LOGIC SYNTHESIS FROM DDL DESC'

rrepared

SAJJAN G. SHIVA
Computer Science Department
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE
Huntsville, Alabama 35807

(NASA-CR-161386) LOGIC SYNTHESIS FROM DDL DESCRIPTION Quarterly Progress Report, 1 Oct. - 31 Dec. 1979 (Alabama Univ. in Huntsville.) 18 p HC A02/MF A01 CSCL 09B

N82-20905

G3/61

Unclas
16743

January 1980

Fifth Quarterly Progress Report
October 1, 1979 - December 31, 1979
Contract NAS8-33096
"DESIGN SYNTHESIS OF DIGITAL SYSTEMS"
George C. Marshall Space Flight Center
Alabama, 35812

FORWARD

This is a technical summary of the progress made since October 1, 1979 by The University of Alabama in Huntsville towards the fulfillment of the contract NAS8-33096, from the George C. Marshall Space Flight Center, Alabama. The NASA Technical officer for this contract is Mr. Robert E. Jones, Electronics and Controls Laboratory.

LOGIC SYNTHESIS FROM DDL DESCRIPTION

The implementation of DDLTRN and DDLSIM programs on SEL-32 Computer System is now complete. These programs were tested with DDL descriptions of various complexity. Testing with newer system descriptions will continue.

An algorithm to synthesize the combinational logic using the cells available in the standard IC cell library was formulated. This algorithm is now being implemented as a FORTRAN program. A description of the algorithm is given in the Appendix.

Some corrections were made to the MINICOMPUTER description (Example 5) in the First Annual Technical Report. An updated version of the description and the corresponding simulation results are shown in Figure 1.

The future work includes the completion of logic synthesis algorithm implementation, extension of the algorithm to include the synthesis of memory elements, registers and the register-transfer equations.

DIGITAL DESIGN LANGUAGE TRANSLATOR

```
<SY>MINI:
<RE>MAR(0:7),MBR(0:11),PC(0:7),ACC(0:11).
<RE>IR(0:11)=OP(3)|IBI|ADR(8),RUN.
<RE> X(0:11).
<ME>M(256:12).
<TE>MBUS(12).
<LA>START.
<TE>P(4).
<TI>R.
<OP>CNTUP(8)$X$
     <TE>X(8),C(8).
     <ID>CC=(C(2:8)|1D1).
     <BO>C=X*CC,CNTUP=X@CC..
<OP>SUM(12)$X,Y$
     <TE>X(12),C(12),Y(12),COUT(12).
     <ID>CIN=COUT(2:12)|0D1.
     <BO>COUT=X*Y+X*CIN+Y*CIN,
          SUM=X@Y@CIN..
<AU>CLK(2):R:
     <ST>S(0):START:P=8D4,->T.
          T(1):P=4D4,->J.
          J(2):P=2D4,->L.
          L(3):P=1D4,->S...
<AU>CPU(4):R:
       <ST>IN(0):START:]P(4)]ACC<-0,MAR<-PC,MBR<-0,X<-0,
                     RUN<-1,->FE..
         FE(1):RUN:]P(1)]MAR<-PC.,]P(2)]PC<-CNTUP$PC$,
                     MBUS=M(MAR),MBR<-MBUS.,]P(3)]IR<-MBR.,
                     ]P(4)]]OP(1)*OP(2)*OP(3)]RUN<-0,->IN;
```

Figure 1(a)

MINI Computer Description

(DDLTRN Input)

```
                        ]IBIT]->DEF;->EX....
         DEF(2):]P(1)]MAR<-ADR.,]P(2)]MBUS=M(MAR),MBR<-MBUS.,
                    ]P(3)]ADR<-MBR(4:11).,]P(4)]->EX..
         EX(3):P(4):]OP#0D3->XAND #1D3 ->XAND #2D3 ->XISZ
                    #3D3 ->XDCA #4D3 ->XJSR #5D3 ->XJMP #6D3 ->XRE
         XAND(4):]P(1)]X<-ACC.,]P(2)]MAR<-ADR.,
                    ]P(3)]MBUS=M(MAR),MBR<-MBUS.,
                    ]P(4)]]OP(3)]ACC<-MBR∧X;ACC<-SUM$MBR,X$.,->FE.
         XISZ(5):]P(1)]MAR<-ADR.,
                    ]P(2)]MBUS=M(MAR),MBR<-MBUS.,
                    ]P(3)]MBR<-SUM$MBR,1D12$.,
                    ]P(4)]MBUS=MBR,M(MAR)<-MBUS,]↑(+/MBR)]
                        PC<-CNTUP$PC$.,->FE..
         XDCA(6):]P(1)]MBR<-ACC.,]P(2)]MAR<-ADR.,
                    ]P(3)]ACC<-0,MBUS=MBR,M(MAR)<-MBUS.,]P(4)]->FE.
         XJSR(7):]P(1)]MBR<-CD4[PC.,]P(2)]MAR<-0.,
                    ]P(3)]MBUS=MBR,M(MAR)<-MBUS.,
                    ]P(4)]PC<-ADR,->FE..
         XRET(8):]P(1)]MAR<-0.,]P(2)]MBUS=M(MAR),
                      MBR<-MBUS.,]P(4)]PC<-MBR(4:11),->FE..
          XJMP(9):]P(1)]PC<-ADR.,]P(4)]->FE.....
   <FL>3,4,5,6,8.
```

Figure 1(a): Continued

```
<FL>4,8
<IN>M(0:3)/5,6,7,8
<IN>M(4:7)/4092,0,0,0
<IN>M(8:14)/5,774,1030,1028,2569,1543,3584
<RE>IN/PC/8
<IN>START/1/
<OU>IN/CPU,IR,PC,MAR/
<OU>FE/MAR,MBR,IR,PC,ACC/
<OU>EX/MAR,MBR,IR,PC,ACC/
<SI>
$EOJ
```

Figure 1(b): DDLSIM Input

DIGITAL DESIGN LANGUAGE SIMULATOR VERSION MSFC 1979 SIMULATION RUN

TIME	CPU	IR	PC	MAP	MAR	MBR	IR	PC	ACC	MAR	MBR	IR	PC	ACC
0	00	0000	000	000										
8					008	0000	0000	008	0000					
16										008	0005	0005	009	0000
32					005	0000	0005	009	0000					
48										006	0000	0768	010	0000
64					000	0005	0768	010	0005					
72										010	1030	1030	011	0005
88					006	0001	1030	011	0005					
96										011	1028	1028	012	0005
112					004	4093	1028	012	0005					
120										012	2569	2569	013	0005
136					012	2569	2569	009	0005					
152										006	0001	0769	010	0005
168					001	0006	0769	010	0011					
176										010	1030	1030	011	0011
192					006	0002	1030	011	0011					
200										011	1028	1028	012	0011
216					004	4094	1028	012	0011					
224										012	2569	2569	013	0011
240					012	2569	2569	009	0011					
256										006	0002	0770	010	0011
272					002	0007	0770	010	0018					
280										010	1030	1030	011	0018
296					006	0003	1030	011	0018					
304										011	1028	1028	012	0018
320					004	4095	1028	012	0018					
328										012	2569	2569	013	0018
344					012	2569	2569	009	0018					
360										006	0003	0771	010	0018
376					003	0008	0771	010	0026					
384										010	1030	1030	011	0026
400					006	0004	1030	011	0026					
408										011	1028	1028	012	0026
424					004	0000	1028	013	0026					
432										013	1543	1543	014	0026
448					007	0026	1543	014	0000					
456	00	3584	015	014										

END OF FILE REACHED ON INPUT
SIMULATION TERMINATED AT TIME = 457

Figure 1 (c): DDLSIM Output

APPENDIX

COMBINATIONAL LOGIC SYNTHESIS FROM AN HDL DESCRIPTION*

Sajjan G. Shiva

Computer Science Department
The University of Alabama in Huntsville
P.O. Box 1247
Huntsville, Alabama 35807
(205) 895-6088

ABSTRACT

Hardware Description Languages are used to input the details of a digital system into an automatic design system. An algorithm to synthesize combinational logic from the description in one such language (DDL) is discussed.

*Submitted to the 17th Design Automation Conference Minneapolis, Minn. June 1980.

1. INTRODUCTION

Hardware Description Languages (HDL) provide a convenient means of inputting the digital system design details into a design automation system. Although HDLs were originally designed to be just description media, they have been used in other functions such as simulation, fault test generation, microcode generation, documentation, etc.. The use of HDLs in LSI design automation systems is not widespread because of the difficulty in translating the HDL description into logic diagrams (or connectivity lists or equivalent), non-familiarity of the hardware designers with high-level language programming aspects, non-uniform design methodologies and the time and cost involved in transporting and tailoring the HDL software developed at one design center to the other. However, the advent of VLSI forces that the design be thoroughly verified at the earliest possible time in the design cycle to minimize the fabrication costs brought about by the final changes in a design. Since a suitable breadboard for a VLSI circuit is the VLSI circuit itself, a thorough computer evaluation at the outset is mandatory. Figure 1 [1,2] shows the complete schematic of an IC design automation system. The Computer Aided Design and Test System (CADAT) of the NASA Marshall Space Flight Center [1] is organized as in Figure 1. Digital Systems Design Language (DDL) [3] has been selected [1] for the CADAT system. This paper addresses the problem of hardware compilation from the DDL description, i.e., the process of converting the output of the DDL translator into logic diagrams (or connectivity list, net list, etc.).

2. DDL TRANSLATOR [3]

The DDL translator converts the DDL description of the digital system into a facility table, a set of Boolean equations and a set of register transfer equations. Figure 2 shows an example description. . .e Boolean equations generated by the translator are in Sum of Products (SOP) form. The Boolean functions in the DDL description that were not in the SOP form are retained as they are by the translator. The designer can thus generate all the Boolean equations in the SOP form only if he desires. Hence, the synthesis procedure discussed here assumes a SOP form for the Boolean functions.

3. THE STANDARD CELL LIBRARY

Table 1 shows a partial list of the standard cells available in the CADAT system. Number of devices for each cell and the cell width (as a measure of the silicon area needed) are also shown. The last column shows the literals in each product term of the function realized by the cell. Note that the patterns containing all 1s (11, 111, 1111) and those with one product term (1, 2, 3, 4) correspond to a single gate realization. It is desirable to realize a function by using larger standard cells (if possible), as shown by the implementations shown in Figure 3 for an example Boolean function. The standard cell library provides four cells that can realize a larger function than a gate equivalent, i.e., 2222, 2112, 222, and 22. Also note that the maximum number of inputs to a cell is 8 (2222). Hence, we limit the number of literals in a Boolean function to be realized to 8. A function larger than this needs to be realized in several 8 literal units. For example,

$Z = P + Q$ where P and Q are 8-literal units

$= \overline{\overline{P} \cdot \overline{Q}}$ $\overline{P}$ and $\overline{Q}$ are separately realized. A NAND cell is used to combine them to form Z

4. THE SYNTHESIS ALGORITHM

1) Scan the Boolean function to be implemented and count the number of literals in each product term to generate the literals/product term pattern for the function.

2) If the function pattern is that of a SUM term (patterns containing only 1s: 1, 11, 111, etc.): implement using the NOR cells with proper number of inputs followed by a NAND cell; stop.

3) If the function pattern is that of a PRODUCT term (patterns 2, 3, 4, etc.): implement using the NAND cells with proper number of inputs followed by a NOR cell; stop.

4. Reduce the product terms with more than 2 literals into a term with 1 literal (these terms are implemented as in Step 3). If the function pattern reduces to all 1s go to Step 2, else proceed.

5. Scan the function pattern to identify the standard patterns: 2222, 2211, 222, 22 in that order. Eliminate the matching portion of the function pattern if the patterns or partial patterns are found (A partial pattern is one which matches the standard cell pattern everywhere except in one digit, for example:

   2221 is implemented as 2222

   2111 is implemented as 2211

   21 is implemented as 22)

6) If the function pattern is exhausted, stop. If not, go to Step 2.

Note that the algorithm does not minimize the Boolean function. Only the literals are counted, not the actual number of input variables needed. This might result in slightly higher cost implementations of some functions when the same input variable repeats. For example:

A + BE + CDE

could be implemented with two cells: 1880 and 1220. (14 devices, 20.7 mils). Using the algorithm:

	A + BE + CDE
Step 1)	1 2 3
Step 4)	1 2 1
Step 5)	[1] [2 1]
Step 2)	[1] [2 1] ← From Step 5

The implementation needs three cells: 1870 and 3 , 1220s assuming that $\bar{A}$ is not available. The cost is (20 devices, 27 mils).

CONCLUSIONS

An algorithm for selecting standard cells for implementing the combinational logic is presented. The algorithm is suitable for implementation as a computer program. The complete synthesis algorithms for the CADAT system are now being investigated. These algorithms extend the algorithm presented here to include the memory cells (flip-flops) and corresponding register-transfers. But the algorithm presented here is suitable for any LSI design environment.

ACKNOWLEDGEMENTS

This work is supported by the NASA-Marshall Space Flight Center under the contract NAS8-33096.

REFERENCES

[1] S. G. Shiva, "Use of DDL in an Automatic LSI Design System," Proc. International Symp. CHDLs, Palo Alto, CA, October 1979, pp. 28-32.

[2] ______, "Hardware Description Languages - A Tutorial," Proc. IEEE, Dec. 1979.

[3] D. L. Dietmeyer and J. R. Duley, "Register Transfer Languages and Their Translation," in Digital Systems Design Automation: Languages, Simulation and Data Base, M. A. Breuer (ed), Computer Sciences Press, Woodland Hills, CA, 1975.

Table 1: CADAT Standard Cell Library (Partial)

Cell No.	Type	No. of Devices	Cell Width (mils)	Function	Literals/Product Term
1120	2 input NOR	4	5.8	$\overline{A + B}$	1,1
1130	3 input NOR	6	7.7	$\overline{A + B + C}$	1,1,1
1140	4 input NOR	8	9.6	$\overline{A + B + C + D}$	1,1,1,1
1220	2 input NAND	4	5.8	$\overline{A \cdot B}$	2
1230	3 input NAND	6	7.7	$\overline{A \cdot B \cdot C}$	3
1240	4 input NAND	8	9.6	$\overline{A \cdot B \cdot C \cdot D}$	4
1310	Buffer Inverter	2	3.9	$\overline{A}$	1
1620	2 input AND	6	5.8	$\overline{A \cdot B}$	2
1630	3 input AND	8	7.7	$A \cdot B \cdot C$	3
1640	4 input AND	10	9.6	$A \cdot B \cdot C \cdot D$	4
1720	2 input OR	6	5.8	$A + B$	1,1
1730	3 input OR	8	7.7	$A + B + C$	1,1,1
1740	4 input OR	10	9.6	$A + B + C + D$	1,1,1,1
1800	4 x 2 input AND + 4 x NOR	16	17.2	$\overline{(AB + CD + EF + GH)}$	2,2,2,2
1840	3 x 2½ input AND + 2 input NOR	10	11.6	$\overline{C(AB + DE)}$*	—
1960	2 x 2 input AND + 4 input NOR	12	13.7	$\overline{AB + E + F + CD}$	2,1,1,2
1870	2 x 2 input AND + 2 input NOR	8	9.6	$(\overline{AB + CD})$	2,2
1880	2 bit carry Anticipate	10	14.9	$\overline{(CDE) + BE + A}$*	—
1890	3 x 2 input AND + 3 input NOR	12	16.9	$\overline{AB + CD + EF}$	2,2,2
2310	2 input EXOR	8	7.8	$A \oplus B$	1,1

* Special Functions

ORIGINAL PAGE IS
OF POOR QUALITY

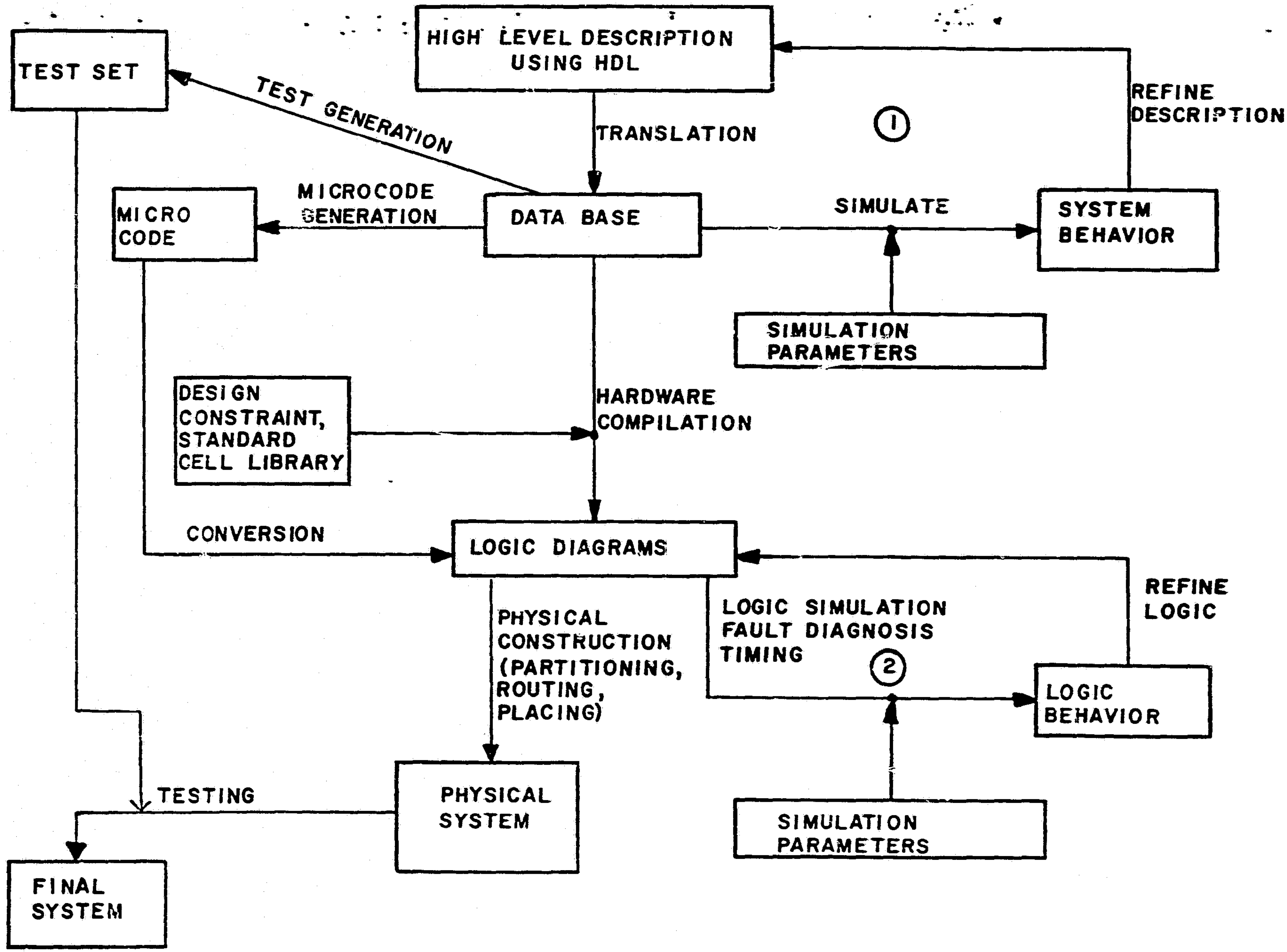

Figure 1 : Digital System Design Automation Process

```
DIGITAL DESIGN LANGUAGE TRANSLATOR

     1:   <SY>COMPLEMENTER:
     2:   <RE>R(1:6),C(2:0),S,I.
     3:   <LA>SW.
     4:   <TI>P.
     5:   <OP> ADD(3)$X$
     6:       <TE> X(3),C(3).
     7:          <ID> CC=(C(2:3)[1E1]).
     8:        <BO> C=X*CC,ADD=X⊕CC..
     9:   <AU>COMP:P:
    10:        <ST>I(0):SW:I<-1,C<-0,S<-0,->S1.
    11:         S1(1):I:]S] R(1)<-↑R(6),R(2:6)<-R(1:5) ;S<-R(6),R<-R(6)[R(1:5)..
    12:          ]C(2)*↑C(1)*C(0)]I<-0,->I;C<-ADD$C>.....
    13:   <FL>3,4,5,6,8.
```

ORIGINAL PAGE IS OF POOR QUALITY

Figure 2(a): DDL Description of a 6 Bit Serial Twos Complementer

DIGITAL DESIGN LANGUAGE TRANSLATOR

FACILITY TABLE

1	LEMENTER	1	1	1	-1	0	339	0	0
2	R	1	6	5	-6	0	0	0	0
3	C	2	0	-3	-6	0	227	358	362
4	S	1	1	1	-6	0	235	291	295
5	T	1	1	1	-6	0	216	321	325
6	SW	1	1	1	-8	0	0	0	0
7	P	1	1	1	-5	0	0	0	0
8	AND	1	3	3	-9	0	0	0	0
9	X	1	3	3	-9	0	191	0	194
10	"1	1	1	1	-9	0	213	0	219
11	C"1	1	3	3	-9	0	0	0	0
12	"2	1	1	1	-9	0	252	0	258
13	COMP	1	1	1	-6	0	244	330	340
14	I	17	0	1	-13	0	195	0	201
15	S1	16	1	1	-13	0	203	0	204
16	1D1		1	1	-17	0	0	0	0
17	6		0	1	-17	0	0	0	0
18	"3	1	1	1	-9	0	255	0	264
19	"4	1	1	1	-9	0	280	0	287
20	"5	1	1	1	-9	0	304	0	316
21	"6	1	1	1	-9	0	337	0	354
22		0	0	0	0	0	0	0	0
23		0	0	0	0	0	0	0	0
24	C"1	3	3	1	11	0	513	0	519
25	C"1	1	2	2	11	0	521	0	531
26	ADD	3	3	1	8	0	489	0	497
27	ADD	1	2	2	8	0	499	0	511
28	R	2	6	5	2	0	456	463	454
29	R	1	1	1	2	0	480	487	478

Figure 2(b) Facility Table (DDL Translator Output)

```
<SY> LEMENTER:
     I=*/COMP'0,
     S1=*/COMP'1D1   ,
     "1=I*S0,
     "2=S1*T,
     "3="2*S,
     "4="2*↑S,
     "5="2*C(2)*↑C(1)*C(0),
     "6="2*↑(C(2)*↑C(1)*C(0)),
       C"1(1:2)=X(1:2)*C"1(2:3),
     C"1(3)=X(3)*1D1   ,
       ADD(1:2)=(X(1:2)@C"1(2:3)),
     ADD(3)=(X(3)@1D1   ),
     ]P*"1 + P*"5] T<-"1*1D1    + "5*0.,
     ]P*"1 + P*"6] C<-"1*C  + "6*↑C.,
     ]P*"1 + P*"4] S<-"1*0  + "4*P(6).,
     ]P*"1 + P*"5] COMP<-"1*1D1    + "5*C.,

     ]P*"3 + P*"4] R(1)<-"3*↑R(6)  + "4*P(6).,
     ]P*"3 + P*"4] P(2:6)<-"3*P(1:5)  + "4*P(1:5).,
     X="6*C,  .
```

Combinational Logic

Register Transfers

NOTES: * = AND, ↑ = NOT, @ = EXOR, + = OR, ← = Transfer, → = GO TO

]] = IF ... THEN

1D1 = 1 bit Decimal 1.

Commas separate the equations

" is part of the variable name

Figure 2(c): DDL Translator Output Equations

AB + CD + EF + G - Function to be implemented

2 2 2 1 - Pattern

	Implementation	Cells Needed	No. of Devices	Area (Mils)
1	2 2 2 1 2 2 2 2	1800	16	17.2
		1220	4	5.8
	*Total Cost		20	23.0
2	[22] [21] ⇓ ⇓ [22] [22]	1870	8	9.6
		1870	8	9.6
		1220	4	5.8
	Total Cost		20	25.0
3	[2] [2] [2] [1]	4 x 1220	16	23.2
		1240	8	9.6
	Total Cost		24	32.8

* Least Cost Implementation

Figure 3: Implementation Cost Comparison
for AB + CD + EF + G