COMPUTER AIDED LOGIC DESIGN

Dr. Frank W. Bliss
Bendix Computer Graphics
Farmington, Michigan

This paper describes an interactive computer-aided design system which converts a state table description of a small scale synchronous digital system into a logic diagram. The input to the design programs is a State Table and a Unit Control Table that is generated by another design automation program. The State Table sequences the operations of the system. The Unit Control Table describes the operations that are to be performed on the defined units of the digital system and the conditions under which the operations are to be performed. The output of the design system is a detailed logical design of the control logic of the digital system. The output is in the form of an interconnection diagram that can be read by a digital hardware simulation program.

The Computer-Aided Logic Design (CALD) System is interactive to allow the designer to change various parameters and generate many different logic designs for any particular State Table. The user may specify the state assignment, the type of memory element, and the maximum allowed gate fan-in. CALD assists the designer in performing State Table reduction and generates the state assignment at the users option. The tabular description of the digital machine is translated into a Boolean description resulting in a set of cannonical form Boolean equations in sum of product form. Memory element application equations are generated. Boolean equation minimization is performed followed by a factoring routine to enforce the fan-in constraint. The equations are then translated into a logic diagram.

The CALD system is implemented in FORTRAN and running on a mini-computer with 8K of core.

Introduction

The digital system design process has been partially formalized as a first step toward automation. The process has been partitioned into several phases of activity. Partitioning was performed so that the output of each design phase is a description of the system which serves as the input to the next phase of design. The design process consists of: Conceptual design, Functional design, Logic Design, Hardware Simulation, Implementation Design, and Construction.

The CALD (Computer-Aided Logic Design) System performs the Logic Design phase of the design process (1). CALD is part of a larger Design Automation System and interfaces with a Functional Design System and a Hardware Simulator. It is necessary to briefly describe these other two systems in order to understand the CALD System.

HALSIM

HALSIM (Hardware Logic Simulator) is a programming system used to describe and simulate digital logic designs on the hardware level (2), (3). The input to HALSIM is a detailed logic diagram that has been encoded into the HALSIM input language. The language defines the interconnections between the various logic elements in the system. The simulator possesses a library of standard integrated circuit elements which are used in describing the logical system. Simulation of the digital system is performed on a "gate by gate" analysis in which signal levels propagate through the simulation much the way they would through its hardware equivalent. The program provides the user with information about possible timing hazards encountered during the operation of the system as well as verifying the logical correctness of the design.

The CALD programs generate a logic diagram that is encoded in the HALSIM language and consists of a collection of HALSIM specification statements. These statements form a wiring list which determines a unique interconnection between the elements. The statements are expressed in terms of signal names and logic elements. A signal name is assigned to each signal in the system. Each element in the logic diagram has a specification statement associated with it. The specification statement consists of an input list, an operator, and an output list. The input list consists of the names of all the signals that are inputs to the logic element. The operator is a logic element name from the element library and the output list contains the names of all signals that are outputs of the element. Figure 1 shows the assignment of names to a simple digital system and the specification statements for the system. There is a one-to-one correspondence between the logic diagram and the HALSIM language description of the logic diagram

FST

FST (Functional Simulation and Translation) is a set of computer programs that perform the functional design of a small scale synchronous digital system (4), (5). Upon completion of the conceptual design phase, the description of the digital system consists of a process flow chart specifying the operations to be performed by the system and implying the necessary functions of the control logic. The functional design phase translates this process flow chart description into a block diagram description. The block diagram consists of a list of the defined units of the system showing conditions necessary for an operation to occur in a given unit, and a state table describing the control logic of the system.

The FST system is composed of four major parts:

1. A language to describe the process flow chart of the digital system.
2. A compiler to convert the flow chart description into a list structured description of the system.
3. A simulator to verify that the described system will perform as expected.
4. A translator to generate the unminimized State Table and the Unit Control Table which constitute the block diagram description of the system.

The FST Language

The FST language consists of two main parts: Unit declarations and statements that describe the operations of the defined units. The units allowed are memory elements, registers, binary counters,

decade counters, and binary adders.

The statements describing the operations of the units consist of a list of operations and the conditions when the operations are to be performed. Statements may be of two basic types:

1. WHEN (Condition List) THEN (Operation List)
   The operations listed are to be performed only when the condition list is true.
2. THEN (Operation List)
   The operation list is always performed.

The Condition List is a well-formed Boolean expression of defined units of the system, Boolean literals, and the following operators: Equality, Negation, Logical And, and Logical Or.

The operations allowed on units of the system are:

1. (Unit Name) Set the Unit
2. - (Unit Name) Reset
3. Shift (Register Name) Right N
4. Shift (Register Name) Left N
5. Circulate (Register Name) Right N
6. Circulate (Register Name) Left N
7. Increment (Counter Name)
8. Decrement (Counter Name)
9. Transfer (Name 1) to (Name 2)
10. Transfer (Binary Literal) to (Name)
11. Add (Name) to (Binary Adder Name)
12. Goto (Label)

The sequence of statement executation is determined by the block structure of the program. Any group of statements may be placed in a block and the order of execution of statements within a block may be specified as concurrent or sequenced.

State Table & Unit Control Table

Upon completion of a satisfactory simulation, the FST translator generates an unminimized State Table and a Unit Control Table. The State Table describes the control logic portion of the digital system. The entries in the State Table consist of the present state, next state, and the condition necessary for control to be transferred from present to next state. The condition is a Boolean expression taken from the Condition List of the FST Language statements.

The Unit Control Table describes the operations that are to be performed on the defined units of the system and the conditions under which these operations are to be performed. The defined units are the memory elements, registers, counters and adders specified by the designer under unit declarations in the FST language. The entries in the Unit Control Table consist of a unit name, an operation, and a Boolean expression specifying the conditions necessary for the execution of the operation. The State Table and Unit Control Table completely describe the operation of a digital system.

Examples

An example of a FST input language statement is given below.

```
WHEN  B(4)*(A=CB) THEN Increment Actr, -A,
  Shift Areg Left 1, Transfer 1001 to B(1-4),
  Goto  L1 $
```

The above statement indicates that when bit four of register B is one and when flip-flop A is in the same state as flip-flop CB then increment a counter called Actr, reset flip-flop A, shift Areg one bit left, transfer 1001 to bits one through four of register B, and go to the statement labeled L1.

If the above statement is assigned state Si by the FST compiler and the statement labeled LI is assigned state Sj then the State Table entries for this statement would be:

Present State	Next State	Boolean Condition
Si	Sj	B(4)*(A=CB)
	Si+1	-[B(4)*(A=CB)]

The Unit Control Table would contain an entry specifying that counter Actr is to be increment if the following Boolean Condition is true:

Si * [B(4) * (A=CB)]

The Unit Control Table would contain an entry for each operation in the Operation List of the FST input statement.

The CALD System

The State Table and the Unit Control Table generated by the FST System is read as input to the CALD System. The logic design is performed from these two tables.

The structure of the logic diagram that is generated by CALD is shown in Figure 2. The basic system configuration reflects the fact that the logic design is performed from a state table. The structure contains a bank of memory elements to represent the states of the State Table. The particular memory element configuration generated by CALD associates N memory elements for 2^N states. A decoder is used to decode the memory element states and control logic sequences the operations of the defined units. Specifically, the control logic examines the decoder outputs and the inputs from the defined units, it then generates the memory element input equations and the outputs specified by the Unit Control Table. The inputs from the defined units represent specific conditions or states of the defined units, eg. counter B=0000, flip-flop C is reset. The outputs generated by the control logic specify the operations to be performed on the defined units, eg. shift register A 4 bits to the right, increment counter B. The CALD Programs perform the detailed logical design of the state memory elements, the decoders, and the control logic. This is shown as the enclosed part of Figure 2. The defined units, along with any transfer or decoding circuitry associated with them, are not designed by the programs.

The description of the Computer-Aided Logic Design system is divided into the following sections:

1. Compiler
2. State Table Reduction
3. Library of Hardware Elements
4. State Assignment

5. Specification of memory element type
6. Memory Element Input Equation
7. Fan-in
8. Reduction of Logic Equations
9. Factoring of Logic Equations
10. Fan-Out calculation
11. Translator (to generate the logic diagram)

Compiler

The purpose of the compiler is to translate the State Table and the Unit Control Table into logic equations that can be eventually implemented in hardware. The Boolean Conditions from the tables are converted to parenthesis-free notation and placed in an output list. Each unit name is placed in the NAME table and assigned a compiler generated name consisting of an alphanumeric code. The Polish output list is scanned for subexpressions. A subexpression is a group of adjacent terms (coded unit names) connected by the same operator. The subexpression is arranged in a canonical form based on the coded names, assigned a name, and stored in either the AND table or the OR table depending on the type of the connecting operator. No duplicates are allowed. The subexpression is then removed from the output list and replaced by its name. This process continues until both tables have been processed.

An example is given below to indicate how the compiler operates. Suppose the following expression is the first Boolean Condition in the State Table.

```
(A*B) + C + D*E*F
```

The compiler scans the expression and assigns compiler names I1 through I6 to the names. The following Polish output list is generated:

```
I1 I2 * I3+ I4 I5 I6 ** +
```

The output list is scanned and the equations are generated in the following order.

```
E1 = I1 * I2
E2 = I3 + E1
E3 = I4 * I5 * I6
B1 = E2 + E3
```

Each equation is placed in either the AND or OR table. The "I" represents an input, "E" represents a subexpression, and "B" represents a complete Boolean Condition. The equations are in a standard form based on the priority of the letter and the number in each name.

State Table Reduction

When compilation is complete the system examines the State Table and generates the necessary information to merge states in order to reduce the size of the Table. State Table reduction tends to reduce the number of logic elements required in the logic design. Compiler and State Table Reduction routines constitute the preprocessing phase of the logical design process.

Library of Elements

The CALD programs contain a limited library of digital integrated circuit logic elements based on the HALSIM Library. The library includes a clock element, a NAND gate, an INVERTER, a JK memory element, a Delay memory element, and a Binary to Octal Decoder. Since only the control logic is designed by CALD, this library is adequate for most design problems. The library is small due to the limited amount of storage available on the DDP116 computer (8K). However, provision has been made for increasing the number of elements in the library.

State Assignment

CALD requires N memory elements to represent 2^N states in the State Table. This configuration tends to generate an economical and compact design. It is necessary to assign an N bit binary code to each of the 2^N states of the State Table. The complexity and cost of the combinational logic are in part determined by the code assigned to the states. The problem of state assignment, so as at completely minimize the cost of the combinational logic, is unsolved. One state assignment procedure that generally reduces cost involves minimizing the total number of flip-flop transitions that occur as the control logic sequences through the entire State Table. This is the approach taken in the CALD programs. In addition, the State Assignment routine optionally allows the user to assign the states and input the assignment on punched cards.

The program prints out the number of flip-flop transitions for a particular state assignment to allow the user to generate and evaluate many different state assignments.

The designer may specify either JK memory elements or Delay memory elements to be used in the logic design. The type of memory element is entered in response to a question printed on the teletype.

Memory Element Input Equations

The memory element input equations (application equations) determine which memory elements must change state at each clock pulse. These equations are calculated for either JK or Delay elements depending on which type was specified by the User. This routine examines each row of the State Table to determine whether a term should be added to any of the memory element input equations. The determination is based on the binary code assigned to the present state and the next state and the type of memory element. The input equations are treated like the Boolean Conditions in the two input tables. The equations are scanned for subexpressions. The subexpressions are given names, arranged in canonical form and stored in either the AND or OR table. One type of Boolean reduction is performed while calculating the input equations. The expression, composed of two terms, Si*Ij + Si*Ij is replaced by its logical equivalent, Si.

The user specifies the maximum allowed gate fan-in as any integer greater than one. This assignment is made in response to a question on the teletype.

Boolean Equation Reduction and Factoring

The Boolean Equation Reduction subroutine

searches both the AND and OR tables for equations that are subexpressions of larger equations. The subexpression in the larger equation is then removed and replaced by the name of the smaller equation. The search is facilitated by the fact that the equations are stored in canonical form.

The Boolean Equation Factoring subroutine factors the equations in both the AND and OR tables to ensure that the length of each equation is less than or equal to the maximum allowed fan-in. The factoring procedure involves choosing a factor, assigning a name to the factor, inserting the factor and its name into the AND or OR table, and then replacing the factor by its name in the equation under consideration. In order to minimize the number of gates in the final design it is desirable to choose factors that are common to more than one equation if at all possible.

Consider the equation below:

$$K1 = S2 + S4 + C8 + C9 + E3$$

If the maximum fan-in is three, CALD will generate the following factor for K1 assuming there are no other equations in the table.

$$K1 = C9 + E3 + -F1$$
$$F1 = S2 + S4 + C8$$

F1 appears in equation K1 with a "-" sign to indicate that F1 must be inverted. Before chosing a factor for K1, the Factoring routine would normally check K1 against every equation in the OR table in search of other equations that have at least three terms in common with K1. F1 would then be chosen from terms that are common to other equations.

The CALD programs calculate the fan-out of each gate in the logic design and prints it. Due to the small amount of memory the programs perform no operations to reduce a large fan-out.

Translator

The Translator translates the logic equations in the AND and OR tables into HALSIM specification statements. Every input and output name in the specification statements includes a "+" or "-" sign at the end of the name to indicate signal polarity. The fan-out for each gate is printed next to the specification statement for the gate. Each gate is checked to see if its output is required in inverted form. In addition, the program generates specification statements for the state memory elements and for Binary to Octal Converters used to decode the states.

The list of specification statements generated by CALD constitute the detailed logic diagram for the control logic of the digital system. A simple example of a CALD generated logic design is shown in Figure 3.

Conclusion

The CALD system employs heuristic methods to provide a practical design environment capable of generating a reasonable economical design, in a short period of time on a small computer. The system is easy and convenient to use. All the information that is needed by the designer is printed as messages on the teletype. The messages request information, indicate options that are available, and provide operating instructions. In addition, the CALD system simplifies the evaluation of a logic design by printing the number of memory element transitions resulting from a particular state assignment and the number of gate legs required in the design.

The combination of FST, CALD and HALSIM provides a powerful, integrated design automation tool that can relieve the designer of many tedious and error prone tasks. A primary advantage of the design system is an increase in the speed of design. Better and more economical designs may also result due to the fact that many designs can be examined and evaluated.

REFERENCES

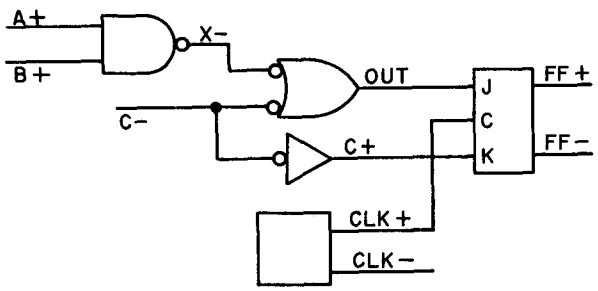
1. Bliss, F.W., "Computer-Aided Logic Design," Ph.D. Thesis, Case Western Reserve University, 1971.

2. Vlack, D., "Computer Simulation of Digital Systems on the Hardware Level," Ph.D. Thesis, Case Institute of Technology, 1967.

3. Mickelson, C.T., "A Revised Hardware Logic Simulator," M.S. Thesis, Case Western Reserve University, 1969.

4. Franke, E.A., "Automated Functional Design of Digital Systems," Ph.D. Thesis, Case Western Reserve University, 1967.

5. Franke, E.A. and H.W. Mergler, "Computer Aided Functional Design of Digital Systems," SEIEEE Record, pp. 13C1-13C4, April, 1968.




```
        ( ) FRMV  (CLK+ CLK-)
   (A+  B+) NAND  (X-)
   (X-  C-) NAND  (OUT)
      (C-) NAND  (C+)
(OUT CLK C+) JKFF  (FF+  FF-)
```

Figure 1

HALSIM Example

```
    Input  Fan Out  Input  Fan Out

     I01+     1      I01-     1


     *** LOGIC DIAGRAM IN HALSIN ***

( ) FRMV (CLK+ CLK-)
(K01+ CLK+ J01+) JKFF (FF1+ FF1-)
(FF1+ .0 .0 .0) BOCT (S01- S02- $ $ $ $ $ $)

                        3    1   0 0 0 0 0 0

(S02-) NAND (S02+)                  2 INVERTER
(S02+ I01+) NAND (C01-)             2
(S02+ I01+) NAND (C03-)             2

(S01-) NAND (J01+)                  1
(C01-) NAND (K01+)                  1

END1

        Gate legs 7
```

FIGURE 3

Input Fan Out Table and Logic Diagram in HALSIM

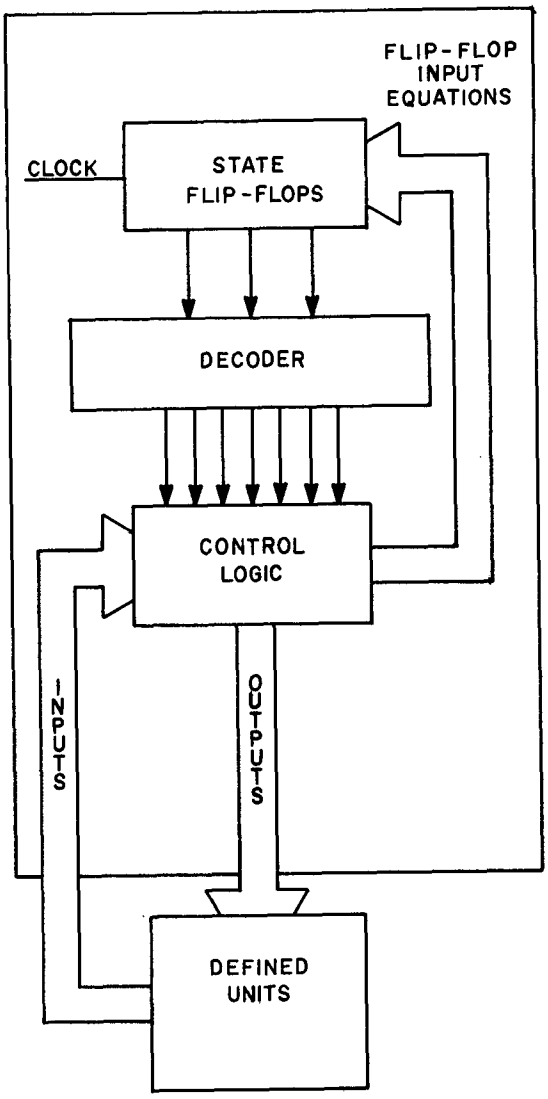


FIGURE 2

Block Diagram of Logic Design Generated by CALD