDL--A Disital System Desisn Lansuase. Madison: U. of Wisconsin, Ph.D. Thesis, 1967.

6. Duley, J.R. and Dietmeyer, D.L. "A Digital System Design Language (DDL)," IEEE Trans. Comp. Vol. c-17 (September 1968), pp. 850-861.

7. Duley, J.R. and Dietmeyer, D.L. "Translation of a DDL Digital System Specification to Boolean Equations," IEEE Trans. Comp. Vol. C-18 (April 19691, pp. 305-313.

8. N, Disital Desisn Lansuase Translator--DDLTRN. Madison: U. of Wisconsin, Dept. of Electrical and Computer Engineering, 13 pp.

9. N, Disital Desisn Lansuase Simulator--DDLSIM. Madison: U. of Wisconsin, Dept. of Electrical and Computer Engineering, 36 pp.

10. Soares, L.E.R. An Implementation of Disital Desiqn Lansuase. Madison: U. of Wisconsin, Dept. of Electrical and Computer Engineering, M.S. thesis, 1970.