1979

The C-MU Design Automation system: an example of automated data path design

Alice C. Parker
Carnegie Mellon University

Donald E. Thomas

Daniel P. Siewiorek

Mario R. Barbacci

THE C-MU DESIGN AUTOMATION SYSTEM

AN EXAMPLE OF AUTOMATED DATA PATH DESIGN

by

A. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer,
G. Leive, J. Kim

DRC-18-15-79
May 1979

The CMU Design Automation System

An Example of Automated Data Path Design

A. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive, J. Kim

Carnegie-Mellon University
Departments of Electrical Engineering and Computer Science
Pittsburgh, Pennsylvania 15213

Abstract

This paper illustrates the methodology of the CMU Design Automation System by presenting an automated design of the PDP-8/E data paths from a functional description. This automated design (using synthesis techniques) is compared both to DEC's implementation and the Intersil single chip implementation.

1. Introduction

As it is becoming possible to integrate larger numbers of logic components on a single chip, the need for more powerful design aids is becoming apparent. Indeed, these aids must be capable of supporting a designer from the system level of design down to the mask level. In this way the systems level designer can become more aware of the implications of higher-level design tradeoffs on implementation properties such as silicon area, power consumption, testability, and speed, and be able to make more timely use of new technologies. The ultimate goal of the Carnegie-Mellon University Design Automation (CMU-DA) System [12] is to provide a technology-relative, structured-design aid to help the hardware designer explore a larger number of possible design implementations. Inputs to the system are a behavioral description of the system to be designed, an objective function which specifies the user's optimization criteria, and a data base specifying the hardware components available to the design system.

The CMU-DA system differs from other design automation systems because the input design description is a functional specification. Such a specification provides a model that, while accurately characterizing the input-output behavior desired for the implementation, does not necessarily specify its internal structure. The system software collectively performs the synthesis function by transforming the input functional description into a structural description. The design process involves binding implementation decisions in a top-down manner as a design proceeds through the design system. More structural decisions are made at each level until a complete hardware specification is obtained, with the most influential design trade offs being performed first in order to cut down the design search space.

The purpose of this paper is to illustrate the methodology of the CMU-DA system. The results given here are worst case - many optimizations which are straightforward have not been implemented yet; research is in progress on others. The design of the data part of a DEC PDP-8/E [5] from the ISP level through to a TTL and standard cell design will be discussed. Only the subset of the full DA system which is presently implemented has been used for this example. The

This research is supported in part by NSF Grant MCS77-09730 and Army Research Office Grants DAAG29-76-G-0224 and DAAG29-78-G-0070.

components are shown in Figure J.

The *P0P-8/E* is first functionally described using the ISP language [1]. A data-memory allocator [7] is used to generate the structure? of the data paths from the functional description. This allocation is in lorms of abstract logic components. The next r.lcp in the design is to bind physical modules to the abstract design from a modulo database [9], This step provides I he system with the capability of designing relative to now technologies. This binding will be illustrated both in terms of TTI. chip* and the CMOS standard cells in the Sandia design system [10]. . The standard cell output of the CMU-DA system is then translated for input to the Sandia design system. At this point, chip area can be calculated and detailed timing information can be gathered using SALOGS [3].

The paper will conclude by comparing these two alternative designs to commercial implementations.

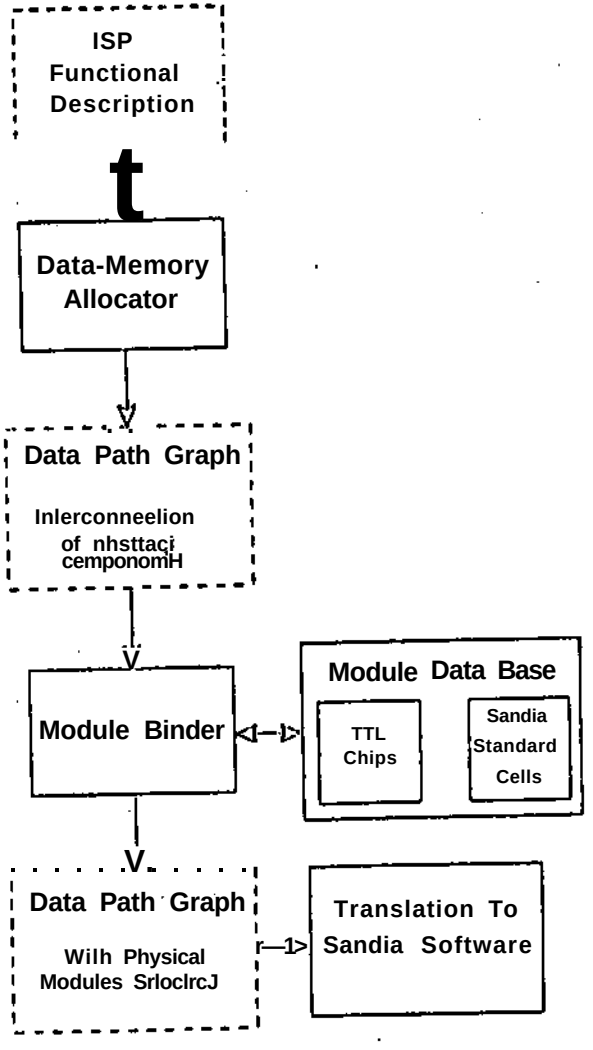


Figure* I: Tho CMU Design System

2. The PDP-8/E Example

Ac the CMU-DA syslc?m has evolved, more complex design example? have been used to observe its performance. The use of the PDP-S/F. is n major slop in two ways: 1) It represents about a five fold incroar.e in Ihe circuit complexity over previously reported example designs [8], and 2) It is a commercially available system that has been implemented in various technologies by different manufacturers over several years, fhur, it is possible lo compare a non-trivial automated design lo designs clone by several designers with different logic components.

The PDP-8/L ir, one model in a family of computers having nearly identical ISP descriptions. (That is, they implement nearly the same instruction sot, with different hardware structure*). A portion of the ISP description used by the automated design system is shown in Figure 2.

The description begins by declaring the memory and processor stale. A 13 bit link-accumulator (lac) register is defined but can alternately be accessed as the one bit link (L) and 12 bil accumulator (Ac). Next, instruction interpretation is defined using tho declared memory and registers. After the instruction fetch and increment of the program counter in the instruction interpretation section, instruction execution (CxCcO) i» called. Illustrated here are the six memory reference instruc ons. Not shown but implemented in iho automated designs are the effective address calculation, input/output instructions and tho operate microinstructions. Note that Ihn Mb=Mp[Pc] instruction fetches the location pointocl to by tho Pc and places it in the Mb register. No mention need be made of transfer of the current Pc to a memory addrnss register, which is a part of the hardware structure.

The ISP description [1] is compiled into a representation which is machine readable by Ihe data-memory allocator. The next sections will discuss the data-memory allocation (where an abstract data path i» synthesized), a module binder that illustrates how tho CMU-DA system can design relative to new technologies, and a translator to Sandia's SALOGS simulator [10].

3. The Allocation Process

The data-memory allocators perform a mapping function from thn algorithmic (ISP) description to the data-path part of the hardware implementation, which is called a data-path graph The data parl consists of Ihe data-storage elements, data operators, and data paths necessary to implement the operations specified in the algorithmic description. Due to the characteristics of Ihe ISP language this mapping may be multi-valued in either direclion, rather than a simple one-to-one translation.

The PDP-8/E design described here was produced by a data-memory allocator which uses the distribuled-logic design style. This .stylo of (or approach to) design encompasses design with small and medium scale integration components. As pointed out earlier, Ihe allocator itself is technology relative and the mapping onto .specific integrated circuit packager? is performed by a separate module binder program. The process referred to as allocation throughout Ihe remainder of this paper is a synthesis of logic using generic logic elements; data pathr., operators, registers, and multiplexers, all of any bil width.

The procedurn used by Ihe allocator might be compared to a two-pass compilation. The first pass may be considered a syntax or feasibility chock The allocator inputs a parsed ISP description, con&k'.idr. data structures analogous in function to symbol tables, *?UM* enforces constraints necessary to insure that the data-storage locations, logical mappings, and

```
pdpfl :=
    BEGIN

! Tho h.*r«lc PDP-R Instruction sol (ulthoul oxtondnd arithmetic
! elomont) IK Imp Intimntnd.  No I/O dovlcor. are includncl In tho
! doncr ipl lon.

! rip Stalo

    Mp[#0!^77771<0:11>,                    ! Main memory
    Mb<0:                                  ! Memory buf for

! PC StAta

    lac<0:12>,                             ! Link- AC register
        L<>          := lac<0>,            ! LInK bit
        nc<0:iJ>     J= ltic<l:12>,        ! accumulator
    Pc*0:ll>,                              ! Program counter
    ! ASt.pc«0}:1>,

! Instruction Ihlorprnt.it lon

    6tnrt in
        RIIGIN
        fjn « 1 NCXT
        run()
        F.ND,

    run\Instruetlon.IntorprotAtlon :«
        BEGIN
        IF t|O r>
            DFGIN
            Mb r UpCPc] j l.v.l. pc «= Pc NEXT
            Pc = Pr i J. Nf.Xl
            cxecO NtXT
            IF intnrrupi. onab In HNO Intorrup t. roquos t =>
                ncniN
                lip CO! « Pc NEXT
                Pc * J
                FND NEXT
            RE5THRT run
            l.NO
        END,
```

```
! Instruel lon Exocutlon

    ox.oc\ Inr. true:t.loh.nvoctit lon tr
        REGIN
        Ir r Mb<0:2> NEXT
        IF (Ir GEO f$) HMO (Ir LEQ ^S) e> m^O NEXT
        IF Ir LEO n n> 11b r Hplm^O) NEXT
        DECODE Ir o
            BEGIN
            M \* Ahfl.jr He r flc HND Mb,          ! find
            tt\ m t.id := l,ir: Inc 4- Mb,         ! HDD (TO
            #2 l« Isr is RflGIN                    ! Incromont and
                                                   ! skip If zoro
                        Mb « Mb + 1 NEXT
                        IF fib EOL 0 o Pc r Pc f 1
                        r.NO,
             13 i« dca i« BEGIN                    ! Deposit and
                        Mb = Oc NEXT               ! clear Re
                        nc « 0
                        TNI),
            #1 i« Jmr, is nrGIM                    ! Jump to
                        Mb e Pc NEXT               ! subroutine
                        PC B |DM I
                        END,
            //!} != jinf) is Pc c VlAi             ! Jump
            ^6 m lolO,                             ! I/O expcutlon
            HI ir opr()                            ! Oporato
                                                   ! microInst.
            LNO Ni:XT
        IF (Ir GEQ #2) OND (Ir LEO 44) •> Mptma) - Hb
    .END
END         ! End of closer lp l lon
```

Figure 2: ISP Description of tho PDP-S/E

input/oulpiil inInfacn chnr.u loi ir»lin> spr'.ificci in .lhe description can ho implomnntnd in hnidwaro. If no errors nro encountered, il proceeds lo allocate the ba-.ic data-storage structures colled for in lho description, and any additional data paths, storage, and operators necessary to implement variablo-cicccssing schemes described by ISP. The second pass may be considered as the semantic phase, with the activity of r.qclo generation replaced by Ihe allocation of data paths, operators, and additional storage as needed to implement lho actions described in the ISP description. Parallelism analysis is performed at several levels to warn the user of error conditions and determine constraints relating to optimization of hardware. The allocation is then completed by the addition of multiplexing where required.

However, allocation differs from compilation in that in a compil.nlion one is concerned with implementing the specified data operations on a fixed data part whose capabilities are known o. *priori.* In allocation, the allocator must bo able to recall and utilize lho capabilities of a data part which is being dynamically created. The allocator thus works from the inside out, first creating the data storage and access structures, and then adding Ihe neces?«ary' data paths and operators to perform Ihe described data operations. Finally, the output of the allocator is a non-planor directed graph, rather than a linear list of compi'cd inductions.

The first version of the allocator is experimental, and it performs only minor optimizations on Ihe allocated hardware. It has been designed to investigate Ihe feasibility of performing the mapping from ISP lo hardware, the types of data structures needed for allocation, and areas where optimizations are possible in future, more sophisticated allocators.

Tho allocator has boon designed as a possible skeletal structure for future allocators in order to standardize input/output formats and data structures.

4. Performance of the Data-Memory Allocator

Thn allocator program was run using the ISP description of the PDf'-S/E and tho resultant data paths are shown in Figure 3. A binding of modules was clone by hand to compare the results of the allocator to the original DEC design (Figure 4) [5].

It is difficult to compare Ihe automated PDP-8/E data-path design with lho original DEC design (or three reasons. First, the ISP description input to tho allocator declares ar, registers some values lho PDP-8/E ur.es but never stores explicitly in registers, such as lho effective address. These show up as registers in lho allocator's design. Second, the allocator designs distributed logic, *and* Ihe DEC design was clone in the central-accumulator design style. (Th.it is, this allocator does not contain Ihe design rules for largo scale collapsing of Ihe data paths'into a central-accumulator stylo of design [13]). Third, the DEC design ha*\ assumed a boundary between Ihe control and data-memory parts of Ihe design, but the boundary is different from that imposed on the allocator by Ihe ISP description. Thus some tor.tr;, flags, and registers which must be declared explicitly in lho ISP description are part of the control in the DEC design.

The main reason for Ihe difference in design seen from Ihe block diagram lovol of Figures 3 and 4 is that the design styles are different. Thq multiplexing is used in different ways. In Ihe DEC version, tho operators nro shared, and are used lo provide no-op paths from one register to another. In Ihe CMU version, only rogislers nro shared and use multiplexed inpuis. The ISP language ir; pailinlly the source of this disparity. In ISP, the user can repeatedly use register A as a destination from various sources. However, Ihe expressions A-»B and C+D do not imply nor discount a single adder. Olher differences in the design include Ihe use of multiplexers for shifting iri the

DEC **<lo'.i|\n, iiiicl** u«f» u(li ur/< omplcjmnnl 0/1 chips for creating complements. "ONiniV[1] of the MQ and AC registers in the DEC version is CIMMO wilhin Iho multiplexing hardware. Constants are of Ion en Mod in ono place and gated ovor already, existent data paths to Ihr registers. In the CMU version, these constants are multiplexed at the register inputs.

In r.pitc of thor.o differences, estimates of chip count indieale thai tho allocator produces a path graph which would require 39/ moro integrated circuit chips that the DEC designers used for the dala paths and registers. These estimates wore done by hand lo gauge the performance of the allocator; an automatic binding i» discussed in the next section. These rslimale?; wore made Using tho same 1970 technology chip r-el the DEC designers had lo deal with. The 397. excess hardware can bo found in multiplexors which connect the registers tho extra registers declared in the ISP description, and duplicated operators like increment and add. Much of this excess can bo allribuiocl lo the lack of optimization capability in thr allocator algorithm, future, more sophisticated allocation al(jorilhmr«, coupled with the capability for high level optimization [I?] available in the complete CMU-DA system will be able to significantly improve the data part design.

Further analysis of Ihis design is in progress and includes a manual implementation of tho control port. Comparisons of the DEC and CMU data-path speeds will then bo possible.

5. Modulo Bindinc

The modulo binding phnr.o of tho CMU-DA system employs the Modulo Data Itasn System (MDH5) and follows the data-memory allocation Mop. It has the ta'.k of translating the abstract link';, memories, registers, and operations in a data-path graph into a design using physically realizable module'.;. A second par.s of module binding will occur after control allocation.

At present Iho module binding portion of the design system is primarily a rc'-rnrch tool that will be used lo investigate automated-module binding. A goal of this research'is to model the module binding problem **sufficiently** lo generalize this part of tho CMU-DA system lo handle a wide range of module types from LSI chips through Standard Colls. The implemented portions of MDBS were used lo assist a designer in binding data-part modulo* lo the CMU PDP-8/E data-paths produced by the Onla-Memory Allocator described in Section 4. The results of the data-part module binding are compared to the DEC PDP-8/E design in Section 7. A similar comparison will be made to the standard-cell binding after those results are presented in Section 8.

6. Organization of MDBS

MDl3S consists of four sections shown in Figure 5: an I/O section thai is responsible for translaling between Ihc internal and external forms of I he pnlh graph; a Module Data Base t?vcco'iri mpf.lmnir.m; a command language interface; and the module binding mechanism.

The input/output section is tho interface lo other parts of the CMU-DA system. Tho path graph, generated by an allocator is placed in internal form for processing. The output file bar. the same format a» Iho inpul path graph (with module binding information appended) and can be reread by the input section for additional processing.

Tho Modulo Dala Uasc is a hierarchical data base that is distributed in various ASCII files. Tho highest lovol of Ihc data base i> the index which i» road automatically during system initialization. Tho index contains pointers lo all defined desipn-r.tylo r.els, each of which is a collodion of module sets appropriate for a given design style. A design stylo set file contains poinlorr. lo Iho actual module set information (the ""Data Book[11]) and summary information (typical cost, speed,

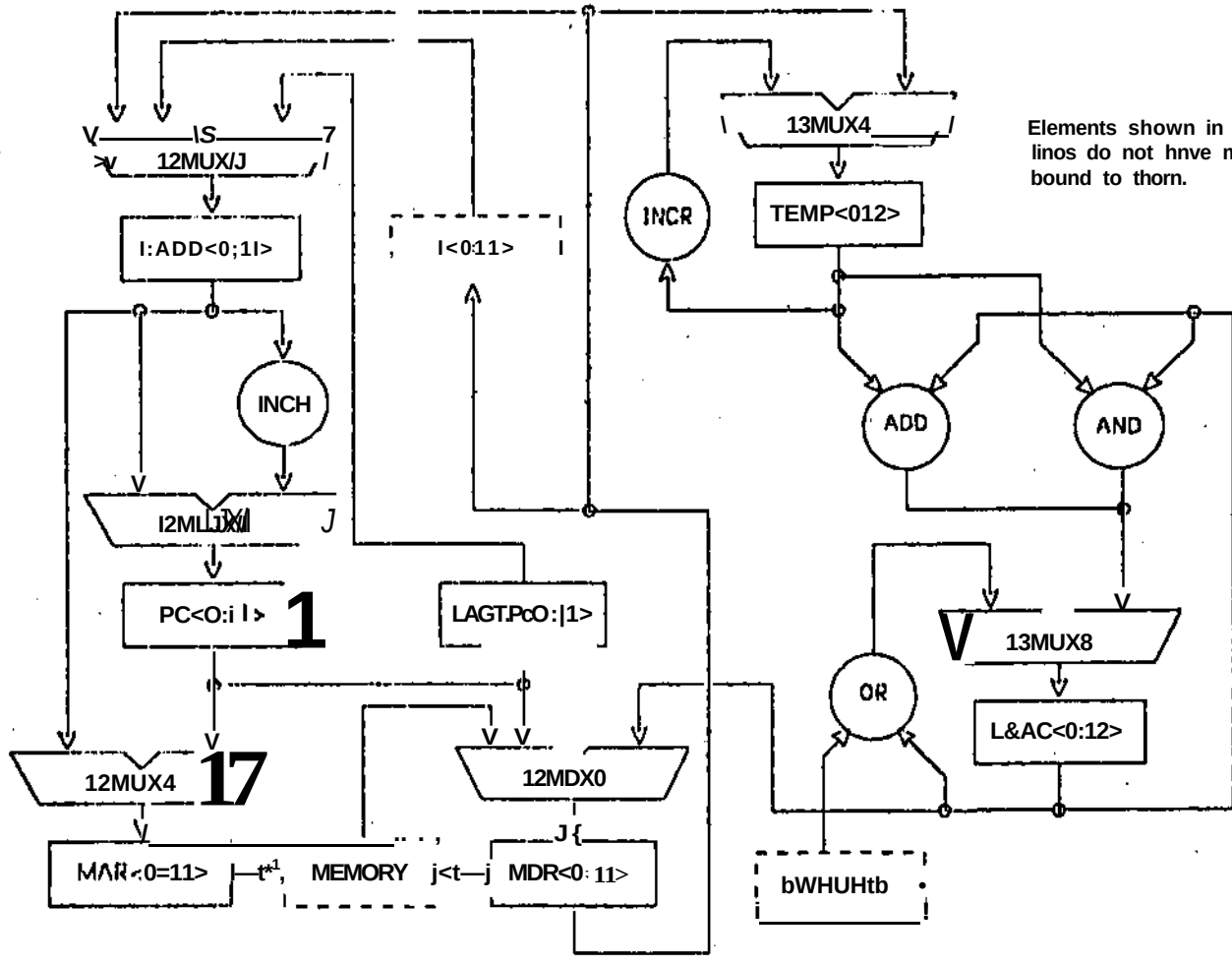
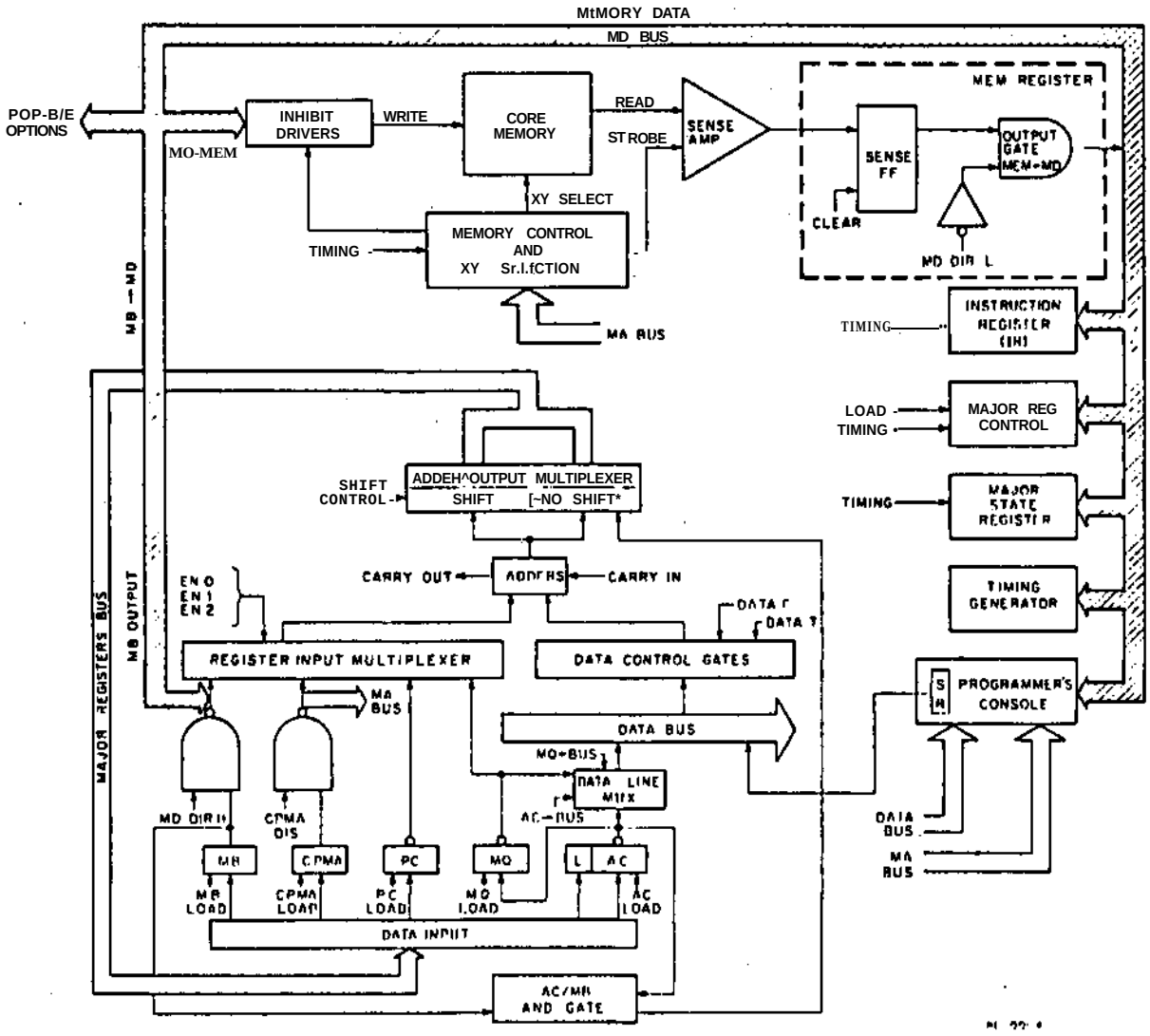


Figure 3: Allocator Gcneralecl PDP-8/E Data Paths



Fi$_K$uro 4: OLC PDP-8/E Data Paths

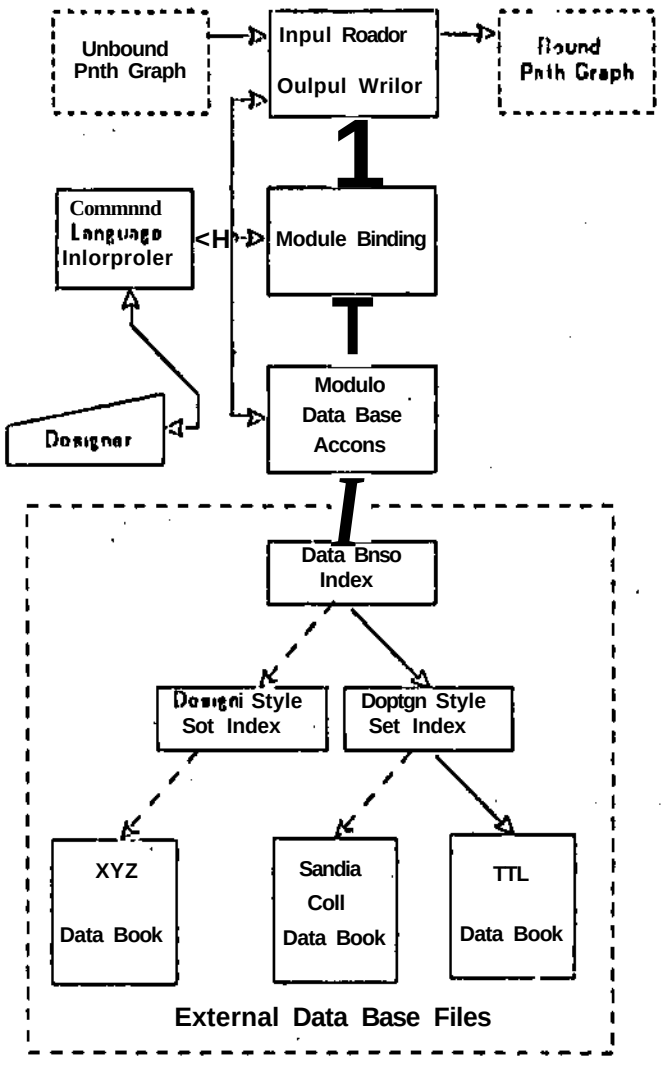


Figure 5: Organization of the Module Data Base System

load, and drive capabilities of modules in the set). Access to data bar.c information during module binding occurs frequently and utilizes various lovok of detail from basic type matching through specific delay and timing characteristics.

The Command Language Interpreter (CLI) is the experimenter/designer's interface to the module binding system. The CLI moy be used only to select a file to be processed and direct the disposition of the resulting bound path-graph. The CLI also provides a number of tools to inspect the path-graph, modify the graph structure and bind designer selected modules to specific, nodes of the graph.

The module hinder applies transformations to both the path graph and module functions in order to match the desired behavior with the available building blocks. The graph transformations are localized decompositions or combinations of nodes that preserve the specific behavior.' Module transformations are primarily combinations of nodes since modules cannot be physically decomposed. However, multifunction modules such as shift registers and ALUs may require partitioning of the non-conflicting functions of the same module onto r.cparalr? nodes of the grnph.

The mosi prevalent type of graph transform is localized to a single node or several connected nodes of a similar type. Registers are usually decomposed into nodes of smaller bit width. Logical operators nre usually modified by reduction to a canonical form, lhon synthesized with available module operations. Multiplexors and demultiplexers are frequently transformed from a single level to a multilevel form. Arithmetic operators (particularly the signed and complement arithmetic modes) require algorithmic decomposition.

fA m6re complex type of transformation involves the combination of nodes of different types into single nodes capable of multiple functions. Shift operators and special purpose arithmetic operators (increment, decrement, and clear)

are generally combined with register functions in available module sets. These transformations often provide significant reductions in the graph complexity by elimination of constants and reduction of multiplexor size.

Operator node transformations primarily involve the application of axioms and identities to combine available modules into an aggregate that performs a desired function. Boolean identities and DeMorgan's Theorem will direct logical synthesis. Arithmetic mode transformations (for unsigned, signed-magnitude, two's complement, and one's complement) will be utilized to synthesize required modes from available modes.

For cases where single transformations on either the graph or the modules do not provide the desired match, an iterative approach will be utilized. The graph and the module transforms will be alternately applied until one or more matches are found, a cycle is detected in the transformations, no further gain is detected by applying transforms, or one of the system constraints (speed, cost, etc.) is violated by the resulting implementation.

A central goal of the CMU-DA system is to produce designs that have been optimized toward the designer's objectives and fall within the external constraints. Module binding is the first operation in the design system that attaches actual costs to the implementation and has specific speed, delay, and power information available. Therefore, evaluation of the bound design must be performed to insure compliance with the constraints. Critical constraints (i.e. constraints which the designs *must* meet) will be dynamically estimated by projecting the final value based on an extrapolation from the number of nodes remaining to be bound and the accumulated value for the nodes already bound; a true evaluation of the fully bound design will be made as a final pass to insure compliance with the constraints. The dynamic evaluation will be used to select between functionally identical module choices with different performance parameters.

7. PDP8 Baseline Example

A partial module binding of the PDP-8/E was done with the available pieces of MDBS, which contain limited transform capabilities.

The module binding section contains a set of primitive operations that can modify the path graph structure (but not its behavior). The operations allow register nodes to be split at bit boundaries, operator nodes to be joined into single nodes with wider bit widths, nodes to be inserted, and nodes to be deleted. These operations allow the path graph nodes to be transformed to conform with the structure of available modules. As the MDBS becomes more fully implemented, these primitive operations will be used to build larger scale path graph transforms that can be applied automatically.

The existing system aids the designer in producing correctly bound path graphs by the strict enforcement of rules concerning application of the structural modification primitives. These operations are restricted to minimize the possibility of modifying the behavior of the path graph. Some examples of these rules are:

- Nodes may not be deleted while connected to more than one link.
- Links may not be deleted while joining two nodes.
- Half links (i.e., signals to external sources) may not be deleted.
- Operator nodes may be joined only if they are of the same type.

A different but equally important kind of assistance is provided to allow the designer to display any aspect of the binding process. For example, the designer may display one module in the data base, all modules providing a specific function, or the entire data book.

The module choices were made by the designer using information from the path graph and a small TTL module data base from the distributed Design Style Set. The purpose of binding the data part with the existing system is to contrast the module selection with a hand designed PDP-8/E. This example provides a worst-case from which to judge the performance of MDBS as more capabilities are added.

Figure 6 was generated by the MDBS. The registers (named variables), operators, and multiplexors are identified in the "Comp." (Component) column. The "Device" column lists the name of the selected package. "Mods." lists the number of modules required to implement the component function. The term "module" refers to a separate functional unit in this context. There may be several modules contained in one package. "Pkgs." lists the number of packages required for each function. "Gates" is an estimate of the equivalent number of logic gates to implement the function. The percentage of the total gates required is listed in the "% Total Gates" column. The cost of each function implementation is computed as the basic cost for the number of packages required plus an overhead mounting cost of $3.00 per package. The percentage of the total cost attributed to each function is listed in the "% Total Cost" column.

Comp.	Device	Mods.	Pkgs.	Gates	%Total Gates	Cost	%Total Cost
ENDD	SN74174	2	2	76	3.98	7.78	2.58
LNC	SN74194	3	3	105	5.50	11.67	3.87
LNST.P	SN74174	2	2	76	3.98	7.78	2.58
PC	SN74161	3	3	204	10.69	11.67	3.87
MDR	SN74174	2	2	76	3.98	7.78	2.58
MAR	SN74174	2	2	76	3.98	7.78	2.58
TEMP<0>	SN7474	1	1	6	0.31	3.35	1.11
TEMP	SN74174	2	2	76	3.98	7.78	2.58
EQL	SN7485	3	3	93	4.87	11.37	3.77
EQL	SN7485	3	3	93	4.87	11.37	3.77
INCR	SN7483	3	3	108	5.66	10.77	3.57
INCR	SN7483	3	3	108	5.66	10.77	3.57
AND	SN7408	12	3	12	0.63	9.60	3.18
OR	SN7432	12	3	12	0.63	9.75	3.23
ADD	SN7483	4	4	144	7.54	14.36	4.76
13MUX8	SN74151	13	13	156	8.17	46.67	15.48
12MUX4	SN74153	12	6	96	5.03	21.54	7.14
12MUX4	SN74153	12	6	96	5.03	21.54	7.14
12MUX4	SN74153	12	6	96	5.03	21.54	7.14
12MUX4	SN74153	12	6	96	5.03	21.54	7.14
13MUX4	SN74153	13	7	104	5.45	25.13	8.33
Totals		131	83	1909	100.00	301.54	100.00

Component Class	% Gates	% Cost
Registers	36.40	19.17
Operators	29.86	23.49
Multiplexors	33.74	52.37

Figure 6: Module Utilization For PDP-8/E Data Part

The choice of registers differed from the hand module binding in a few locations. The hand bound PDP-8/E used SN74194s (four bit universal shift registers) exclusively while the MDBS chose SN74174s (six bit D registers) when there was no requirement for the added shift capability. The Program Counter (PC) register was selected as three SN74161s (universal counters) based on the INC flag associated with the path graph node. The Accumulator (LAC) was first split into a one bit node and a twelve bit node, then the twelve bit part was allocated as three SN74194s. This is the same allocation that was done by hand. However, the

choice only ':nlisfios the shift operation flags (l.SHFT and RSlin). The increment requirement (INC) has effectively been partitioned out and must be bound separately.

The package count wa'. 30/ higher for the MDBS selection than for the \)[.C implementation (64 packages for DCC vs. 83 packages for MOFifO. This agrees closely with lho results obtained by selecting modules strictly by hand (refer to Section 4). The 30/ difference is attributable to the different design styles used and the allocator's implementation of the design. Comparing the total cost of modules for the DEC implementation and the MDf3S implementation (Figure 6), it is found that the costs also are 307 higher for the' automated implementation, while the number of equivalent gates is 652 higher for the automated implementation. This indicates that MDE35 chose modules with a higher level of integration than DEC did.

A comparison of the percentage of equivalent gator, arid lho percentage of cost accumulated in three functional classes (registers, operators, and multiplexors) indicates surprisingly uniform comparisons. Thn percentage of both gates and cost is higher for registers in the MDHS implementation than in the DEC implementation. This trend is expected since the DEC PDP-8/E uses a central accumulator design style. Also, the slightly lower percentages for gates, and costs in the operator class is reasonable for I he DEC implementation. The most surprising comparison is the near identical percentage of gates devoted to data path routing (i.e. multiplexors) in the two designs implemented in different design styles. It would be expected that lho central .accumulator style would utilize more data path routing than lhe distributed style. This apparent anomaly'is a rluo to llv *xrc.a* where the module binding can make local improvements in a path graph for distributed designs. I3y utilizing functions intrinsic to certain modules (such ns the CU~Al? on registers), constants and their associated data paths can bo eliminated and improve the cost of implementing a design.

The TTL module binding using MDBS compares favorably with the DCC implementation and previous hand module bindings of the automated path graph. It is expected that much improvement in the package count (and the cost) is forthcoming as transforms and evaluation techniques are implemented in MDFJS. However, TTI. module binding is just one objective of a generalized design system. The following section discusses an approach to binding CMOS standard cells to a design with the objective of being able to automate and produce LSI designs.

8. Standard Coll Generation

The Sandia standard cell library ("111 can also be used as physical modules lo implement lho automated PDP-8/E data part design. Then, using the Sandia software package [1, 10], it is possible to produce a simulation, insert faults, and perform automated cell placement and 1C mask generation for a CMOS LSI chip implementation.

The standard cell binder used for this experiment was a small, automatic, package which accessed a local data base of Sanclin colls. This package only performed the essential transformations on the graph-expansion of nodes to match the standard cells. However, this package also gives us a worst-case measure for module binding performance.

The translation of the design from one environment, the module binder output, lo another, the simulator input, involves both the expansion of multi-bit paths produced by the module bincJer to the single bit connection format of the simulator input and also the explicit identification of fan-out points. In addition to this latter process, termed resolving, the translator must generate pale specific parameters, such as propagation delays, by computing capacitances lo obtain a realistic simulation using SALOGS. Delay parameters are inserted in lh«

simulation model by the ur.o of delay paler* wilh associated times.

The input to 5AL0GS is a description of the network, written in NDL [3]. • NDL describes the interconnection of functional blocks. Input and control signals are generated through the SALOGS simulation Innguaco* SALSIM **[4].** ' In SAL0G5, gate representation includes built-in .simulator elements such as inverters, transmission Rales, NAND, AND, OR, and NOR gates. In addition, any set of elements can bo defined as a functional block and the block Used as a now element.

The NDL can Ihon bo used to automatically generate an 1C maśk and to determine chip area. The portion of the total data-path area taken up by the different modules is summarized in Figure 7.

The upper portion of the Figure compares the size of the data-path elements listed in Figure 3. The lower part of the table describes some modules that are more accurately defined as control than data-path. As expected, the percent of sub-total Rate count in the upper portion of the table closely resembles the results from the TTL binding shown in Figure 6.

In sum, the CMU PDP-8 design required #74,042 mil-sqr, ignoring the area taken up by routing. The experience with Sandia's IC mask closipn system indicates that routing takes up about an additional 75/ of the area occupied by the standard cells,'yielding a chip area of 129,574 mil-sqr. Dy way of comparison, Intersil's onr' chip CMOS CPU implementation of the PDP-8 lakes up 29,0 M mil-sqr [6]. It is estimated that 357 of Intersil's CPU chip is devoted to the functional elements equivalent to that generated by the CMU-DA system. Thus, there is a factor of J3 difference in the area required for the two designs.

Component	Oraa	/ of Total fire*	Niimbor of tGatos	7 of Total GMos	7. of CAIO subtotal
PC	5124.3	6.9	170	9.4	13.0
me	/i.177.1	5.9	117	6.5	8.9
noo	3839.9	5.2	117	6.5	8.9
OR	432.0	0.6	18.0	1.0	1.4
HND	476.3	0.6	18.0	1.0	'1.4
inrr,3	1413.3	1.9	49	2.7	3.8
INCR	3544.6	4.8	108	6,0	8.3
*:13MUX4	4212.8	5.7	60	3.3	4.6
12MUX4	3101.8	5,3	56	3.1	4.3
12MUX4	3901.8	5.3	56	3.1	4.3
12IIUX4	3'im.a	5.3	56	3.1	4.3
12MUX4	3901.8	5.3	56	3.1	4.3
J3IIUX8	$)183.4	12.4	139	7.7	10.6
Enno	1323.0	1.8	45	2.5	3.4
INCR	?[>44.G	4.8	108	6.0	8.3
LD3T.P	M23.0	1.8	45	2.5	3.4
NOR	JJ23.0	1.8	45	2.5	3.4
MOR	1323.0	1.8	45	2.5	3.4
r.ub to t* 1	57067.5	77.2/	1308	72.5/	100.0/
flynNOZ	317.5	0.4	12	0.7	
i:*3>:K0L9	2JiOJ.O	3.3	74	4.1	
2xI?IWX2	2K4 4.2	3.G	50	2.8	
IxIMUX2	141.0	0.2	3	0.2	
2>EQL2	J!>8.4	0.2	5	0.3	
SWITCH	1323.0	1.8	45	2.5	
I	J373.0	1.8	45	2.5	
2xF.0L12	2210.6	3.0	64	3.5	
LSS	24r,n.9	3.3	84	4.7	
GEQ	24G0.9	3.3	84	4.7	
NFQ	11.03.8	1.5	32	1.7	
3xFLHG	330.8	0.5	11	0.6	
TOTfll	74042.0	100,1/	1817	100,8/	

<' G.i In counl l& In lorins of 2 Input NHND gates
** 13MUX4 Is 1 of 4 flUX tilth bit width of 13
*** nxTDt n hullc.itnR n cop l PIS of 9 Input EQl coinpstra tor

Figure 7: Module Data from Translator

A model can be dnvi$«:d to attribute this seemingly large difference to various p.'fls of the design system. Since each part of the design :.;ys»tcm builds on top of the previous stage, a multiplicative model is used. This model must take into account the non-optimality of the allocator, non-optimality of the module binder, differences in basic feature size, and the differences in routing techniques. The result of the allocator section indicates a design requiring 1.3 times the size of the DEC design. There is also a difference in the basic feature size of the Intersil and Sandia technologies. By way of comparison, a 12 bit register implemented with Sandia's standard cells occupies 4 times the area of equivalent register In Intersil's design [2]. The multiplicative model then becomes

$$13.0 \approx (1.3)(4)R_f$$

where R is a factor indicating a difference in size between the CMU design and tho Intersil design introduced by the module binder. In this case R « 2.5. Not included in the model are factors due to difference between hand packed and channel routing techniques, nor factors considering that large structures (e.g. wide multiplexors) can be more optimally designed by hand than by combination of simple standard cells.

In the worst case, assuming similar input structures and feature size, the CMU module binder would produce a design taking 2.5 times the area of the Intersil design. However, as discussed above, there are other factors which increase the CMU design size that were not accounted for in the model.

9. Summary and Conclusions

The paper has illustrated the methodology behind the CMU Design Automation System. In particular, the datapath of a non-trivial digital system (PDP-8/E) has been designed from an ISPL functional description. Two types of physical modules were bound to the datapath design.

The binding using TTL series modules indicated that the CMU design required 307- more modules than the DEC implementation. The binding using CMOS standard cells indicated that the CMU design is at most a factor of 2.5 off, and due to differences in routing techniques may be actually closer in area to the Intersil design.

As a whole Ih* system has demonstrated the synthesis function in digital system design. The allocator research indicates automated logic synthesis with optimization is feasble and specific module-set information is not necessary in order to produce a reasonable design. The module binding section has demonstrated how the system can design relative to new technologies. Future work with the design system will deal with optimization techniques to be used in better directing the design algorithms for more complex designs.

References

1. Barbocci,M., Barnes,G., Cattell,R., Siewiorek,D. The Symbolic Manipulation of Computer Descriptions ; The ISPS Computer Description Language. Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., March, 1978.

2. Bell, G., C. Mudge, J.E. McNamara. *Computer Engineering.* Digital Press, 1978.

3. Case, G.R. and J.D. Stauffer. SALOGS-IV, Program to Perform Logic Simulation and Fault Diagnosis. Proceedings 15th Design Automation Conference, IEEE, 1978.

4. Case, G.R. and7.D. Slauffer. SALSIM - A Language For Control of Digital Logic Simulation. Proceedings of the[1] 11th * Annual Asilomar Conference on Circuits, Systems, and Computers, IEEE Circuits and Systems Soc, IEEE Control Systems Soc, Naval Postgraduate School, Univ. of Santa Clara, November, 1977, pp. 370-373.

5. DEC Staff. *PDP-B/E Maintenance Manual.* Digital Equipment Corporation, 1972. DEC-8E-HR1B-D

6. Electronics Magazine Staff. CMOS, Moving Along. *Electronics 48,* 10 (May. 1975), .

7. Hafer, L. Data-Memory Allocation in the Distributed Logic Design Style. Master Th., Carnegie-Mellon University, December 1977.

8. Hafer, L.J, and A.C. Parker. Register-Transfer Level Automatic Digital Design: The Allocation Process. Proceedings of the 15th Design Automation Conference, IEEE, 1978.

9. Leive, G.W. The Binding of Modules to Abstract Digital Hardware Descriptions. PhD Thesis Proposal, Carnegie-Mellon University, 1977.

10. Preas, B.T. and C.W. Gwyn. Architecture For Contemporary Computer Aids to Generate IC Mask Layouts. Proceedings of the 11th Annual Asilomar Conference on Circuits, Systems, and Computers, IEEE Circuits and Systems Soc, IEEE Control Systems Soc, Naval Postgraduate School, Univ. of Santa Clara, November, 1977, pp. 309-317.

11. Sandia Staff. *Standard Cell User's Guide.* Sandia Laboratories, 1978.

1 2. Snow, E.A., D.P. Siewiorek and D.E. Thomas. A Technology-Relative Compuler-Aided Design System: Abstract Representations, Transformations and Design Tradeoffs. Proceedings of Ihe 15th Design Automation Conference, IEEE, June, 1978.

13. Thomas, D.E. and D.P. Siewiorek. Measuring Designer Performance to Verify Design Automated Systems. Proceedings of the 14th Design Automation Conference, IEEE, 1977, pp. 411-418.