SU-SEL 77-014

A STRUCTURAL DESIGN LANGUAGE FOR COMPUTER AIDED DESIGN OF DIGITAL SYSTEMS

W. M. vanCleemput

Technical Report No. 136

April 1977

The work described herein was supported by the
Joint Services Electronics Program
under Contract No. N0001 4-75-C-0601.

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY . STANFORD, CALIFORNIA

SU-SEL 77-014

A STRUCTURAL DESIGN LANGUAGE FOR COMPUTER AIDED DESIGN OF DIGITAL SYSTEMS

W. M. vanCleemput

Technical Report No. 136

April 1977

Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

The work described herein was supported by the Joint Services Electronics Program under Contract No. N00014-75-C-0601.

A STRUCTURAL DESIGN LANGUAGE FOR COMPUTER AIDED DESIGN OF DIGITAL SYSTEMS

W. M. vanCleemput

Technical Report No. 136

April 1977

Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

ABSTRACT

In this report a language (SDL) for describing structural properties of digital systems will be presented. SDL can be used at all levels of the design process i.e. from the system level down to the circuit level. The language is intended as a complement to existing computer hardware description languages, which emphasize behavioural description. The language was motivated partly by the nature of the design process.

INDEX TERMS: Computer-aided design, hardware description languages, design automation, structural design language, design methodology.

1. THE HARDWARE DESIGN PROCESS

The design of digital hardware is frequently considered as a hierarchical process. Often, a distinction is made between well-established levels of abstraction in this design process. The system level, register-transfer level, gate-level and circuit level are some of these levels.

Two systematic approaches to systems design are often proposed. In the top-down approach a system is decomposed into collections of interconnected subsystems until one reaches elementary components. In the bottom-up approach, more elementary components are combined to form more complex ones.

In practice however, neither a pure top-down nor a bottom-up approach is used, but rather a combination of both, depending greatly on the individual designer., Furthermore, in actual design practice, it is not appropriate to predetermine the levels of detail at which a designer should work. These levels are chosen by the individual and are very dependent on the technology used for implementing a design.

In the early stages of the design process, most emphasis is placed on the behaviour of the system, i.e. on a complete behavioural description. In the later stages the designer adds more and more structural information to the design in progress until the physical design can be implemented using available components.

Depending on the purpose the designer has at a certain stage of the design process, the hierarchical relationships may be quite different. Figure 1 illustrates this difference. For instance, a given gate-level design may, for the purpose of logic simulation, lead to two different elementary gate models, depending on the nature of the logic simulation program used.

2. DIGITAL DESIGN LANGUAGES

2.1 Introduction

In the last decade, a large number of languages has been proposed for assisting in digital hardware design. The potential applications of these languages are the description of a design, input to a (functional level) simulator and automated synthesis from a high-level description.

One of the first such languages was Chu's CDL [Ch65], which allowed for the description and simulation of synchronous systems. The DDL language [DD68] allowed for the description of both synchronous and asynchronous systems and also provided for the automatic generation of an actual hardware design [DD69]. The ISP and PMS notations were originally introduced as a tool for describing computer architectures [BN71]. More recently ISP has been used as the input to a design automation system with automated hardware generation capabilities [BS75].

Currently existing automated hardware generation capabilities are limited to a small subset of actual technology available. The two systems mentioned above map either into NAND gates or into register-transfer modules. The main problem in writing a hardware compiler is that one has to update it continuously in terms of the set of hardware primitives to be used as technology and design practices evolve. But even if technology did not advance, it would be very difficult indeed to force hardware designers to adhere to a rigid scheme for realizing certain functions.

It seems more appropriate to give every designer the possibility of mapping his own higher-level primitives into lower-level ones. The SDL language described in this paper can be used as a vehicle for implementing such a computer-aided designer-controlled logic mapping system.

2.2 Level of Accuracy and Relation to the Physical Hardware

Most of the existing digital design languages emphasize the description of a design at the register-transfer level with the possibility of some description at a lower level, i.e. the module or gate level. Usually it is impossible to describe the system accurately at this lower level. This is illustrated by the simple example of Figure 2, where four different realizations of the expression $X = A \wedge B \wedge C$ are shown. Each of these implementations may have a slightly different dynamic behaviour, but this is not clear from the expression.

In most register-transfer languages, it is difficult, if not impossible, to express the fact that more than one instance of an operator is required in order to realize the system. Consider, for example, the following CDL [Ch65] statements:

```
/condition1/ A = B + C
/condition2/ X = Y + Z
```

Depending on whether or not condition 1 and condition 2 are mutually exclusive, one or two physically distinct adders are required. The problem is that the designer usually is aware of this. If he would be able to express the physical structure at this point, it would be possible to verify his decision, during the register-transfer-level simulation. But even when the conditions are mutually exclusive a designer may still want to use different instances for a given operator. Consider the following CDL example, where K(1) and K(5) are mutually exclusive:

```
    /K(5)*CLOCK/ AC = AC + MDR
```

which states that when control signal K(5) and CLOCK are active then the sum of the registers AC and MDR is transferred into register AC. Similarly one can write:

```
    /K(1)*CLOCK/ PC = PC + 1
```

which describes the incrementation of the program counter called PC. In most implementations these two additions will be performed by different devices, yet this is not clear from the description.

This description in CDL can be changed to imply more of the system's structure in the following way:

```
    /K(5)*CLOCK/ AC = AC .ADD. MDR
    /K(1)*CLOCK/ PC = PC .INC.
```

where ADD and INC are user-defined operators.

Although both descriptions are correct they may imply very different physical implementations.

Similarly, if the designer would be able to specify the data paths required at the register-transfer level, then the validity of the various transfers could be validated easily.

It is clear that currently the translation of a register-transfer-level design into one that can be physically implemented using a given technology is a task that a designer has to perform manually and for which little or no computer-aided design tools are available.

It is, however, desirable to capture whatever structural information a designer may have decided upon at any level of the de'sign process and to use this structural information, together with the behavioural description, to validate the design where possible.

2.3 The Need for Structural Information

In the initial design stages one is mainly concerned with the behaviour of a system. Gradually a designer adds more and more structural information to the design until it can be implemented using physical components. Often some structural requirements are part of the original design specifications. Because of this a language is needed that can be used by the designer at all levels of the design process and that allows him to record accurately all the information that is pertinent to his design, especially physical structure.

When structural information is available then he may be able to check his behavioural description against the structural description for possible inconsistencies as was pointed out in the previous section.

3. LANGUAGE CONCEPTS

3.1 Introduction

The main objectives of the SDL language are:

1) accurate representation of structural information.

2) to be useful over all levels of the design process.

3) to be applicable to the different purposes a designer may have during the design process.

4) to be able to perform designer-controlled mapping of higher-level hardware primitives into lower-level ones.

In order to achieve these objectives, a language and an associated compiler were developed for describing system connectivity. Depending on the purpose of the designer, the description will be expanded by means of a macro facility into a structural description that suits a particular purpose.

3.2 Description of Structural Information

Each system or subsystem is characterized by a name as follows:

NAME:<name>;

Every system is connected the outside world by external connections. This is specified as follows:

EXT:<external connector name>:<terminal name list>;

where <external connector name> refers to the (optional) name for the external connector(s) and <terminal name list> is a list of terminal names in the order in which they appear in the physical system. A terminal name is either a simple identifier or a range.

A valid simple identifier is any combination of letters and digits such as VCC, GND, 7474. A range consists of two numbers separated by a colon e.g. 21:25. A range of numbers is expanded into a list of numbers starting with the first bound up to the last bound using 1 or -1 as an increment. For example, the following EXT declaration

```
     EXT:P1:VCC,2:4,GND,9:7;
```

is equivalent to:

```
     EXT:P1:VCC,2,3,4,GND,9,8,7;
```

It is frequently useful to distinguish between inputs and outputs of a system. This information may e.g. be used for checking the consistency of a design. This may be specified as follows:

```
     INPUTS:<pin list>;
     OUTPUTS:<pin list>;
```

where <pin list> is a list of pin names and <pin name>::= <component name>.<terminal name>. A component name is either a simple identifier as defined before or a subscripted identifier.

Another property of the external terminals of a component that they may be logically equivalent. If this is the case, the signals connected to a set of equivalent terminals may be permitted without changing the logical function of the system. This is e.g. the case for the inputs of an m-input AND, NAND, or NOR gate. Consider a 4-input NAND gate as in Figure 3(a), with inputs 11, 12 and 13 and output 0. This can be expressed as follows:

EQUIVALENCE: 11, 12, 13;

A more complicated case of logical equivalence is illustrated by the triple 3-input NAND gate of Figure 3(6). In this case the gates themselves can be interchanged. This is expressed in the following way:

EQUIVALENCE:(1A, 1B, 1C)-1Y,(2A, 2B, 2C)-2Y,(3A, 3B, 3C)-3Y;

Within the context of this language, it is necessary to explicitly declare the types of all components to be used in the description:

TYPE:<type name list>;

All type names have to be simple identifiers. An example is the following declaration:

TYPE:7400, 74S74, NAND, SN7410;

For every type name, one has to declare all the components that are of that type:

<type name>:<component name list>;

Component names may be either simple identifiers or subscripted identifiers, where

<subscripted identifier>::=

<simple identifier> '<' <subscript list> '>'

For example, the following declaration

74S74:C<1, 3:5, A, B6>, D5;

is equivalent to

74S74:C<1>, C<3>, C<4>, C<5>, C<A>, C<B6>, D5;

Finally one has to define all the interconnections in terms of nets. Two possibilities exist: directed nets and undirected nets. The format is as follows:

```
<net name> = <pin list>;
<net name> = FROM (<pinlist>) TO (<pin list>);
```

A net name can be either a simple identifier or a subscripted identifier. It is possible to use this declaration for defining busses. This is done using the concatenation operator ('-') as illustrated in the following example:

```
ABUS<1:4>=C<1:4>.5, C<9:12>.7;
```

This bus definition is equivalent to:

```
ABUS<1> - ABUS<2> - ABUS<3> - ABUS<4> =
C<1>.5 - C<2>.5 - C<3>.5 - C<4>.5,
C<9>.7 - C<10>.7 - C<11>.7 - C<12>.7;
```

Which in turn is equivalent to the following four simple net declarations:

```
ABUS<1> = C<1>.5, C<9>.7;
ABUS<2> = C<2>.5, C<10>.7;
ABUS<3> = C<3>.5, C<11>.7;
ABUS<4> = C<4>.5, C<12>.7;
```

In addition to the previous types of statements, it is possible to declare macros (not to be confused with the macro-expansion of a design over several levels, which will be discussed later on). These macros provide for immediate textual substitution of each string occurrence in the input text.

As an example, consider the following macro definition:

```
MACRO:REGISTER:&1:&2<1:&3>;
```

then the following declarations

```
REGISTER(DF1, ACC, 16);
REGISTER(DF2, PC, 12);
```

would be expanded as:

```
DF1:ACC<1:16>;
DF2:PC<1:12>;
```

which in turn are equivalent to:

```
DF1:ACC<1>, ACC<2>, ACC<3> . . . , ACC<16>;
DF2:PC<1>, PC<2>, PC<3> . . . , PC<12>;
```

3.3 The Concept of PURPOSE

For every description the designer has to specify the possible uses of the description. Examples of usage are: register transfer level simulation, printed circuit board layout, integrated circuit mask layout, fault test generation, gate-level logic simulation, etc..

The form of this declaration is:

```
PURPOSE:<purpose name list>;
```

Note that the name of a purpose may be chosen by the designer, provided that he supplies the necessary library information as explained in the next section.

3.4 The Concept of LEVEL

Besides a purpose,a description also has a level of accuracy or detail. Again, this level is user-defined and not imposed in advance by a rigid system enforcing a previously established design philosophy. A typical example would define gate level, circuit level, etc.. A level is associated with a purpose i.e. there may be more than one description of a system at the same level, depending on the purpose. The form of this declaration is:

LEVEL:<level name>;

For every purpose and level combination, specified by the designer, a library with the name <purpose>.<level> is required. This library contains information on the next level of detail for all the types used at a previous level. In order to indicate the end of a macro expansion for a given purpose, the lowest level description will have a special level, called END, specified.

4. EXAMPLE

The following example illustrates some of the concepts in the language and also shows how a designer may use its capabilities.

The purpose of the designer in this example is to design a 16-bit shift register using available MSI components, in this case an 8-bit shift register, the SN7491 (Figure 4). The description of the 16-bit shift register in function of the primitive components is then:

```
NAME:SHIFT16;
PURPOSE:LOGSIM,PCBGEN,CKTANALYSIS;
LEVEL:TTLPACK;
TYPES:SN7491;
EXT::DATA,ENABLE,CLOCK,Q;
SN7491:C1,C2;
NET1 = FROM (.CLOCK) TO (C1.CLOCK,C2.CLOCK);
NET2 = FROM (.ENABLE) TO (C1.ENABLE,C2.ENABLE);
NET3 = FROM (.DATA) TO (C1.DATA);
NET4 = FROM (C1.Q) TO (C2.DATA);
NET5 = FROM (C2.Q) TO (.Q);
END;
```

This description of the 16-bit register in terms of two 8-bit registers may be used for the purpose of logic simulation (LOGSIM), printed circuit board generation (PCBGEN) and circuit analysis (CKTANALYSIS). The level of this description is defined by the user as TTLPACK. Suppose the designer wants to perform a gate-level simulation of this design. Before doing this he has to translate (map) his logic into the primitives known to the logic simulator e.g. gates, invertors and simple flip flops. For this purpose the previous description has to be expanded by the macroprocessor making use of a macro library called LOGSIM.TTLPACK. The only component that we need for this simple example is the type SN7491. Its description in the library may look as follows (Figure 5):

```
NAME:SN7491;
PURPOSE:LOGSIM,CKTANALYSIS;
LEVEL:GATE;
TYPES:NAND,INV,RS;
EXT::DATA,ENABLE,CLOCK,Q;
INPUTS:.DATA,.ENABLE,.CLOCK;
OUTPUTS:.Q;
NAND:G1;
INV:G2,G3;
RS:FF1,FF2,FF3,FF4,FF5,FF6,FF7,FF8;
NET1 = FROM (.DATA) TO (G1.IN1);
NET2 = FROM (.ENABLE) To (G1.IN2);
NET3 = FROM (.CLOCK) TO (G3.IN);
NET4 = FROM (G3.OUT) TO (FF1.CK,FF2.CK,FF3.CK,FF4.CK,
              FF5.CK,FF6.CK,FF7.CK,FF8.CK);
NET5 = FROM (G1.OUT) TO (G2.IN,FF1.R);
NET6 = FROM (G2.OUT) TO (FF1.S);
NET7 = FROM (FF1.Q) TO (FF2.S);
NET8 = FROM (FF1.QBAR) TO (FF2.R);
NET9 = FROM (FF2.Q) TO (FF3.S);
NET10 = FROM (FF2.QBAR) TO (FF3.R);
NET11 = FROM (FF3.Q) TO (FF4.S);
NET12 = FROM (FF3.QBAR) TO (FF4.R);
NET13 = FROM (FF4.Q) TO (FF5.S);
NET14 = FROM (FF4.QBAR) TO (FF5.R);
NET15 = FROM (FF5.Q) TO (FF6.S);
NET16 = FROM (FF5.QBAR) TO (FF6.R);
NET17 = FROM (FF6.Q) TO (FF7.S);
NET18 = FROM (FF6.QBAR) TO (FF7.R);
NET19 = FROM (FF7.Q) TO (FF8.S);
NET20 = FROM (FF7.QBAR) TO (FF8.R);
NET21 = FROM (FF8.Q) TO (.Q);
END;
```

In the library LOGSIM.GATE, the following information will be present:

```
NAME:RS;
PURPOSE:LOGSIM;
LEVEL:END;
EXT::R,S,CK,Q,QBAR;
INPUTS:.R,.S,.CK;
OUTPUTS:.Q,.QBAR;
END;
```

```
NAME:NAND;
PURPOSE:LOGSIM;
LEVEL:END;
EXT::IN1,IN2,OUT;
INPUTS:.IN1,.IN2;
OUTPUTS:.OUT;
END;

NAME: INV;
PURPOSE:LOGSIM;
LEVEL:END;
EXT: : IN,OUT;
INPUTS:. IN;
OUTPUTS:.OUT;
END;
```

Using these macro definitions, a description can be generated for input to a gate-level logic simulator. Suppose the designer would like to also perform a circuit simulation of the circuit; in this case he has to expand the description using the CKTANALYSIS libraries.

For example, in the library CKTANALYSIS.GATE the following description of an invertor called INV may be present (Figure 6):

```
NAME:INV;
PURPOSE:CKTANALYSIS;
LEVEL:COMPONENT;
TYPES:TRANSISTOR1,DIODE2,R6K,R4K,R2K,R1K,R100;
EXT::IN,OUT,VCC,GND;
INPUTS:.IN;
OUTPUTS:.QUT;
TRANSISTOR1:T1,T2,T3,T4;
DIODE: D1 :
R6K:R1;
R4K:R2;
R2K:R5;
R1K:R4;
R100:R3;
NET1 = .IN,T1.E,D1.A;
NET2 = D1.B,.GND,R4.B,T3.E,T4.E;
NET3 = .VCC,R1.A,R2.A,R5.A;
NET4 = R1.B,T1.B;
NET5 = T1.C,T2.B;
```

```
NET6 = R2.B,T2.C;
NET7 = T2.E,R3.A;
NET8 = R3.B,R4.A,T3.B;
NET9 = R5.B,T3.C,T4.B;
NET10 = T4.C,.OUT;
END;
```

If there is a model for the transistor and diode specified above in the circuit analysis program then the library CKTANALYSIS. COMPONENT will contain the following description of the transistor of type TRANSISTOR1:

```
NAME:TRANSISTOR1;
PURPOSE:CKTANALYSIS;
LEVEL:END;
EXT::B,E,C;
END;
```

If there is no such model available, then one has to be specified using more primitive elements.

5. Macro Expansion of a Design

One of the purposes of a system description in SDL is to be able to expand this description into one at a lower level for a certain purpose. Because of the structure of the system, it is up to the designer to define levels and to describe building blocks at the various levels.

In order to expand a description from a level A to the next lower level B all types T_i declared at the higher level (A) have to be defined at the lower level (B).

The set of types of the expanded circuit (at level B) consists of the union of the types declared in the description of T_i at level B. For the example of Figure 7, the only types used are NAND, INV and RES.

The set of external signals of the expanded circuit is identical to the set of external signal at level A. Input, output and equivalence declarations for these are obviously the same.

Components are expanded in the following way: a component x of type y at level A is replaced by the components z_i, defined for type y at level B. In order to preserve uniqueness of names, these new components are renamed as x/z_i. This is illustrated in Figure 7, where the example of Figure 4 consisting of two components Cl and C2 of type SN7491 is expanded into 22 components named Cl/FFl, Cl/FF2, . . . Cl/FF8, C2/FFl, C2/FF2, . . . C2/FF8 and Cl/Gl, Cl/G2, Cl/G3, C2/Gl, C2/G2, C2/G3.

Finally nets are expanded in the following manner:

1) Nets that are totally internal to a type description at level B are simply renamed by prefixing the net name with the component name. For example NET4 of type SN7491 is expanded to Cl/NET4 and C2/NET4 for the two occurrences of that component.

2) Nets that belong to a type description at level B and that are partly external are merged into the net at level A to which they are connected. For example, NET21 of component Cl and NET1 of component C2 at level B are merged into NET4 at level A. The new net NET4 will be: NET4 = FROM (Cl/FF8.Q) TO (C2/Gl.I1);.

6. Future Developments

6.1 Integrated Design System

A large number of currently existing hardware design automation programs make use of some structure-oriented description. Examples of this are logic simulators such as TESTAID and TEGAS [Sz72], circuit analysis programs such as ECAP, SPICE [JM76], fault test generation packages, printed circuit layout systems and integrated circuit layout systems.

It is rather easy to translate a SDL description into any of these input languages. This makes it possible to obtain an integrated computer-aided design system based on existing software packages.

Currently a Printed Circuit Design system and interfaces to logic simulators (TEGAS, TESTAID) and circuit analysis programs are being implemented. In the future, systems for IC layout and logic diagram generation will be considered.

In order to design a printed circuit board from the SDL description, the information has to be supplemented with physical information about the actual components and the board.

6.2 Hierarchical Design Verification

Finite state automata can be used to describe the behaviour of a digital system. The DDL language [Di68] actually uses this formalism for describing systems at the register transfer level.

An important step in the digital design process is to realize the behaviour of a system S by interconnecting a number of subsystems

S(i). The structure of such a realization can be expressed in terms of a structural design language such as SDL. In some cases the synthesis of a given automaton can be automated. However, in many instances an automaton is realized as a composition of more elementary automata. This composition is often arrived at by the designer. The problem is then to validate the manually generated design. The first step in such a design validation procedure is to derive a behaviour for the composition S(1) * S(2) * S(3) . . . when the behaviour of the automata S(i) and the inter-connections are known. The second step is to verify whether the behaviour of this composite design is the same as that of the originally postulated behaviour. This is often done by simulation, although a proof of equivalence of behaviour would be more appropriate.

REFERENCES

[Barbacci and Siewiorek, 1975] Barbacci, M. R. and Siewiorek, D. "Application of an ISP Compiler in a Design Automation Laboratory," Proc. Int. Symp. Computer Hardware Description Languages and their Applications, New York, Sept. 1975, pp. 69-75.

[Bell and Newell, 1971] Bell, C. G. and Newell A. "Computer Structures: Readings and Examples," 'New York: McGraw Hill, 1971.

[Chu, 1965] Chu, Y "An Algol-like Computer Design Language," Comm. ACM, vol. 8, no. 10, pp. 607-615, 1965.

[Duley and Dietmeyer, 1968] Duley, J. R. and Dietmeyer, D. "A Digital System Design Language (DDL)," IEEE Trans. Computers, vol. C-17, no. 9, pp. 850-860, Sept. 1968.

[Duley and Dietmeyer, 1969] Duley, J. R. and Dietmeyer, D. "Translation of DDL Digital System Specification into Boolean Equations," IEEE Trans. Computers, vol. C-18, no. 4, pp. 303-313, 1969.

[Jensen and McNamee, 1976] Jensen, R. W. and McNamee, L. P. "Handbook of Circuit Analysis Languages and Techniques," Englewood Cliffs, N. J.: Prentice Hall, 1976.

[Szygenda, 1972] Szygenda, S. A. "TEGAS2, Anatomy of a General Purpose Test Generation and Simulation System for Digital Computers," Proc. 9th Design Automation Workshop, Dallas, Texas, June 1972, pp. 116-127.

[vanCleemput, 1976] vanCleemput, W. "Computer-aided Design of Digital Systems, a Bibliography," Woodland Hills, California, U.S.A.: Computer Science Press, 1976.

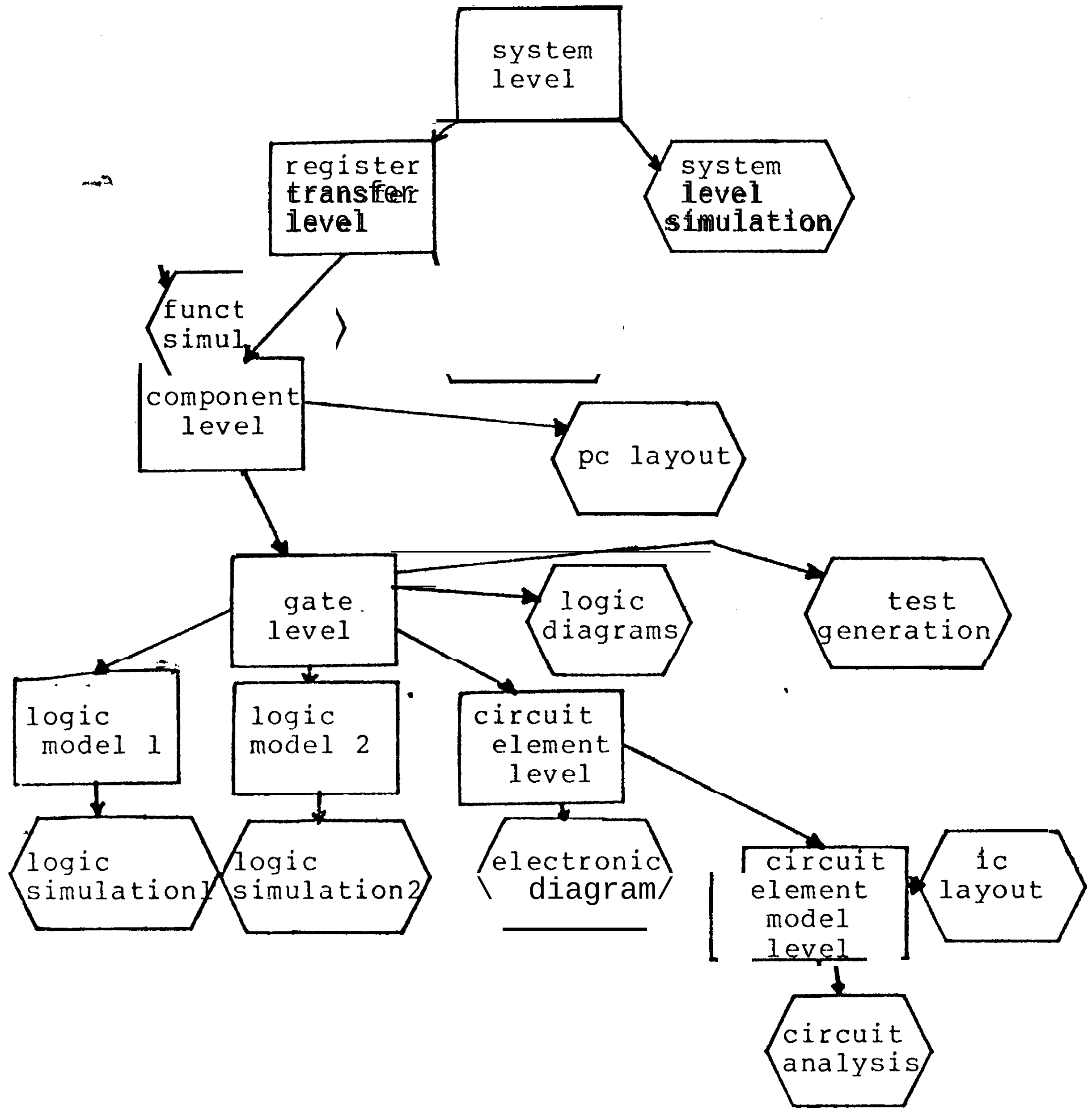

Figure 1 Hierarchical relationship between design levels and purposes.

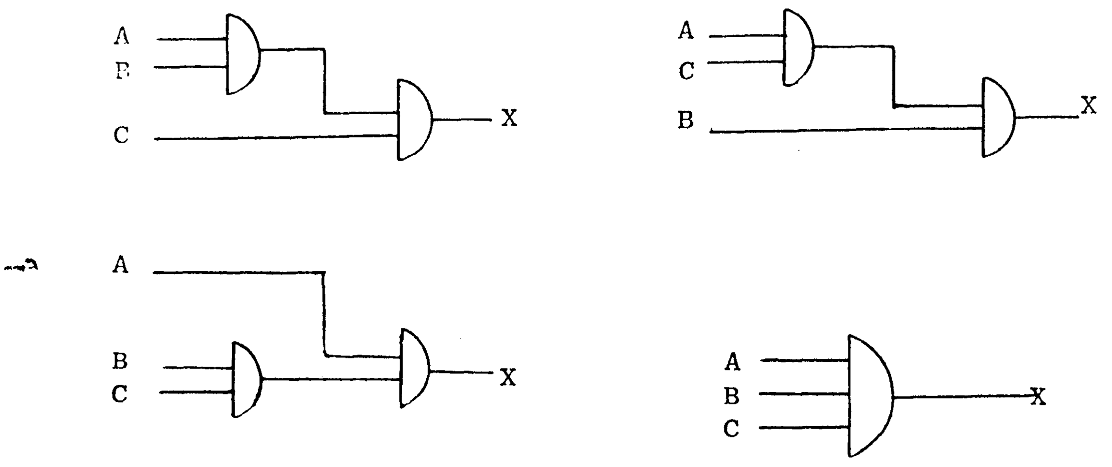

Figure 2 Four different implementations of the expression X = $A \wedge B \wedge C$

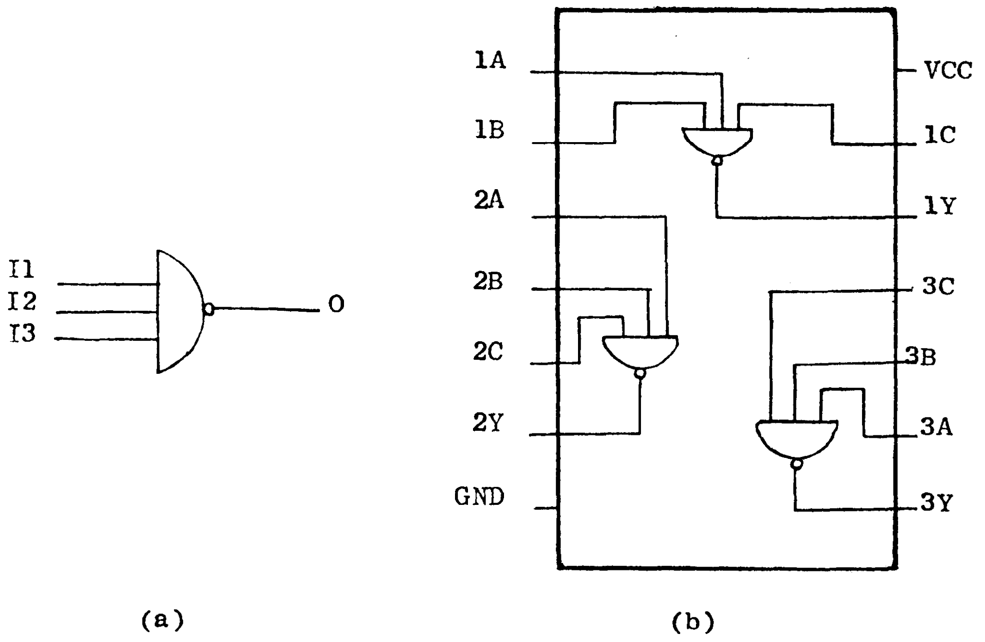

Figure 3 (a) Logical Equivalence of the inputs of a NAND gate.
(b) Logical Equivalence of individual gates of a component (7410)

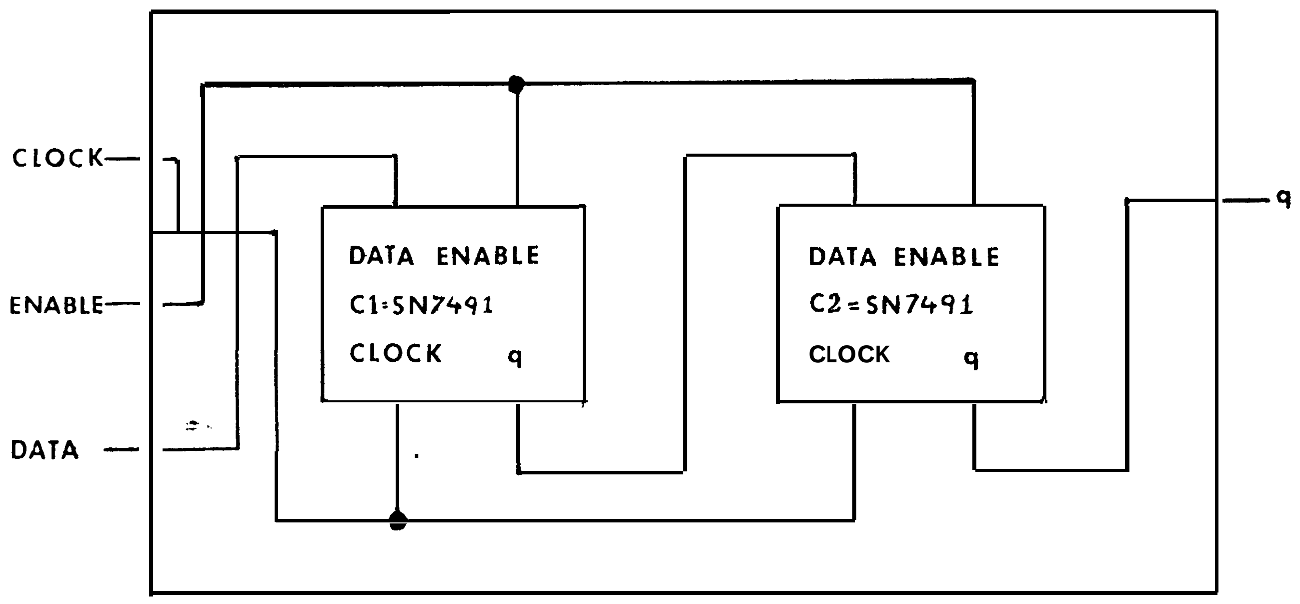

Figure 4 A 16 bit Shift Register using SN7491 Modules.

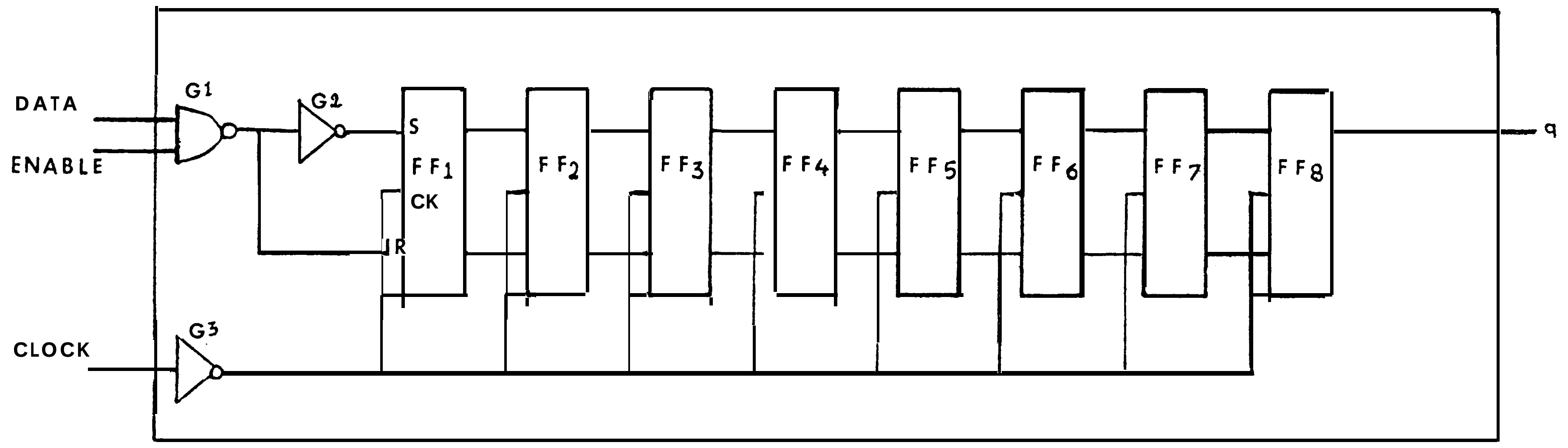

Figure 5 Gate level model of an 8-bit shift register (SN7491)

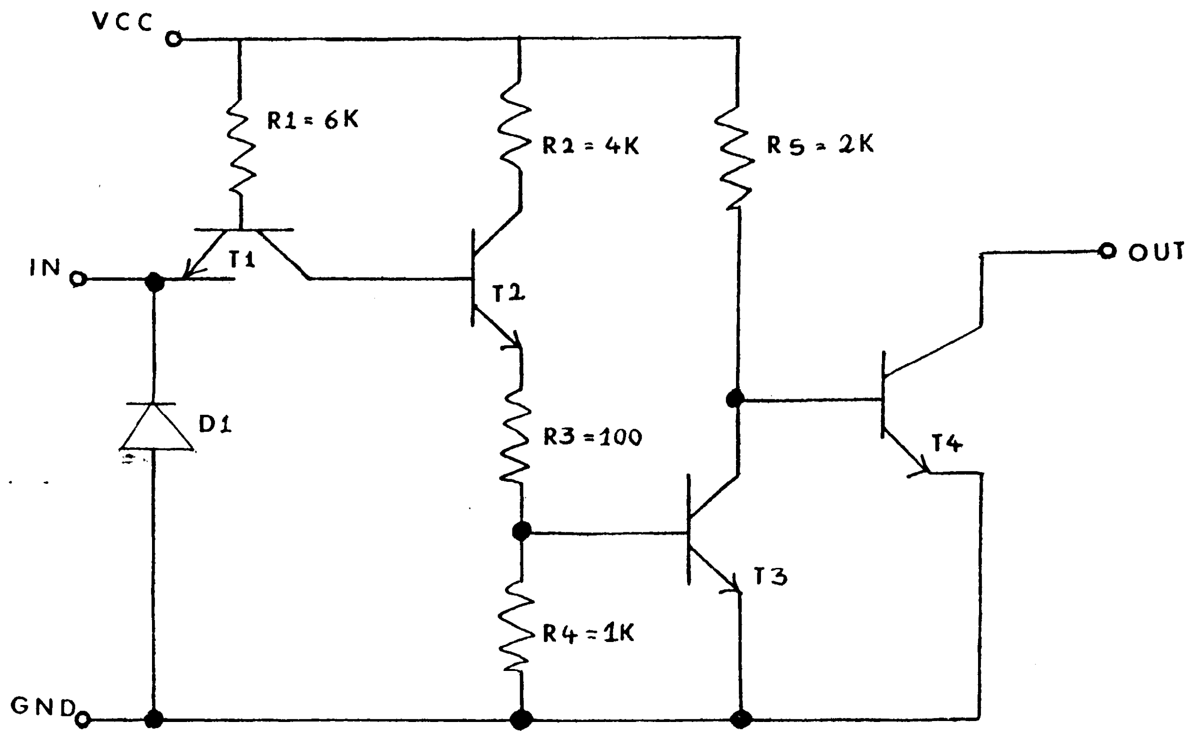

Figure 6 Circuit-level model of a TTL invertor

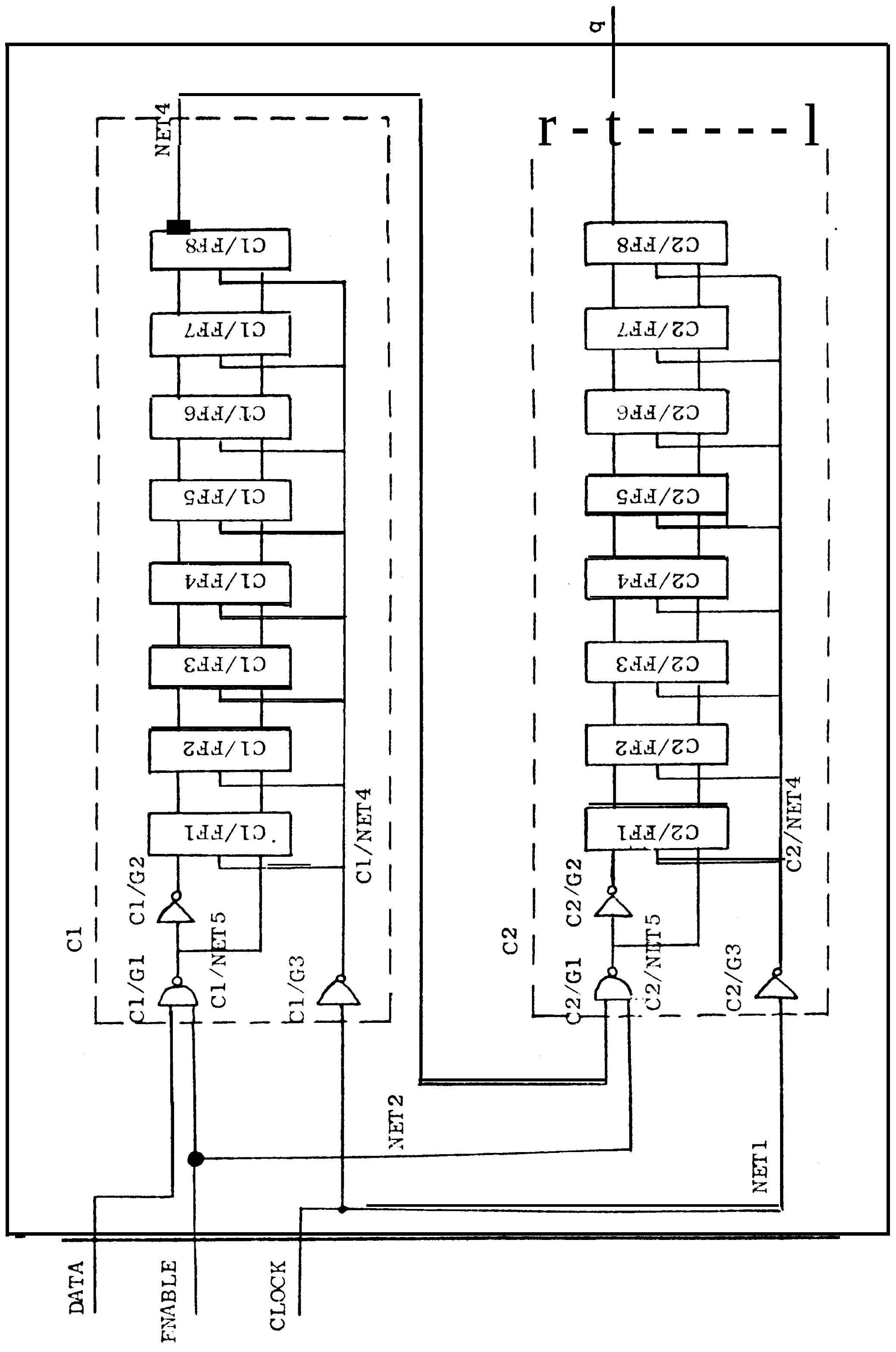

Figure 7 Expansion of the shift register of Figure 4 to the gate level.