MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 126

On Structured Digital Systems

by

Suhas Patil

(A paper to be published in the Proceedings of the Workshop on Computer Hardware Description Languages and Their Application)

This paper was prepared with the support of the National Science Foundation under grant DCR74-21822.

July 1975

On Structured Digital Systems*

Suhas S. Patil
Project MAC
Massachusetts Institute of Technology
Cambridge, Massachusetts

Abstract. In this tutorial paper the author takes the position that digital systems should be described and implemented as structured systems. This approach is needed to satisfactorily cope with new requirements such as correctness of systems and to cope with timing problems which are becoming increasingly difficult to handle for large systems which are implemented with fast logic gates. The paper gives examples of asynchronous structured systems and argues that asynchronous methods are most suited for structured design.

Introduction

One of the objectives of this paper is to argue that the time has come to both describe and implement hardware in a structured way.

Over the years, the requirements of digital hardware have changed in important ways, and the progress in LSI technology has made it possible to attempt a structured approach which at one time would have been too expensive to be useful. Correctness of designed hardware, its reliable operation and modifiability are becoming very important. And better methods are needed to deal with timing problems in digital systems which are built with faster and faster logic and whose size continues to increase. It is my feeling that the challenges posed by the changed environment can be best answered by structured descriptions and possibly structured implementations of digital systems. It is also my feeling that the theoretical work on structured hardware has progressed to a point where it can provide useful guidance to the practical people.

This paper first makes a case for structured hardware and then answers the question of how digital hardware can be described and implemented in a structured way by presenting a concrete scheme for structured design and implementation.

Methods of Design.

One of the most commonly used method of digital system design is based on the concept of a centralized clock which determines the instances at which the registers in the system accept new values. In addition to the registers, the system has appropriate logic circuits which determine the new values for the registers from the present values. This method of design was the most effective known method for avoiding timing problems. This step-by-step behavior of the system was also well matched for the description and analysis of systems, which were small enough to allow the designer to follow all the changes taking place in the entire system at each clock instance. There is no doubt that this method has served us well for many years, but because of the changing nature of digital systems and what is expected from them, and also because of the changes which have taken place in the technology of implementation, it is necessary to examine what further advances in the methodology of digital system description and implementation are warranted.

Assumptions about the digital systems which were valid then are no longer equally valid. For example, because of the high speed of operation, the propagation delays of wires can no longer be neglected, and the assumption that the clock designates the same time everywhere in the system is also no longer truly valid. If the system consists of loosely coupled parts, then the old assumption of being able to fully follow all actions taking place everywhere at each instant of time also becomes unreasonable. As the assumption on which the old methods of digital system description and implementation were based become less and less valid, the ability of these methods to realize systems free from timing problems is lessened. One can no longer be sure that the realized digital systems do not have timing problems until they have been thoroughly tested and debugged -- a process which is both costly and annoying.

New Requirements. With the increase in size and complexity of digital systems, new requirements are being added. A large digital system, for example, may fail not only because of component failure but also because of incorrect design. Verification of correctness (if not a proof of correctness) of the digital system is thus as important to the overall reliability of a system as is the use of reliable components. This situation is similar to the one faced by the software people who must not only run their systems on reliable hardware but must verify the correctness of their systems. It is my opinion that this requirement will become increasingly important with time just as it has become important with software, and the people concerned with methods of hardware description and realization should pay proper attention to this requirement.

*This paper was prepared with the support of the National Science Foundation under grant DCR74-21822.

The requirement that the design should be clear to understand has always been there, but with increase in the size of digital systems this requirement is becoming very important. Clarity of design is needed not only for aesthetic reasons but because a design which lacks clarity is likely to have more errors, and the task of detection of errors and verification of the correctness is likely to be harder. Hardware description methodologies should be designed to help the designer in the task of presenting a clearer description of his system.

Digital systems often need to be changed. There are usually last-minute design changes before the design is frozen, and often changes have to be made in existing systems to meet unexpected new situations. Errors can be introduced into the system when such changes are made if seemingly small alterations require changes to be made in many different places and if the alterations have effects which are not easily seen. It is important that the changes not destroy the efforts that have already been expended in arriving at the correctness proofs of the existing system. Otherwise there may be great temptation in not working out the proof of correctness of the new system and leave the matter as is, hoping that what was correct in the old system is also correct in the new system -- a potential source of errors in real systems.

Structured Systems

I feel that without a systematic approach to the description and implementation of digital systems, the above mentioned requirements would be very difficult to meet. Furthermore, not only should the method of description be systematic, but to gain the most possible advantage, the systems as they are conceived should have a systematic structure. Structuring, an important concept of system theory, is often employed to impart to the systems some well defined and desirable properties. A desirable property for example might be the freedom from timing problems or a property known as speed independence. A suitable structure, for example, may help a system sustain some component faults. A hierarchical structure, for example, may improve the clarity of the system description.

There is no universally accepted definition of structured systems. For our purposes I would like to define structured systems to be those which are consistent with the philosophy underlined below.

The Concepts of Structured Systems.

(i) Composition -- The system should be composed of well defined parts which play well defined (and preferably intuitively appealing) roles in the operation of the system. The parts may be arranged in many levels of hierarchy; at each level parts may be interconnected with well defined links to form a network of parts, and a (composite) part at one level may be further expanded (defined) at lower levels of hierarchy.

(ii) Communication -- The logical interaction among parts takes place only through the defined links and follows a communication discipline (and communication mechanism) set forth for the system; the parts should be logically independent of each other except for the explicit interaction set forth in the description of the system.

(iii) Comprehension -- The complexity of the system at each level of description should be within the limits of comprehension of the agent trying to understand the operation of the system or verify its correctness.

The earliest work on structured hardware design that I am aware of is that of Muller [1] in connection with ILLIAC II. Then Clark and his group at Washington University made important contributions with the work on macro-modules [2]. Following this, Bell at Carnegie Mellon University [3] and Patil and Dennis at MIT [4] made their contribution to structured hardware design with RTM and asynchronous modular systems, respectively. Project LOGOS at Case Western University has also made important contributions [5] to structured hardware design.

The particular approach which is presented below is based on the work done at MIT.

Examples of Structured Digital Systems

Two examples will be considered. The first one (Figure 1) illustrates the basic ideas with the aid of a multiplier unit which is implemented with other (elementary) units. The second example (Figure 2) illustrates a conventional processor constructed from structured units. Both examples describe asynchronous digital systems.

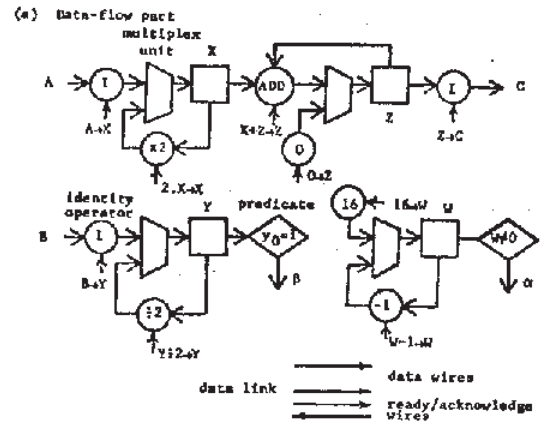
The digital system consists of two parts, a data-flow part and a control part. The data flow part has the memory cells and the operators which store and manipulate the data. Data manipulation may involve some transformation on the data or just the transportation of the data from one point to another. The data-carrying paths are called data-links and have a pair of control wires, called ready-acknowledge wires, in addition to the data carrying wires. In the examples which we shall consider here, data is first placed on the data wires and then a signal is sent on the ready wire in the direction of the link to signal that the data is available on the link. The operator or the memory cell (a register) to which the link is connected can take advantage of the signal to initiate action. For example, a memory cell can update its value and send the new data on the output data-links of the cell. Later the cell will receive acknowledge signals on the output-links, and when all acknowledge signals are received it will return a signal on the acknowledge wire of the input data-link (the link from which it received the data).

If a link has only the ready/acknowledge wires and no data wires, then the link is called a control link. Many operators have such control links which originate from the control part and are used by the control to command the operator to perform a predesignated operation. If the operator is to be commanded to perform one of several actions then a data-link would control the operator; the data on the link will perform the selection of the operation and the ready signal on the link will initiate the operation. A signal on the acknowledge wire will indicate that all the actions that must take place in connection with the execution of the operation have been

completed so that the execution of other operations may now be initiated.

Some operators such as the multiplexer unit (Figure 1a) have no control link from the control part because all control signals needed by it are available from the data links. In Figure 1 we use such a unit to take care of multiple fan-in into a memory cell (register). When this unit receives input on any of the input links, it establishes connection between that input and the output and the data is passed on to the output together with a ready signal. When it receives an acknowledge signal from the cell (register), it acknowledges on the input link; the connection between that input link and the output link remains established until data is received from some other input. Normally two inputs would not be sent to it concurrently, but if that situation is possible then an arbiter in the unit would determine which input will be connected to the output first. The data flow part of the system thus uses both schema-type operators and data flow-type operators [6].

In the examples that are presented here, the operators expect that the data values that are available to it on the data wires of the input link remain at that value until new data is presented so that the need to store the input data in the operator is eliminated. For example, in Figure 1 the adder returns an acknowledge signal on an input data-link immediately after it receives the data (and the ready signal) but expects the input data to be valid throughout its operation and until a new data is received on the same link. When a ready signal is received on the control link from the control part, the adder assumes that the input data are available at the input, performs the addition, puts new data on the output link and sends a ready signal on the output link. When an acknowledge signal is received on the output link, the adder acknowledges on the control link.



(b) Control specification

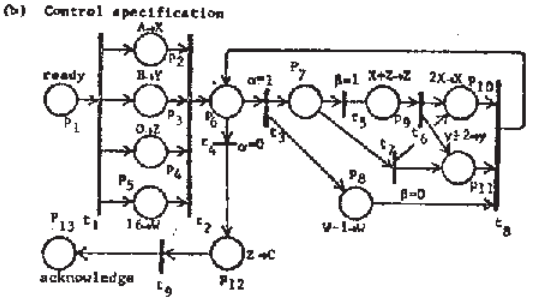


Figure 1. A multiplier.

The control part is specified with a Petri net [4,7]. A Petri net consists of places (drawn as circles) and transitions (drawn as bars) which are connected by directed arcs. Arcs connect only places to transitions and transitions to places. A place may contain a token (we consider only such nets which do not have more than one token in a place); a transition needs a token from each input place to fire. In the act of firing, a transition removes a token from each input place and then puts a token in each output place. If a place represents a call to an operator (Figure 1) then placing a token in the place results in a ready signal being sent to that operator on a link. This signal commands the operator to perform predesignated action, and when the action is completed, an acknowledge signal is returned on the link by the operator. The token at the place, which was frozen at that place during this time, is then released to be used in the firing of other transitions.

Example of the Multiplier. The integer multiplier shown in Figure 2 is based on the long-hand multiplication algorithm. Operands A and B are placed in cells X and Y, respectively. Cell Z is initialized to 0 and cell W to 16 (the number of bits in the word representing the operand Y). Then the numbers equal to operand A multiplied by increasing powers of 2 are successively generated in cell X, and Y is shifted right. If at any stage the least significant bit of Y is a 1 then X is accumulated in Z; otherwise the computation proceeds to the next step.

The multiplier itself can be thought of as an operator, which multiplies A and B to produce result on C. Its operation is initiated by a ready signal which has the effect of putting a token in place p_1. When the operation is completed a token appears in place p_{13}. Then an acknowledge signal is returned in reply to the ready signal. The pair of places p_1 and p_{13} represent a ready-acknowledge link.

When a token is placed in p_1, transition t_1 fires and initiates operations $A \rightarrow X$, $B \rightarrow Y$, $0 \rightarrow Z$ and $16 \rightarrow W$ in parallel. When these operations are performed, transition t_2 puts a token in p_6. This token leads to the firing of either transition t_3 or t_4 depending on the Boolean condition of the output of predicate α. It may be noted that in the course of performing the operation $16 \rightarrow W$, predicate α gets the new value of W together with a ready signal. The predicate operator then evaluates the predicate and sends the outcome to the control unit on link α. When the control receives this value, an acknowledge signal is sent to the predicate. The predicate then returns an acknowledge signal to the cell W. The cell W returns an acknowledge signal to the operator which then acknowledges the completion of the operation $16 \rightarrow W$. Thus we are guaranteed that before p_6 gets a token the machinery to determine which way the token should go is already established.

Firing of transition t_3 puts a token in p_7 (which tests whether a step of accumulating X into Z is to be performed) and a token in p_8 (which decrements the value of W by 1). If the lowest order bit of Y is a 0 then β will be false and transition t_7 will directly put tokens in places p_{10} and p_{11} to prepare the system for the next step in iteration. When X is shifted left (multiplied by 2), Y is shifted right (divided by 2) and W is decremented by 1, transition t_8 fires putting a token in p_6 to start the next step of the iteration.

After 16 steps of iteration, α becomes false. Then transition t_4 puts a token in p_{12} which causes the result of computation to be sent out on the output link C. When an acknowledge signal is received on link C, transition t_9 puts a token p_{13} so that the multiplier can reply with an acknowledge signal to indicate that the multiplication has been performed and that the result of the multiplication has been delivered to whoever was supposed to receive it.

Example of a Simple Conventional Processor. Figure 2 illustrates a conventional processor with four general-purpose registers R_1, ..., R_4, program counter P, instruction register I, address register A, data register D and buses X and Y. It has one ALU and many (identity) transfer operators to move data from one point to another. The multiplexer puts the contents of the registers designated by the control input into the bus register X , and the distributor puts the contents of Y into a designated register. This processor (which has been composed only for the purpose of illustration) has five types of instruction: transfer, arithmetic/logical operation, load and store and halt. Figure 2b gives a specification of the control of the processor in an abbreviated notation for Petri nets. In this notation the name of the operations are written in place of the places which represent them and transitions are not drawn when several operations are performed in a single sequence. The control shows the decoding and execution of the instructions.

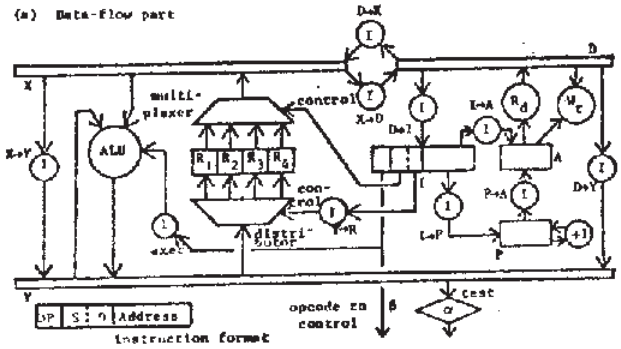


Figure 2. A conventional processor.

Question of Implementation

The components of data flow part can be implemented with the conventional MSI and LSI parts together with some additional circuitry to generate the ready/acknowledge signals. It is not very important what precise mechanism (simple delay or a more sophisticated circuitry to detect the completion of the operation) is used to obtain the ready and acknowledge signals but it is important that the operators and other units of the data-flow part when viewed from the outside conform to their functional behavior.

If one were to open the walls of an operator and view it as an electrical circuit, one may find that it has many modes of operation, both the desired and the undesired, especially if delays within the device are altered or if used in a manner different from what it is intended for. An important idea of structured design is that the operator is an abstraction of the circuit and at that level of abstraction it is well behaved if it is used in the manner it is intended to be used. (Parnas and Siewiorek discuss important ideas on levels of abstraction in a recent paper [8].)

Recent developments have made it possible to systematically implement the control part of the system (which is specified as a Petri net) in an asynchronous programmable logic array [9, 10]. In the array the row wires represent transitions and the columns (consisting of two wires each) represent either places, ready/acknowledge links or Boolean inputs. Two diodes at the cell in the interconnection of a row and a column implement the type of arc that connects the transition and the place; the diodes in the cell at the intersection of a Boolean column and a transition determine whether the firing of the transition requires the Boolean column to be in state 1 or 0 (true or false). At the top of the array each column is connected to a circuit of about the same complexity as a flip flop. The input/output wires of the control are connected to these circuits. In the diagram of Figure 3, which shows an implmentation of the control of Figure 1b, an arc from a transition to a place is represented by a cross (X) and an arc from a place to a transition by a dot (·), and 0 and 1 represent the Boolean condition required for firing of the transition.

One of the important features of the array is that in order to obtain the control part of a system it is sufficient to provide the Petri net specification of the control; the rest of the steps require simple mechanical translation which is guaranteed to produce a physical implementation consistent with the Petri net specification.

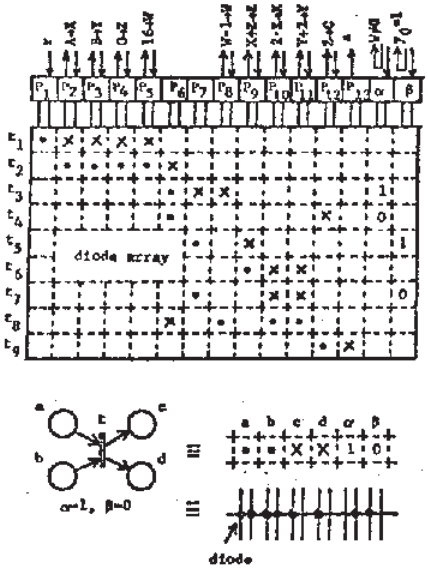


Figure 3. Logic array implementation of the control of Figure 1.

Structured Design and Asynchronous Systems

I feel that it is not an accident that researchers at Washington University, Carnegie Mellon University and MIT have chosen the asynchronous approach in working on structured digital systems. It is my opinion that structured asynchronous systems have distinct advantage in realizing the objectives of structured digital systems and in meeting the other requirements (correctness, modifiability and freedom from timing problems). People have carried negative feelings about asynchronous systems from the time work on problems of asynchronous circuits was published many years ago. There is no doubt that the asynchronous systems as they were conceived by those researchers have a lot of problems which those working with synchronous approach satisfactorily solved for systems of that time. People who have not followed the developments in asynchronous systems should take a note of the fact that structured asynchronous systems are very different kinds of objects than the unstructured asynchronous systems studied by the early researcher, and they are able to cope with problems of timing even better than the synchronous systems are able to.

In a structured asynchronous system the parts interact with well defined ready/acknowledge signals which precisely realize the logical and temporal relationship among the parts. The arrival of a signal itself rather than the exact moment of arrival is of significance. The parts are thus loosely coupled in the temporal sense; the logical dependency, which is the most important aspect of the interrelationship of the parts, is thus maintained even if some parts slow down or speed up.

The concept of hierarchical structure and levels of system goes very well with structured asynchronous systems. Both the principles of structured systems and the method of structured asynchronous digital systems require well defined communication discipline.

Discussion of Cost and Feasibility

The cost of structured digital systems when considered from the point of view of the number of components involved will certainly be greater than an unstructured system which takes advantage of all minimization techniques, but cost of components is not the only cost that is incurred in the design of the system. There is cost associated with how much time and effort is needed to design and debug a system, and if correctness is important then there is a cost associated with producing a convincing case or a proof of correctness of the system. Moreover, with the advent of LSI, it is no longer reasonable to take the count of the number of logic gates as a measure of the cost as the costs are determined more by the count of the packages involved, and there is a great variability about how many components can go into a package.

Even if the raw hardware costs of structured systems are higher than those of unstructured systems, it is my opinion that the LSI technology has brought down the cost of digital hardware to a level where the small additional cost is well worth the added benefit of structured systems.

Remarks

The design methodology which was discussed in this paper was at the register-operator level (sometimes called register transfer level). This is a very convenient level of abstraction for general implemantation of digital systems. Advances in the technology of LSI and advances in our knowledge of digital system design might one day make it possible to move to a higher level where each component may be more like a processor rather than an operator or a register. Not enough work has been done in this area to know exactly what form these systems will take, but it would be reasonable to guess that in addition to their own problems, these systems will have many features and problems similar to the software systems and that structured approach will perhaps become even more important than it is today.

Structured approach, in my opinion, offers many opportunities for important advances in the art of digital system description and implementation. Asynchronous systems may play an important role in achieving the goals of structured digital system design.

References

1. Muller, D. E. Asynchronous logics and application to information processing. Switching Theory in Space Technology, Stanford University Press 1963, 289-297.

2. Clark, W. A., et. al. Macromodular computer systems. Proceedings of the AFIPS, SJCC, 1967, 335-364.

3. Bell, G. C., J. Grason and A. Newell. Designing Computers and Digital Systems. Digital Press, Maynard, Mass. (1972).

4. Patil, S. S., and J. B. Dennis. Description and realization of digital systems. Proceedings of the Sixth Annual IEEE Computer Society International Conference, 1972, 223-226.

5. Bradshaw, F. T. Directed graph models for hardware/software design. Proceedings of IEEE Workshop on Computer Hardware Design, Languages and Their Applications, New York, September 1975.

6. Dennis, J. B., and D. P. Misunas. A computer architecture for highly parallel signal processing. Proceedings of the ACM 1974 National Conference, ACM, New York, 1974, 402-409.

7. Patil, S. S. Cellular arrays for asynchronous control. Proceedings of the ACM 7th Annual Workshop on Microprogramming, 1974, 178-185.

8. Parnas, D. L., and D. P. Siewiorek. Use of the concept of transparency in the design of hierarchically structured systems. Comm. of the ACM 18, 7 (July 1975), 401-408.

9. Patil, S. S. An Asynchronous Logic Array. Technical Memorandum 62, Project MAC, M.I.T., Cambridge, Mass., May 1975.

10. Patil, S. S. Micro-control for parallel asynchronous computers. Proceedings of the Euromicro Workshop, Nice France, North-Holland Publishing Co., 1975.