Computer Hardware Description Languages and their Applications

W.M. vanCleemput

Computer Systems Laboratory
Stanford University
Stanford, California 94305.

1

INTRODUCTION

Although design languages have been in existence since the early 1960's, only in the last few years has there been a concerted effort to bring them into the design process as a useful tool.

The major applications of design languages are:

1. Description of the behavior and/or structure of a system as a means for accurately communicating design details between designers and users.
2. As the input to a system level simulator.
3. As the input to a automatic hardware compiler.
4. As the input to a formal verification system.

Several attempts have been made to integrate hardware and software design. Examples are the LOGOS system [Ro76] developed at Case Western Reserve University and the SARA system developed at the University of California at Los Angeles [Es77].

2

THE HARDWARE DESIGN PROCESS

2.1 INTRODUCTION

It is difficult to describe the hardware design process formally since it depends to a large extent on the individual designer and on the specific design problem to be solved. Starting from a set of sometimes vague and incomplete specifications, the designer applies a series of successive transformations (iterative improvements) until the system can be realized (built) within a given technological environment or until it is clear that the specifications are not feasible.

It lies in the nature of the human intellect that for all but the most trivial designs, it is impossible to create a final design at once. Rather, a human designer tries to break down the problem into a number of interconnected subproblems. This process is them repeated until a solution to all the subproblems is known or until a well-known procedure can be applied to solve these subproblems.

Although formal methods do exist for solving certain problems, such as the minimization of combinational circuits or the state assignment for sequential circuits, many designers base their designs on a "library" of examples. These examples may originate from previous design experience, from the literature or from classroom exposure.

Hardware design has been greatly influenced by the development technology. The advent of MSI and LSI building blocks has magnified the tendency of doing hardware design 'by example'. In many instances, it is no longer clear how one should optimize a circuit containing medium or large-scale building blocks. For large systems the design is usually partitioned along functional lines among several designers. Since most designs are based on intuitive concepts such as experience, some design alternatives may be overlooked.

In the design of digital hardware several levels of abstraction have become universally recognized [see e.g. [BN71, La74]]. We will discuss each of these levels briefly:

2.2 ARCHITECTURAL LEVEL

At the architectural level, the designer is concerned with the overall structure of the system. He is interested in components such as processors, memories, I/O devices and in their interconnections in as far as such a configuration seems likely to satisfy the specifications. Each of the components has certain quantitative attributes. E.g. a memory has its size, wordlength and access time as its attributes. One attempt to describe digital systems on the system level is the PMS notation, introduced by Bell and Newell [BN70, BN71]. Most designers resort to the use of general purpose languages such as Fortran, Algol, Pascal or PL/1 to describe systems behavior at the architectural level. More specialized languages for simulation are SIMSCRIPT, GPSS[Sc74] and Simula[BD74].

2.3 REGISTER LEVEL

At the REGISTER TRANSFER LEVEL, the designer is concerned with realizing the functional specifications by sequences of operations. These operations are usually specified as transfers of information between the different facilities established in the functional design step. If we consider a digital system as a large finite state machine, then the purpose of the register-transfer level design is to establish the various states as well as the particular actions to be taken when the system is in a given state. At this level of the design process, the designer can form himself a reasonably adequate picture of the overall system and its performance, without being sidetracked by design details. The register level is the one at which most computer hardware description languages are used. Some of the best-known languages are: CDL [Ch65], DDL [DD68], ISP [BN70], AHPL [Hi74] and Cassandre [Bo71].

2.4 LOGIC DESIGN LEVEL

At the LOGIC DESIGN LEVEL, one is concerned with the mapping of the microoperations and the control structure, defined in the previous step, into physical hardware elements. This requires a detailed knowledge of the technology in which the design is to be implemented. The result of this phase may be a set of logic diagrams or Boolean equations, with the primitives being those available in the actual technology. At this level a design description is often in the form of a connectivity list (structural information).

2.5 INTENDED BEHAVIOR VERSUS ACTUAL IMPLEMENTATION

The initial specification of a design includes a sometimes vague and incomplete statement of how the actual system will behave. Such a specification is often in the form of a natural language. Besides specifying the intended behavior of a system, a designer may also specify implementation details, i.e. how smaller units can be connected in order to form a larger functional unit. Throughout the design cycle, two essentially different kinds of information can be captured: intended behavior of a system and actual implementation details. The former requires a behavior description language while the latter can be best expressed by a structure description language.

Ideally, behavior and structure information should be expressable using a single language. In principle, such a unified representation is possible because a simple function call in a behavior description, in which all the arguments are signals, corresponds to a structure description as well as to a functional description.

```
OUTPUTS = FUNCTION (INPUTS);
FUNCTION (INPUTS, OUTPUTS);
```

In practice, however, such unified notation has not been used.

3

BEHAVIOR DESCRIPTION LANGUAGES

3.1 INTRODUCTION

In the following sections, the major features of some of the most widely available languages will be illustrated. The major reason for selecting ISP, DDL and CDL was the availability of a compiler/simulator to the author.

3.2 CDL

The CDL language was proposed by Chu [Ch65] and subsets of the language were implemented by many groups. CDL was also used as an educational tool in two text books [Ch70, Ch72].

The following example describes a simplified 12-bit architecture with a straightforward instruction set. Note the emphsis on implementation details such a clocks, decoders and switches.

```
C DECLARATION SECTION
C  MDR = MEMORY DATA REGISTER
C  MAR = MEMORY ADDRESS REGISTER
C  ACC = ACCUMULATOR
C   PC = PROGRAM COUNTER
C  STATE = STATE REGISTER
C  RUNFF = RUN FLIPFLOP
C   INIT = INITIAL STATE FLIPFLOP
C      X = DUMMY REGISTER (ARGUMENT
C                        FOR OPERATORS)
 REGISTER,MDR(0-11),ACC(0-11),MAR(0-6),PC(0-6)
 REGISTER, X(0-11),INIT,STATE(0-3),RUNFF
C   THE SUBREGISTER DECLARATIONS ALLOW THE
C   RENAMING OF PARTS OF ALREADY DECLARED
C   REGISTERS.
 SUBREGISTER,MDR(OP)=MDR(0-3),MDR(I)=MDR(4)
 SUBREGISTER, MDR(ADDR)=MDR(5-11)
```

Figure 1: Register Declarations in CDL

```
C   THE MEMORY IS ADDRESSED BY MEANS OF THE
C       MAR REGISTER.
 MEMORY,M(MAR)=M(0-63,0-11)
C   THE DECODER HAS A SINGLE ACTIVE OUTPUT FOR
C   EACH OF THE VALID STATES.
 DECODER,K(0-9)=STATE
C   THE SWITCHES ALLOW FOR EXTERNAL EVENTS
C    TO BE MODELLED.
C   E.G. AN INTERRUPT CAN BE MODELLED THIS WAY.
C   E.G. AN INTERRUPT COULD BE MODELLED THIS WAY.
 SWITCH,POWER(ON),START(ON),STOP(ON)
C   IN GENERAL TERMINALS REPRESENT COMBINATIONAL
C   NETWORKS. IN THIS EXAMPLE, THE CONSTRUCT IS
C   USED TO RENAME THE OUTPUTS OF THE DECODER.
 TERMINAL,ADDK=K(0),SUBK=K(1),JOM=K(2),STO=K(3)
 ,JMP=K(4),SHR=K(5),CIL=K(6),CLA=K(7),STP=K(8)
 ,FETCH=K(9)
C   THE CLOCK HAS 3 PHASES: 0, 1 AND 2.
 CLOCK,P(2)
```

Figure 2: Other Declarations in CDL

```
C   THE FOLLOWING LINES DESCRIBE ACTIONS TAKEN
C   WHEN AN EXTERNAL EVENT OCCURS.
 /STOP(ON)/ RUNFF=0
 /POWER(ON)/RUNFF=0,STATE=8
 /START(ON)/INIT=1
C   MACHINE INITIALIZATION
 /INIT*P(2)/STATE=9,INIT=0,RUNFF=1
C   THE INSTRUCTION FETCH IS DESCRIBED HERE.
 /FETCH*P(0)/MAR=PC, IF(RUNFF.EQ.0) THEN
                             (STATE=8)
 /FETCH*P(1)/ MDR=M(MAR), PC= PC.COUNT.
 /FETCH*P(2)/ STATE=MDR(OP),MAR=MDR(ADDR)
```

Figure 3: Instruction Set in CDL, part 1

```
C   ADD INSTRUCTION
 /ADDK*P(1)/ MDR=M(MAR)
 /ADDK*P(2)/ ACC=ACC .ADD. MDR, STATE=9
C   SUBTRACT INSTRUCTION.
 /SUBK*P(1)/ MDR=M(MAR)
 /SUBK*P(2)/ ACC=ACC.SUB.MDR,STATE=9
C   STORE INSTRUCTION.
 /STO*P(0)/ MDR=ACC
 /STO*P(2)/ M(MAR)=MDR,STATE=9
C   CLEAR AND LOAD ACCUMULATOR.
 /CLA*P(1)/ MDR=M(MAR),ACC=0
 /CLA*P(2)/ ACC=ACC.ADD.MDR,STATE=9
```

Figure 4: Instruction Set in CDL, part 2

```
C   STOP INSTRUCTION.
 /STP*P(0)/ RUNFF=0
 /STP*P(2)/ IF(RUNFF.EQ.0)THEN (MAR=0,PC=0)
                          ELSE (STATE=9)
C   JUMP INSTRUCTION.
 /JMP*P(2)/ PC=MDR(ADDR),STATE=9
C   JUMP ON MINUS INSTRUCTION.
 /JOM*P(2)/ IF(ACC(0)) THEN (PC=MDR(ADDR)),
                                 STATE=9
C   SHIFT RIGHT.
 /SHR*P(2)/ ACC= ACC.SHR.,STATE=9
C   ROTATE  LEFT.
 /CIL*P(2)/ ACC=ACC.CIL.,STATE=9
```

Figure 5: Instruction Set in CDL, part 3

```
C    TEST PROGRAM.
C  LOCN   OPCODE OPND
C      0     CLA     10
C      1     SUB     11
C      2     JOM      4
C      3     STP
C      4     ADD     12
C      5     SHR
C      6     CIL
C      7     STP
C     10     =5
C     11     =10
C     12     =2
 END
```

Figure 6: Test Case for CDL


```
*OPERATOR,  X(0-11).SHR.
  //  0-X(0-10), RETURN
 END
*OPERATOR,  X(0-11).CIL.
 // X(1-11)-X(0), RETURN
 END
$SIMULATE
*OUTPUT   LABEL(1)=ACC,MDR,MAR,PC,STATE,RUNFF
*LOAD
 ACC=0,MAR=0,PC=0,RUNFF=1
 M(0-)=:70A,:10B,:204,:800,:00C,:500,:600,:800
 M(10-)=:005,:00A,:002
*SWITCH    1,POWER=ON
*SWITCH    2,START=ON
*SIM       100,10
```

Figure 7: Simulator Control Language for CDL

3.3 DDL

DDL (Digital Design Language) was first formulated by Duley and Dietmeyer [DD68]. A Fortran-based implementation was done at the University of Wisconsin [Di74]. An Algol-based version was implemented by Duley at Hewlett Packard. A Pascal version, based on the Algol version, was implemented at Stanford University [CD79a, CD79b]. The example that follows is for the latter version of DDL, which deviates to some extent from the original version.

One of the major characteristics of DDL is its assumption that a sequential design is to be considered as a finite state machine. Again, this description emphasizes some degree of implementation detail.

The example as shown here is similar to the ISP example that follows in the next section.

```
OPERATION
   INCRPC = [PC <-  PC (+) 1 TAIL 8],
   GETINST = [IR <- M[PC]],
   DIRADD = [Z <- IR[7:0]],
   INDADD = [Z <- M[IR[7:0],7:0]],
   ANDOP = [ACC <- ACC * M[Z]],
   TADOP = [CARRY CON ACC <- ACC (+) M[Z]],
   INCMEM = [M[Z] <-  M[Z] (+) 1 TAIL 12],
   DCAOP = [M[Z] <- ACC],
   CLRACC = [ACC <- 12B0],
   SETRET = [L <- PC],
   SETJMP = [PC <- Z],
   IOTOP = [IOREG <- IR[7:0]],
   COMPLA = [ACC <- - ACC],
   INCACC = [CARRY CON ACC <- ACC (+) 12B1],
   SUBACC = [CARRY CON ACC <- ACC (-) 12B1],
   ACCSL1 = [ACC <- ACC[10:0] CON 1B0] ,
   ACCSR1 = [ACC <- 1B0 CON ACC[11:1]],
   RETRN = [PC <- L],
   JMPOP = [PC <- ACC[7:0]],
   CLRRUN = [RUNFF <- 1B0],
   SETRUN = [RUNFF <- 1B1]
  END
```

Figure 9: Operation Declaration in DDL

```
" SIMPLE MINICOMPUTER . "
 REGISTER Z[7:0], ACC[12:0], IR[11:0], CARRY,
       L[7:0],  PC[7:0],  IOREG[7:0], RUNFF,
       M[0:31,11:0].
```

Figure 8: Register Declarations in DDL

```
CONTROL
   WAIT: IF (RUNFF (=) 0) THEN -> WAIT ENDIF/
   IFETCH: IF RUNFF THEN GETINST, INCRPC
                    ELSE -> WAIT    ENDIF/
   EFFADD: IF IR[8] THEN INDADD ELSE DIRADD
                                       ENDIF/
 EXEC: CASE IR[11:9]
       DO TADOP
       DO INCMEM, IF (M[Z] (=) 0) THEN INCRPC
                                          ENDIF
       DO DCAOP, CLRACC
       DO SETRET, SETJMP
       DO SETJMP
       DO IOTOP
       DO CASE IR[7]
          DO IF IR[6] THEN INCRPC ENDIF,
             IF IR[5] THEN RETRN ENDIF,
             IF IR[4] THEN JMPOP ENDIF,
             IF IR[3] THEN CLRRUN ENDIF,
             IF (IR[2]*ACC[11])
             + (IR[1]*(ACC (=) 0 ))
             + (IR[0]*-ACC[11])
              THEN INCRPC ENDIF
          DO IF IR[6] THEN INCRPC ENDIF,
             IF IR[5] THEN COMPLA ENDIF,
             IF IR[4] THEN CLRACC ENDIF,
             IF IR[3] THEN INCACC ENDIF,
             IF IR[2] THEN SUBACC ENDIF,
             IF IR[1] THEN ACCSR1 ENDIF,
             IF IR[0] THEN ACCSL1 ENDIF
       ENDCASE
       DO ANDOP
              ENDCASE,
             -> IFETCH/
 END
```

Figure 10: Control Section Declaration in DDL

3.4 ISP

ISP was originally formulated by Bell and Newell [BN70] and used as a description tool in a text on computer architecture [BN71]. A compiler and simulator were implemented by Barbacci [Ba77, Si74]. ISP is now the basis for an extensive register-transfer-level design automation system at Carnegie Mellon University [BS75].

The example in this section illustrates the ISPL version of the ISP language and was adapted from [Ba77]. This example describes the same 12-bit machine as in the previous section. Note that a lot of implementation detail is no longer visible. The intention of ISP is to be able to hide implementation details, thereby concentrating on the pure behavioral description.

```
MINI:=(DECLARE  !MEMORY AND REGISTERS
        M[0:#377]<11:0>;          !MAIN MEMORY
        Z<7:0>; !EFFECTIVE ADDRESS REGISTER
        CACC<12:0>; !13 BIT ACCUMULATOR + CARRY
            ACC<11:0> := CACC<11:0>;
        IR<11:0>;         !INSTRUCTION REGISTER
        L<7:0>; !RETURN REGISTER
        PC<7:0>;          !PROGRAM COUNTER
        IO.REG<7:0>;      !INPUT-OUTPUT REGISTER
        RUN<>;  !RUN MODE
    ! PROCEDURE TO INCREMENT PROGRAM COUNTER
    INCRPC:=( PC<-(PC+1)<7:0>)
  ERALCED
```

Figure 11: Declarations in ISP

```
START:= (DECODE RUN =>
        STOP;             ! IF RUN=0
        (IR<-M[PC] NEXT INCRPC NEXT
         (DECODE IR<8> => Z<-IR ;
                      Z<-M[IR<7:0>]<7:0>) NEXT
         (DECODE OP =>   !INSTRUCTION DECODING
            ACC<-ACC AND M[Z];        !AND
            CACC<-ACC + M[Z]; !TAD (SETS CARRY)
            (M[Z]<-(M[Z]+1)<11:0> NEXT
             (IF M[Z] EQL 0 => INCRPC)); !ISZ
            (M[Z]<-ACC NEXT ACC<-0);  !DCA
            (L<-PC NEXT PC<-Z); !JSR
            PC<-Z; !JUMP
            IO.REG<-IR<7:0>; !IOT
            (DECODE IR<7> =>
          ((IF IR<6> => INCRPC) NEXT
            (IF IR<5> => ACC<- NOT ACC) NEXT
            (IF IR<4> => ACC<-0) NEXT
            (IF IR<3> => CACC<-ACC+1) NEXT
            (IF IR<2> => CACC<-ACC-1) NEXT
            (IF IR<1> => ACC<- ACC $SR0 1) NEXT
            (IF IR<0> => ACC<- ACC $SL0 1));
                 !END OF UCLASS=0
          ((IF IR<6> => INCRPC) NEXT
              (IF IR<5> => PC<-L) NEXT
              (IF IR<4> => PC<-CACC<7:0>) NEXT
              (IF IR<3> => RUN<-0) NEXT
              (IF (IR<2> AND CACC<11>) OR
                  (IR<1> AND (ACC EQL 0)) OR
                  (IR<0> AND (NOT CACC<11>))
                          => INCRPC)
          )
           ) !END OF IR<7> DECODING
           ) !END OF INSTRUCTION DECODING
        ) !END OF RUN=1 MODE
        ) NEXT  !END OF INSTRUCTION CYCLE
    START
 )
```

Figure 12: Behavior Description in ISP

3.5 OTHER LANGUAGES

Among other languages that have been implemented and used are: Cassandre [Bo71], developed at the University of Grenoble, France and used mainly in Europe; AHPL, a derivative of APL [HP73, Hi74]; LCD, developed at IBM and RTS, developed at the University of Darmstadt, Germany.

An excellent survey of all major computer hardware description languages can be found in [Su74]. A bibliography on the subject can be found in [Va76, Va77, Va78].

The proliferation of Computer Hardware Design Languages has been of concern to a large number of people in the field. A Conference on Computer Hardware Description Languages, headed by J. Lipovski, has been active in finding a solution to the problem of multiple languages. Recently, a subcommittee, headed by R. Piloty, compiled a preliminary version of a Consensus Language [CONLAN].

4

STRUCTURE DESCRIPTION LANGUAGES

4.1 INTRODUCTION

As pointed out before, a structural description of a digital system is useful in the early stages of the design process (architectural design) and in the later, physical implementation phases.

In this section we will illustrate these concepts by describing a system in two ways: first using the PMS notation [BN70], and second, using the SDL notation [Va77a]. Another language for representing structural information is the SL/1 notation developed by Gardner [Ga74]. It must be pointed out that the best representation of structural information is not in the form of a one-dimensional language but in the form of a two-dimensional graphic representation. Such a computer-generated graphic representation was used in the SCALD system [MW78]. The following example illustrates the use of SDL for describing computer systems at the high (architectural) level. Figure 13 shows the PMS diagram for a simple PDP-8 computer system. Figure 14 shows the schematic diagram corresponding to the PMS description of Figure 13 and to the SDL description of Figure 15.

```
                 T.console
                     |
  Mp<0>--S---L--Pc--L(IObus)--S--K-
             |       |         |
             L       |--L(Data L
             |       |    Break) |
  Mp<1>--S-|         |         |-S--K-
             |                 |
             L                 L
             |                 |
  Mp<2>--S-|                   |-S--K-
             |                 |

     Memory Bus          I/O Bus
```

Figure 13: PMS Diagram for a simple DEC PDP-8 System

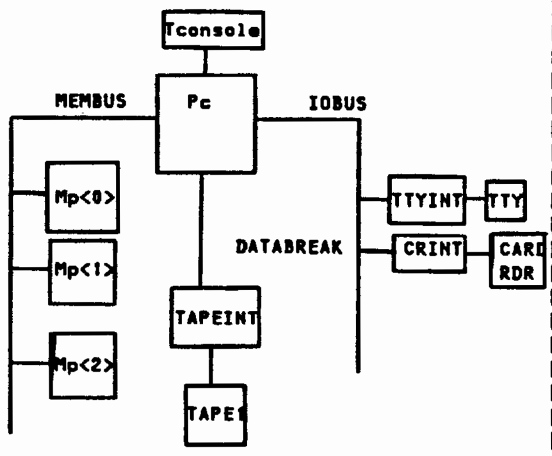

Figure 14: PDP-8 System Structure

```
NAME: DEC_PDP_8;
EXT:: IOBUS, DATABREAK;
TYPES: MEM4K (WL="12", TC="1.5us"),
       CPU8 (TYPE="PDP-8/S"),
       ASR33,
       TM8 (* Tapedrive *),
       TN8 (* Tape Interface *),
       KL8 (* TTY Interface *),
       KK8 (* card reader interface *),
       CR8 (* Card Reader *);
COMPSEGMENT;
Pc = CPU8 (MEMBUS, CONSLINK, IOBUS, DATABREAK);
Mp<0> = MEM4K (MEMBUS);
Mp<1> = MEM4K (MEMBUS);
Mp<2> = MEM4K (MEMBUS);
Tconsole = ASR33 (CONSLINK);
TAPEINT = TN8 (DATABREAK, TB);
TAPE1 = TM8 (TB);
TTYINT = KL8 (IOBUS, TBUS);
CRINT = KK8 (IOBUS, OBUS);
TTY = ASR33 (TBUS);
CARDREADER = CR8 (OBUS);
END;
```

Figure 15: SDL Description of the DEC PDP-8 System

5

APPLICATIONS OF HARDWARE DESCRIPTION LANGUAGES

5.1 DOCUMENTATION

A very important application of hardware description languages is the description of a system's behaviour and/or structure for the purpose of accurate communication between designers and users. A prime example of languages originally intended for this purpose are the PMS and ISP notations proposed by Bell and Newell [BN70, BN71, Si74a, Si74b]. The PMS notation was intended for the description of the physcial structure of hardware at the system level while ISP was intended for describing the behaviour of a system at the instruction set level. An interesting application of a structural description language is the PMSL system [Kn73], in which a system is described in PMS notation and where tools are provided for analyzing the performance of this system. A particular application of digital design languages as a descriptive tool is in teaching hardware design and computer architecture. Among the textbooks that use a digitel design language for this purpose are Chu [Ch70, Ch72], Dietmeyer [Di71], Hill and Petersen [HP73] and Bell and Newell [BN71].

5.2 AUTOMATIC HARDWARE GENERATION

A potentially important application of hardware description languages is as the input to a hardware compiler that automatically translates the high-level language description into a logic design. This seems extremely useful since together with a register-transfer-level simulator it would allow rapid and accurate hardware design. However several problems do exist with this approach:

1. The inability of most of today's hardware design languages to describe the hardware accurately enough in order to satisfy a designer.

2. The ever-changing characteristics and complexity of the hardware primitives in which a design is to be mapped.

Examples of such automated hardware compilers are the ALERT system [FY69], DDL [DD68] and the Carnegie-Mellon RT-CAD system [BS75].

5.3 SIMULATION

Simulation is a widely used tool for partially validating a design at almost any level of the design process. At the system and functional level, the behaviour is frequently simulated using general-purpose simulation languages such as GPSS [Sc74], SIMSCRIPT or SIMULA [BD74].

Many hardware description languages can be used as an input language to a simulator, usually at the register-transfer level. Examples of such languages are CDL [Ch65, Ch74], DDL [DD68, Di74], ISP [Ba77] and Cassandre [Bo71]. These simulators can be used to verify the flow of data and the functional behaviour of a system. The effectiveness of such a simulation depends on the descriptive power and accuracy of the associated design language.

Once the logic design is completed, one can make use of a gate-level simulator to further verify the system. Gate-level simulation can also be used to verify test sequences and to study the influence of faults on the system. For gate-level simulation all components are reduced to simple gates and possibly delays, together with their interconnections (structural description).

6

THE FUTURE

A large number of hardware designs is intended for a small-scale production, often a single copy. Furthermore, most systems do not require a performance that approaches the limits that are the state-of-the-art. In these cases the design cost far exceeds the cost of the actual hardware and therefore a less efficient hardware implementation could be tolerated if this would lead to a reduction in design cost. In such an environment, a design language with an associated to multi-level simulator and a hardware mapping facility can be very useful. Since the choice of the technology is the task of the designer, a good design system would allow him to specify this mapping if he chooses to. This would certainly make the system more independent from changes in hardware technology [Va77a].

Although some efforts have been make to prove programs and hardware designs correct, for large programs or systems this approach is not likely to be succesful because of the inherent complexity of proof of correctness procedures. A more likely solution to the design validation problem in general is the development of programming languages and hardware design techniques that will allow proving correctness of a design efficiently.

REFERENCES

[Ba77] Barbacci,M. "The ISPL Language," Carnegie-Mellon Univ., Dept. of Computer Science, 1977.

[BS75] Barbacci,M. and Siewiorek,D.P. "The CMU RT-CAD System: an Innovative Approach to Computer-Aided Design," Carnegie-Mellon University, Computer Science Review, Sept. 1975, pp. 39-53.

[BN70] Bell,C.G. and Newell,A. "The PMS and ISP Descriptive Systems for Computer Structures," Proc. Spring Joint Computer Conference, 1970, pp. 351-374.

[BN71] Bell,C.G. and Newell,A. "Computer Structures: Readings and Examples," New York: McGraw Hill, 1971.

[BD74] Birtwistle,G.M.; Dahl,O.J.; Myhrhaug,B. and Nygaard,K. "SIMULA Begin," New York: Petrocelli Books, 1974.

[Bo71] Bogo,G et al. "Cassandre and the Computer-Aided Logical Systems Design," Proc. IFIP Congress, Ljubljana, Yugoslavia, Aug. 1971, pp. 1056-1065.

[Br72a] Breuer,M.A. "Recent Developments in the Automated Design and Analysis of Digital Systems," Proc. IEEE, Vol. 60, no. 1, pp. 12-27, Jan. 1972.

[Br72b] Breuer,M.A. [ed.] "Design Automation of Digital Systems, vol. 1: Theory and Techniques," Englewood Cliffs,N.J. : Prentice Hall, 1972.

[Br75] Breuer,M.A. [ed.] "Digital System Design Automation: Languages, Simulation and Data Base," Woodland Hills, Cal.: Computer Science Press, 1975.

[Ch65] Chu,C.Y. "An ALGOL-like Computer Design Language," Comm. ACM, vol. 8, no. 10, pp. 607-615, Oct. 1965

[Ch70] Chu,Y. "Introduction to Computer Organization," Englewood Cliffs,N.J. : Prentice Hall, 1970.

[Ch72] Chu,Y. "Computer Organization and Microprogramming," Englewood Cliffs, N.J. : Prentice Hall, 1972.

[Ch74] Chu,Y. "Introducing CDL," IEEE Computer, vol. 7, no. 12, pp. 42-44, Dec. 1974.

[CD79a] Cory,W; Duley,J. and vanCleemput,W. "DDL-P Language Manual" Stanford Univ.,Computer Systems Lab., Technical Report, 1979.

[CD79b] Cory,W; Duley,J. and vanCleemput,W. "DDL-P Command Language Manual" Stanford Univ.,Computer Systems Lab., Technical Report, 1979.

[Di71] Dietmeyer,D.L. "Logic Design of Digital Systems," Boston: Allyn and Bacon, 1971.

[Di74] Dietmeyer,D.L. "Introducing DDL," IEEE Computer, vol. 7, no. 12, pp. 34-38, Dec. 1974.

[DD68] Duley,J.R. and Dietmeyer,D.L. "A Digital System Design Language," IEEE Trans. Computers, vol. C-17, no. 9, pp. 850-861, Sept. 1968.

[Es77] Estrin,G. "Modelling for Synthesis, the Gap between Intent and Behavior," Proc. Symp. on Microprocessors and Design Automation, Palo Alto, Cal., Feb. 1977, pp. 54-59.

[FY69] Friedman,T.D. and Yang,S.C. "Methods used in an Automatic Logic Design Generator [ALERT]," IEEE Trans. Computers, vol. C-18, no. 7, pp. 593-614, July 1969.

[Ga74] Gardner,R "," Univ. of California, Los Angeles, Dept. of Computer Science, Ph.D. Thesis, 1974.

[HP73] Hill,F.J. and Peterson,G.R. "Digital Systems: Hardware Organization and Design," New York, Wiley, 1973.

[Hi74] Hill,F.J. "Introducing AHPL," IEEE Computer, vol. 7, no. 12, pp. 28-30, Dec. 1974.

[Kn73] Knudsen,M.J. "PMSL, An Interactive Language for System Level Description and Analysis of Computer Structures," Carnegie-Mellon Univ., Ph.D. Thesis, April 1973.

[Ma70] Macdougall,M.H. "Computer System Simulation: an Introduction," Computing Surveys, vol. 2, no. 3, Sept. 1970.

[MW78] McWilliams,T. and Widdoes,L.C. "SCALD, Structured Computer-Aided Logic Design," Proc. 15th Design Automation Conf., Las Vegas, June 1978.

[Ro76] Rose,C.W. "Design Systems: a Five-Year View,", Digest COMPCON 1976 Spring, PP. 190-193.

[Ru76] Russo,R.L. "Methods of Verification in Design Automation," IEEE Computer, vol. 9, no. 4, pp. 54-55, April 1976.

[Sc74] Schriber,T.J. "Simulation using GPSS," New York: Wiley, 1974.

[Si74a] Siewiorek,D.P. "Introducing ISP," IEEE Computer, vol. 7, no. 12, pp. 39-41, Dec. 1974.

[Si74b] Siewiorek,D.P. "Introducing PMS," IEEE Computer, vol. 7, no. 12, pp. 42-44, Dec. 1974.

[Su74] Su,S.Y.H. "A Survey of Computer Hardware Description Languages in the U.S.A.," IEEE Computer, vol. 7, no. 12, pp. 45-51, Dec. 1974.

[Va76] vanCleemput,W.M. "Computer-Aided Design of Digital Systems: a Bibliography," Woodland Hills, Cal.: Computer Science Press, 1976.

[Va77] vanCleemput,W.M. "Computer-Aided Design of Digital Systems: a Bibliography, volume 2: 1975-76," Woodland Hills, Cal.: Computer Science Press, 1977.

[Va77a] vanCleemput,W.M. "A Hierarchical Language for the Structural Description of of Digital Systems," Proc. 14th Design Automation Conf., New Orleans, June 1977, pp. 378-385.

[Va78] vanCleemput,W.M. "Computer-Aided Design of Digital Systems: a Bibliography, volume 3: 1976-1977," Woodland Hills, Cal.: Computer Science Press, 1978.