LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

Design of a CMOS Self-timed Two by Two Packet Router

Computation Structures Group Memo 242
December 1984

Tam-Ahn Chu

This research was supported by the National Science Foundation under grant MCS-7915255 and by a Hughes Fellowship.

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR COMPUTER SCIENCE

COMPUTATION STRUCTURES GROUP MEMO 242

DESIGN OF A CMOS SELF-TIMED TWO BY TWO PACKET ROUTER

Tam-Anh Chu

Abstract

This paper describes the design of a CMOS self-timed two by two packet router, using novel circuit techniques. The packet router is the constituent element of a store-and-forward communication network for a multiprocessor computer. The self-timed design methodology introduced here partitions the design into data paths and system controllers, the latter consisting of Finite State Machines and Distributed Control Structures. This organization allows the exploitation of concurrency in the design and is well suited for systems requiring parallel operation. A CMOS router was designed, fabricated and tested, a maximum throughput rate of 10.9MHz was achieved.

This research was supported by the National Science Foundation under grant MCS – 7915255 and by a Hughes Fellowship.

December 1984

1. Introduction

This paper describes a design of a self-timed two by two packet router, using novel circuit techniques. The packet router is the constituent element of a store-and-forward communication network [1] for a data-flow multiprocessor machine [6]. Using the indirect Boolean hypercube topology, the number of routers required to connect N processors is $\frac{1}{2}N\log_2 N$. For large N, a synchronous routing network of this size is difficult to maintain due to clock distribution problems, and has poor reliability due to different modes of failure; the most significant is failure due to the synchronizer problem [2]. A network built out of self-timed components can avoid these problems. The self-timed packet router is implemented in CMOS, and the performance of the fabricated chips is highly acceptable in terms of area, power and speed. The self-timed design methodology introduced here is different from previous approaches in that the architecture is partitioned into data path, finite state machines (FSM) and distributed structures (DS). The FMS's and DS's together constitute the *system controllers*. Data path modules consist of *data circuits* which perform the desired logic function, and *stage controllers*, whose task is to synchronize the modules' operation with others. In a *system controller*, a FSM provides predicate signals to steer the flow of control in the DS, which can be partly implemented in CMOS Domino logic. Self-timed control circuits are synthesized using a graph model called Signal Transition Graph. Within the context of this paper, we will introduce this graph as a specification tool; however, this graph approach can allow direct synthesis of self-timed circuits. This topic will be addressed more fully in another paper [4].

In Section 2, the general characteristics of this design methodology are summarized. Section 3 describes the router and its implementation, we will also present the behavior specification of the router and the specific realization in CMOS of data path components and their timing control circuitry. Section 4 is dedicated to the system controller and the implementation of the FSM and DS. Section 5 gives statistics on CMOS router chips fabricated by MOSIS in terms of area, speed and power consumption.

2. A Self-timed Methodology

Self-timed circuits form a subclass of the asynchronous switching circuits in which a uniform communication discipline such as the reset signaling protocol is enforced throughout the system. One feature of such an approach is the use of local communication and distributed control, as opposed to the global communication and central control of the synchronous approach, and it is well-suited for systems with a great deal of built-in parallelism. Another special property of self-timed circuits is *speed-independence* or *delay-insentivity*, as defined by Muller [9, 8], which indicates that they operate correctly regardless of the variation in delay of logic components. Such a property is quite desirable as it implies that self-timed circuits do not fail due to hazard and race conditions, and that they operate reliably under a wide range of operating conditions that affect the delay characteristics of components. A basic assumption in the realization of speed-independent circuits is that gate delays, and in some more general models, wire delays, are unbounded. However, in most practical situations, only realizations with unbounded gate delay are possible; circuits with unbounded wire delay can be built by using data coding schemes such as the dual-rail code [10], except for feedback wires in basic asynchronous state components (such as SR flipflops and C-elements) which are assumed to have negligible delays. We have found that a design methodology based on this unbounded gate and wire delays assumption is too inefficient and results in unnecessary performance penalties in area and speed, despite the robustness and other desirable features of such an approach [3].

In this self-timed design methodology, the architecture is partitioned into *data paths* and *system controllers*. A *data path* is broken down into *data circuits* which perform the desired operation, and *stage controllers*, whose task is to synchronize the modules' operation with others. A *system controller* is composed of a *Finite State Machine* (FSM) and a *Distributed Structure* (DS), and the FSM is used to produce predicate signals to steer the flow of control in the DS. Control modules constituting a DS are basic constructs which allow the exploitation of concurrency of control operations proposed by Dennis [5]. This type of control discipline is a natural and efficient method for achieving parallelism. For applications where a great deal of concurrency exists, this decomposition of a system controller is more efficient than the central control organization, as the latter approach requires either product machines or a large amount of system states to coordinate concurrent operations.

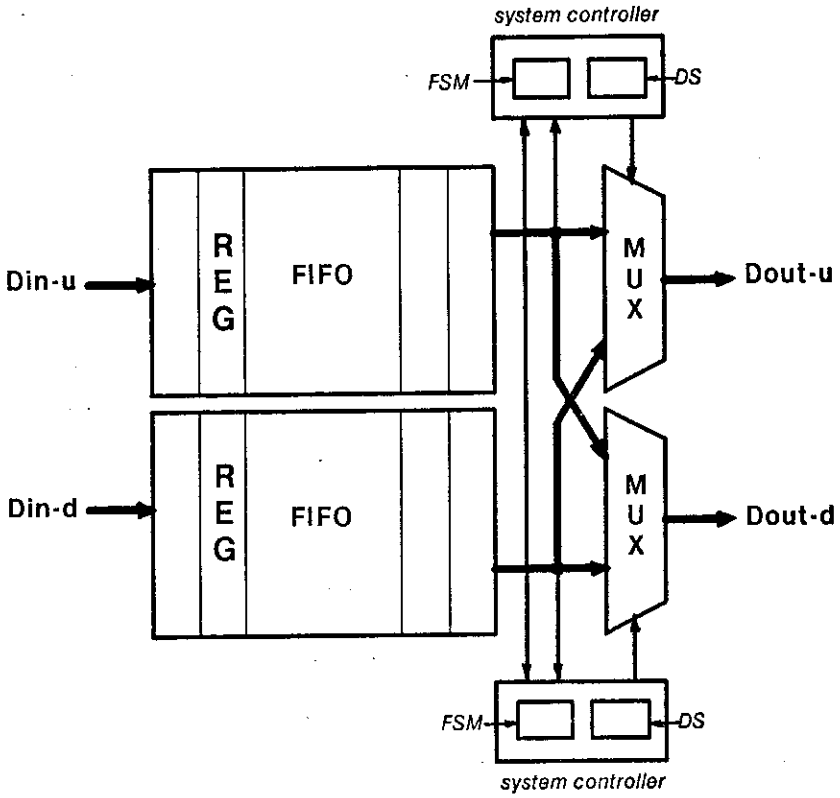
In terms of VLSI implementation, the *stage controllers* for data paths, the DS's and a part of the FSM's are basically timing control circuits which can be synthesized directly from a graph model called Signal Transition Graph, which yields efficient and truly speed-independent circuits. *Data circuits* can be built using matched delays instead of speed-independent techniques such as dual-rail code, as the delay of logic gates and wires in a chip with no defects is bounded and can be well characterized, and one can almost always determine the upper bound of the delay of circuit elements and construct other circuits to track that delay. Such a delay tracking scheme is more economical, and in practice, works well over wide ranges of variation in voltage, temperature and circuit layout. On the other hand, control circuitry normally involves random logic with diverse loading conditions. A truly self-timed and speed-independent implementation is justified here, as it produces robust circuits and avoids problems due to races and hazards. Finally, special circuit structures related to the technology (e.g. CMOS) can be selected, and the internal design of self-timed modules can be optimized for parallel operation. Techniques used include the exploitation of pipelined and parallel operation for modules, in particular, the use of parallel resetting in modules to reduce the cycle time of the reset signaling protocol. In certain cases, CMOS domino logic [7] provides a direct implementation of the parallel reset scheme.

3. Oganization of the Router

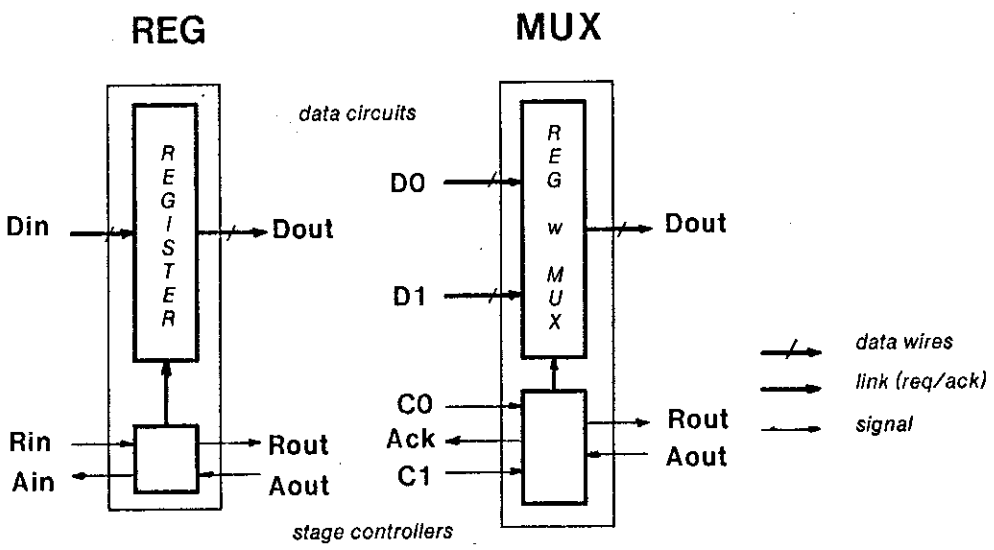
3.1 Behavior Specification

The block diagram of the router is shown in Figure 1(a). It contains two FIFO queues to hold packets sent in byte-serial format. Packets are of variable length, and an extra bit called Last-Byte is appended to a byte to delimit the packet boundary. This bit is 1 for the last byte of a packet and 0 for others. The first byte of a packet contains the address information. The router decodes the address and forwards the packet to the desired output port, an address bit of 0 will form a link from that input port to the upper output port, a 1 will form a link to the lower output port. There are two *system controllers*, each consists of a FSM and a DS. The controllers read the address bit and the LB (Last-Byte) signals and generate control signals for the output multiplexors. These control signals are determined from the first byte of a packet and recycled for other bytes. Two controllers also communicate as packets from one input port may need to go to the opposite output port. In case when packets from both input ports require the same output port, an arbiter is used to resolve conflicts.

Fig. 1. Block diagram of the Router and Data Path Components



(a) Organization of a Two by Two Packet Router



(b) Register and Multiplexor Modules

3.2 Data Path Components

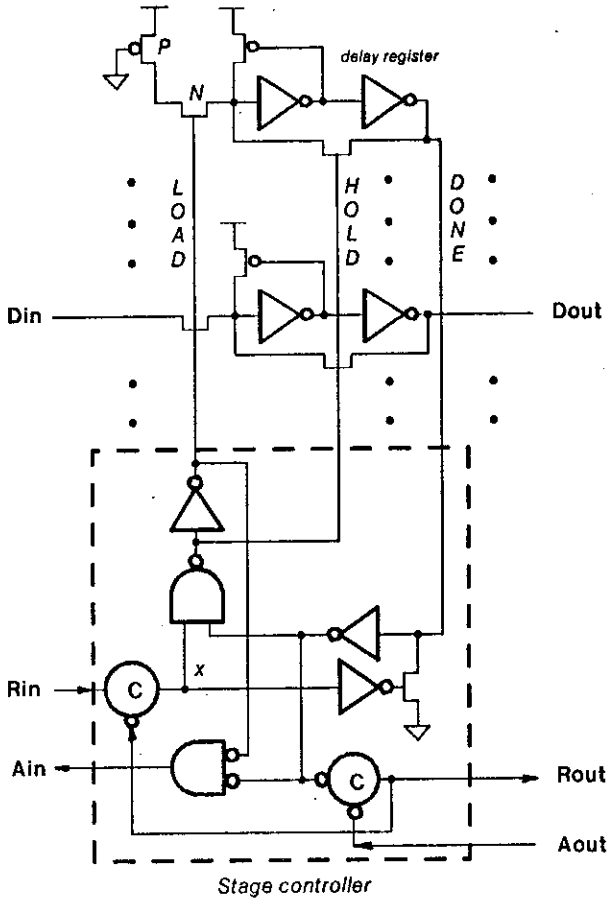
Two main data path modules are self-timed registers which form the FIFO's, and the self-timed multiplexors. These modules consist of a *data circuit* and a *stage controller* which handles the timing and signaling protocol. Figure 1(b) shows the structure of register and multiplexor modules. This section will describe their implementation in details.

An implementation of the register module is shown in Figure 2. Its *data circuit* is a pseudo-static register, where N-channel pass-transistors are used instead of CMOS transmission gates. A feedback P-type pullup restores the correct High voltage level due to a voltage threshold drop across the N-channel pass transistors.. The geometries of the N and P devices of the CMOS inverters are sized such that they are sensitive to low-to-high transition to allow the feedback mechanism to take over quickly after the input goes High. In the same Figure, a *delay register* located at the top is used to accurately track the time required to load the registers, and this technique works well even if the *LOAD* control signal is propagated on poly-silicon. Even though this pseudo-CMOS register configuration certainly reduces layout area, the main reason for using it is to ensure the correct timing sequence for turning on and off the input and feedback N-type pass transistors, which are controlled by *LOAD* and *HOLD* signals. In a standard CMOS design with transmission gates, simultaneous transitions of a controlling signal (*LOAD* or *HOLD*) and its complement are required to prevent race conditions. But such requirement may be difficult to achieve in most cases, and special care must be exercised to make sure the opposite transitions of a control signal are close enough to each other.

The stage controller operates in pipelined fashion: once data are loaded into the registers, it notifies both input and output ports by raising A_{in} and R_{out}. The operation of the controller is as follows. After input data D_{in} are stable, R_{in} is raised, which causes *HOLD* to go Low and then *LOAD* to go High. Thus, the N-type pass gate in the feedback path is turned off before the input pass gate is turned on, ensuring that registers in the current stage do not drive the outputs of registers in the preceding stage, as this may cause data to shift backward while the control signals move forward. This may happen if the transient persists long enough and registers in the preceding stage have their feedback N-type transistors on and their input N-type transistors off. By only using *LOAD* and *HOLD* signals but not their complements, their correct timing relation can be guaranteed easily. After data are loaded, *DONE* signal goes High which in turns causes *HOLD* to go High and *LOAD* Low. At the same time, R_{out} and A_{in} are raised. However, in order to prevent input data from changing while *LOAD* is still High, one has to wait until *LOAD* goes Low before raising A_{in}. When R_{in} goes Low and R_{out} is High (indicating that data are available at the output), the *DONE* signal, being the ouput of the delay register, is reset through a strong N-type pull-down device. Subsequently, A_{in} drops and R_{out} also drops after A_{out} has gone High, signifying that the succeeding module has read the data.

This controller supports pipelined operation, makes use of internal parallelism to speed up the module and reduce the reset time. Another feature of this design is it permits data to be stored in consecutive stage without overwrite problem even though a master/slave register configuration is not used. It was synthesized using a graph model called Signal Transition Graph (STG). Figure 3 shows an STG specification of the register stage controller, where a node corresponds to a signal transition in the circuit, an arc specifies a timing constraint between signal transitions. The + and - signs indicate the direction of the transtion. In this graph, an internal node *x* has been introduced, this node is the output of a C-element. A Join operation is depicted by a number of arcs from different transitions joining at one transition, and it specifies that the resulting transition occurs or *fires* only

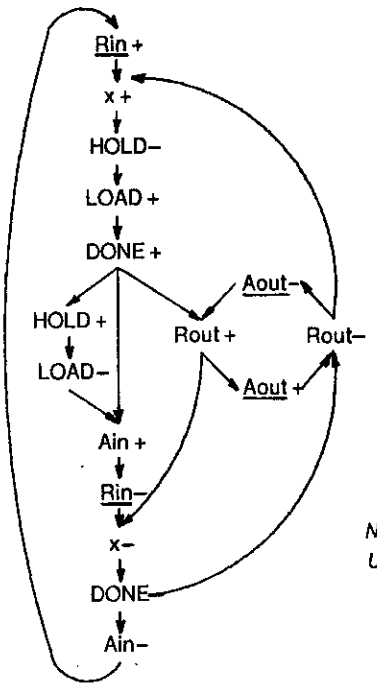
Fig. 2. Implementation of Register module



after all transitions that cause it have occurred. A Fork operation is depicted by a number of arcs emanating from one transition to a number of different ones, indicating that these fire only after the causing transition has fired. Since there is no specific time metric assigned to each arc, the occurence of the resulting transitions is concurrent. Thus the Fork is the basic mechanism for creating parallel operation. A method for synthesizing self-timed control circuits directly from Signal Transition Graphs is described in [4]. Within the scope of this paper, we will not discuss the synthesis technique and only use the graph as a behavior specification for the circuit.

The Multiplexor module (Figure 4) consists of a controller and data circuits. The latter are built from register cells with input N-type multiplexors controlled by $LOAD_0$ and $LOAD_1$ signals. The controller is a modification of one of the register modules. C_0, C_1 form a dual-rail signal (they cannot both be 1 at the same time) which selects the input multiplexors. This signal is generated by the *system controllers*. The operation of the Multiplexor module is similar to the Register module and is quite obvious from Figure 4. The Multiplexor module is also pipelined and uses internal parallel operation to improve speed.

Fig. 3. Signal Transition Graph of Stage Controller of Register Module



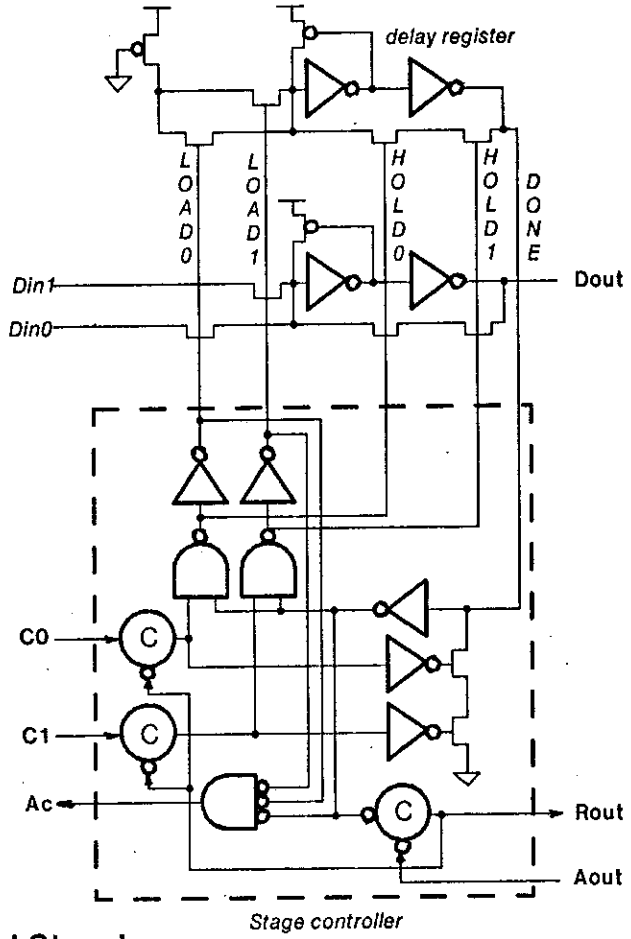
4. System controllers

As mentioned earlier, a system controller is partitioned into a FSM and a DS. The FSM provides predicate signals to steer the flow of control in the DS. This control structure is based on the ideas in [5], and we have found that CMOS Domino logic can provide a well-suited and efficient implementation.

4.1 Finite State Machine

The block diagram of the FSM is shown in Figure 5(a). Basically, it consists of a combinational logic function (a multiplexor in this case), a master and a slave register to hold the current and next state. The FSM accepts the address d_0-*in* and the Last-Byte signal *lb-in* from the FIFO and produces three predicate signals *fb*, *lb* and *dir*, where *fb* = *1* only if the current packet byte is the first one, *lb* = *1* only if it the last, and *dir* is the address bit, sampled during the first packet byte and recycled for other bytes. Module **S*** accepts the input request signal R_{in} and produces a clock pulse Φ and its complement $\overline{\Phi}$ at link 1. As shown in Figure 5(b), the same feedback timing loop as used in the register module is employed here to generate a clock pulse whose duration is guaranteed to be long enough for loading the data into register cells. Note that due to a more relaxing timing requirement, transmission gates are used and controlled by both Φ and $\overline{\Phi}$, whose transitions in opposite direction need not be strictly simultaneous to prevent race conditions. The **S*** and **D** modules in Figure 5(a) are examples of *control modules*, **S*** is a modified *Sequence* module which produces a *pulse* (instead of a *level* signal as in the normal S-module) and then activates a *request/acknowledge* cycle at link 2. This pulse is the clock signal Φ which loads data into the master registers. Immediately after Φ goes low, $\overline{\Phi}$ becomes High and the state information is moved to the slave registers. **D** module is a *Decision* module which activates a cycle at link **F** if the predicate signal is *false*, and link **T** if it is *true*.

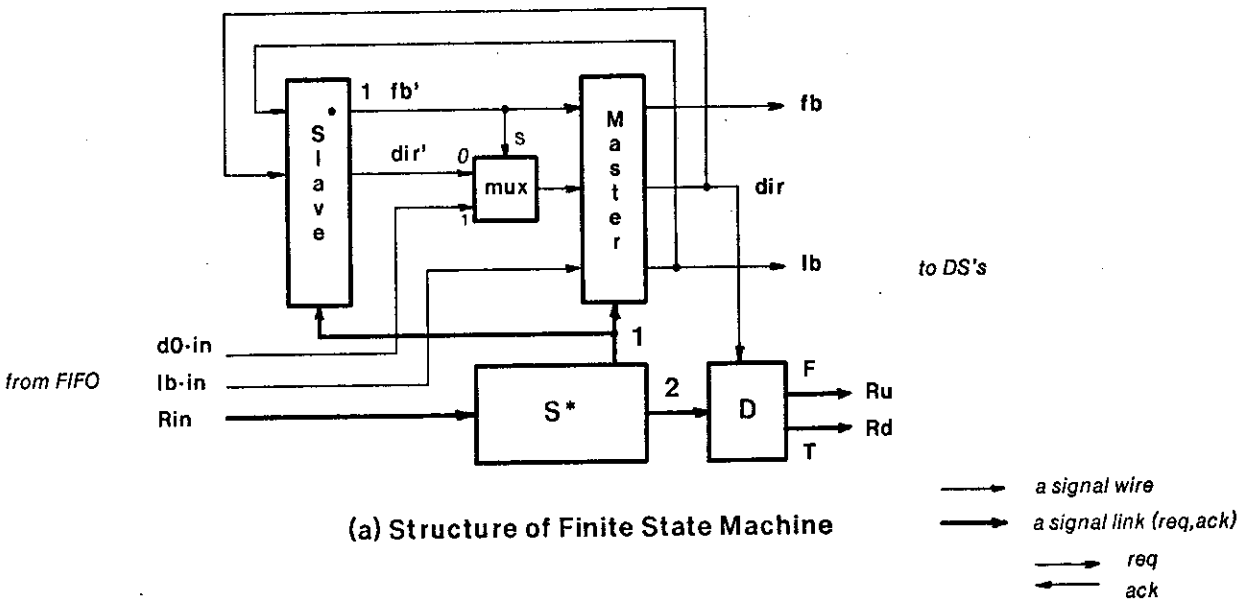
Fig. 4. Implementation of Multiplexor Module



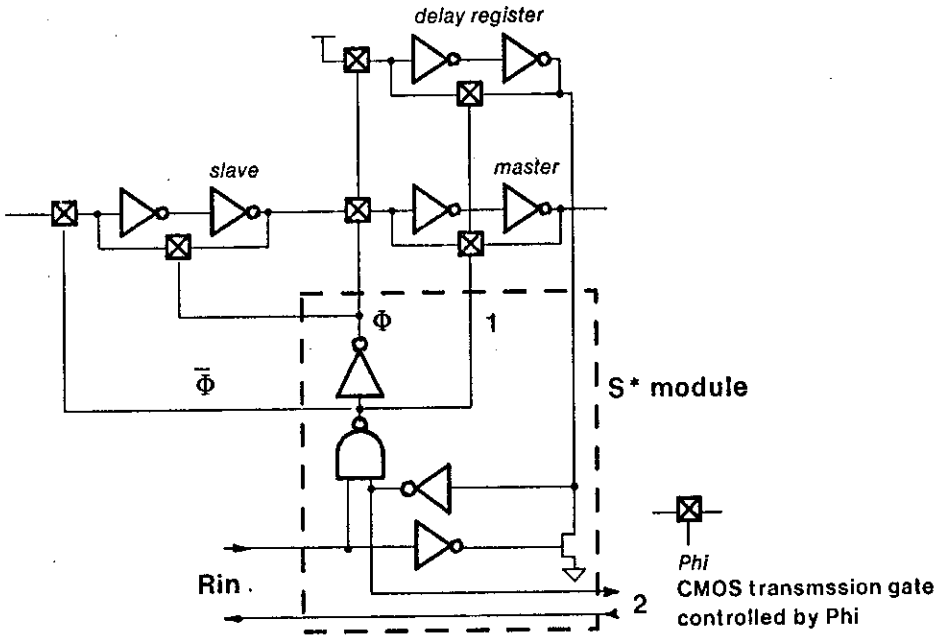
4.2 Distributed Structure

The distributes structure consists of control modules which communicate locally through *request/acknowledge* signal links. The representation and logic implementation of some modules are shown in Figure 6(a), where heavy arcs indicate links and signal wires with a *p* are *predicates* (those with an *e* are enabling signals for domino logic discussed later). An **S**-module serially activates links 1 and then 2, in that order. A **D**-module activates link **T** if the predicate signal is *true*, and link **F** if the predicate is *false*. The **UD**-module is a composite module which can accept *request* from either input link and raise either a True or False signal at the output port. As shown in Figure 6(a), it will raise C_0 if the upper input link is active, and C_1 if the lower link is, the Acknowledge signal is routed back to the proper sender through AND gates. Due to the fact that concurrent *requests* from both FSM's can occur, the DS needs an *arbiter* module, as shown in Figure 6(b). This particular device contains an arbiter circuit and other circuitry for two *Engage* and two *Release* ports. Once it is engaged by a *request* from one port, the other port is locked out. The state of the arbiter also remains locked until it is released by a Release request for the same port. The self-timed arbiter circuit (Fig. 6c) contains a cross-coupled flipflop, a threshold detector to prevent problems due to the metastable state (a similar NMOS design is given in [10]) and two strong N-type pull-down devices which reset the arbiter through a global Reset signal. It allows each port (1 or 2) to go through a complete *request/acknowledge* cycle before a request from the other port is processed.

Fig. 5. Structure and Implementation of Finite State Machine

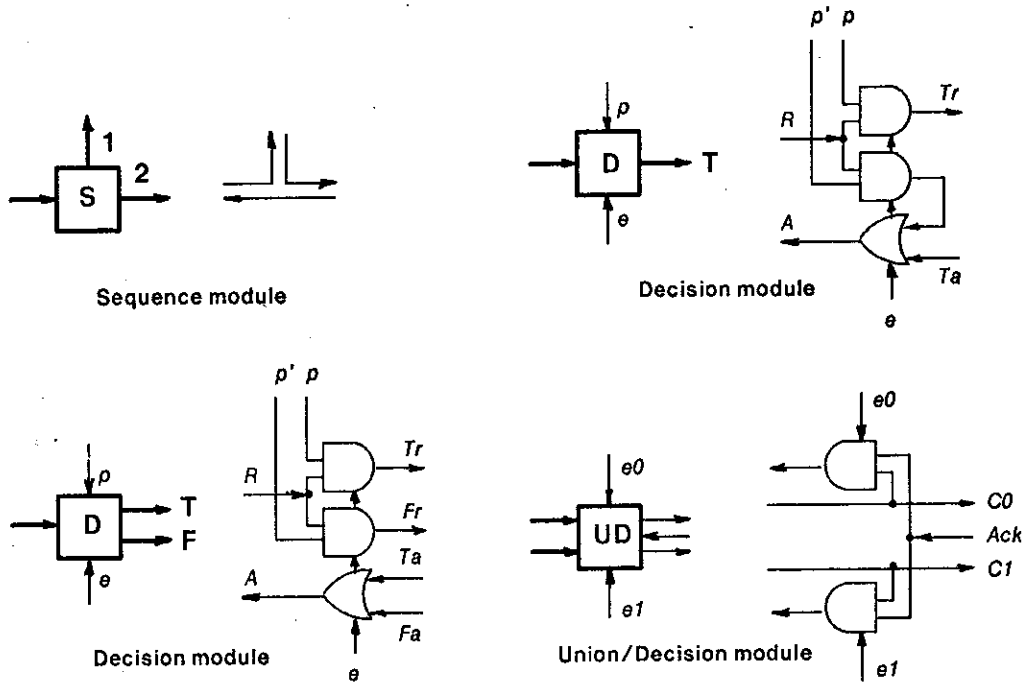


(a) Structure of Finite State Machine

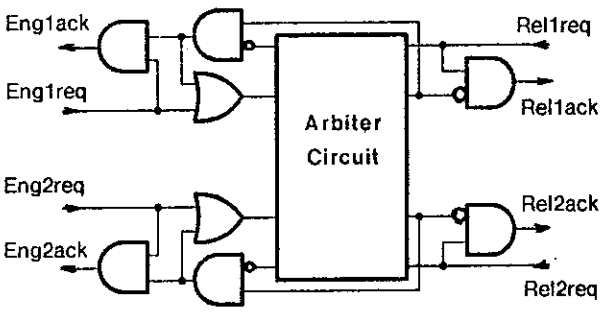


(b) Implementation of FSM

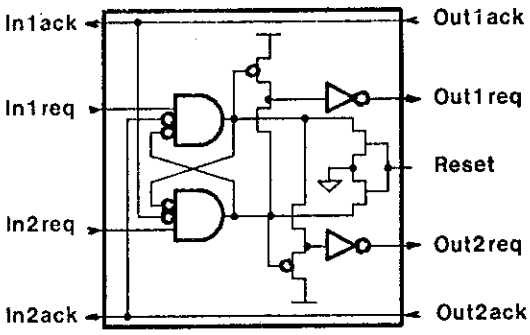
Fig. 6. Control Modules for Distributed Structure



(a) Implementation of Control modules



(b) An Arbiter module with Engage/Release links



(c) A CMOS self-timed arbiter circuit

The operation of the distributed structure (Fig. 7) can be described as follows. On receiving *requests* **R-u** and/or **R-d** from the state machines, DS checks if the current byte is the first byte (the address), and if so, it sends an *Engage request* to the Arbiter module. Subsequently, after receiving an *acknowledge* from the Arbiter module, it activates a dual rail signal (C_0,C_1) to control the Multiplexor. Upon receiving an acknowledge from the Multiplexor, it then checks if the current byte is the last byte; in this case, a *Release request* is sent to release the engaged Arbiter. Due to the design of the Arbiter module, *Engage/Release* action cannot take place in the same cycle, therefore, a packet must contains at least two bytes, the first byte causes an *Engage* cycle, the last a *Release* cycle.

In searching for a fast circuit implementation for the DS, we have found that CMOS domino logic [7] is an elegant and efficient way to realize these control structures. Figure 8(a) shows a domino logic gate which perform a sum of product logic function $x = ab + cd$. This function is implemented by N-type transistor network in the rectangular box shown. When the enable signal *e* is Low, the P-type pullup is turned on and it keeps the internal node *y* High, therefore the output Low. When *e* goes High, the pullup is turned off and the N-type pulldown is turned on. If a pulldown path in the N-type network exists, the internal node will be discharged to Ground and the output becomes High, otherwise, the internal remains High due to a weak feedback P-type device marked with *. Note that while *e* is High, if the internal node is accidentally discharged by some input transient, it cannot be restored to High, unless *e* goes Low again. In Figure 8(b), a CMOS implementation of a C-element is shown (this implementation is not a Domino logic gate), where the inverter marked with an * is a weak one, i.e., it is easily overpowered by the series pullup and pulldown devices. As indicated in Figures 6(a) and 7, the *e* signals are enabling signal for domino logic gates. Domino logic provides a natural and efficient way for implementing self-timed circuits. During the *active phase* of a cycle of operation, signals are propagated forward from one module to another to perform the control

Fig. 7. Distributed Control Structure for the Router

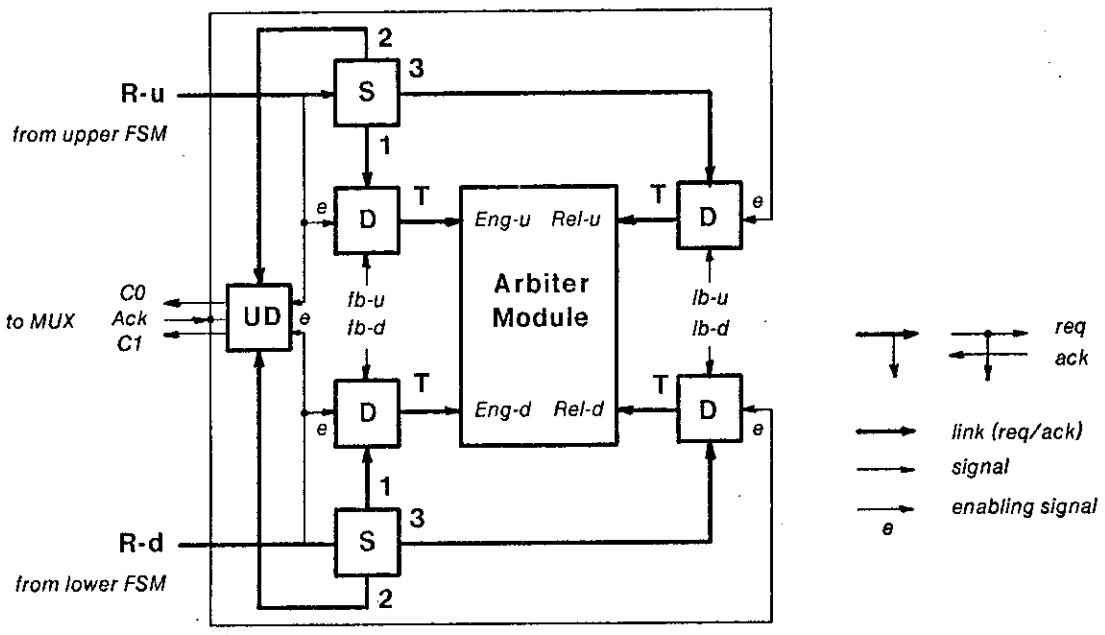
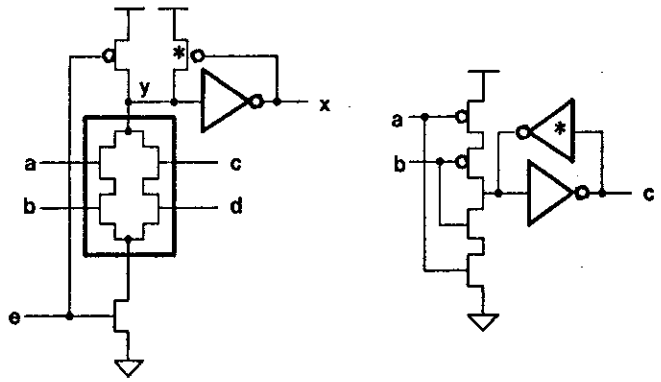


Fig. 8. Special CMOS circuits



(a) A domino gate (b) A C-element

functions. During the *reset phase* (when *request* and *acknowledge* signals go Low), all modules can be reset at the same time in some cases. A domino logic implementation does exactly this, and the time required for resetting is reduced to about two (inverting) logic stage delays. Due to the nature of precharged circuits, domino logic can only be used to implement multi-stage logic with no inverting intermediate variables. These variables become High in the reset state (when the enable signale is Low) and they may cause accidental discharge of internal nodes through temporary pulldown paths in the NMOS networks when the enable signal *e* starts to go High. There are methods to overcome this limitation but in general will make the structure unduly complicated. The best solution is to use dual-rail code for inverted signals and single rail for others. In our case, even though inverted signals such as *fb'* and *lb'* are used in **D** modules, no accidental discharge will occur because they are inputs to AND gates controlled by other inputs which are guaranteed to be Low in reset state; no special technique is required here because no pulldown paths exist when *e* starts to go High.

Self-timed circuit modules operating in pipelined fashion cannot be implemented with domino logic, as their state (reset or active) depends on both input and output signals. A chain of non-pipelined modules can be considered to belong to the same *domain* which can be driven from the same enabling signal. In Figure 7(a), the first two modules, **D** (with predicate *fb*) and **UD** in a chain, belong to the same domain and are enabled by the same *request* signal **R-u**. The last **D**-module (with predicate *lb*) belongs to another domain and is enabled by signal Ack from the Multiplexor module.

Another optimization used here is to collapse and-or logic stages into single domino logic gates, even though they may not belong to the same control modules. We have been able to achieve about 30% speed-up in the controller design by using domino logic instead of a straight-forward CMOS implementation of the control modules.

5. Results and Conclusions

A 2x2 packet router was designed and submitted for fabrication through MOSIS, an ARPA-sponsored foundry. The router is implemented in 3 micron CMOS technology (λ = 1.5), it contains 2456 transistors and occupies a layout area of 3.13 x 2.34 mm^2. We tested 46 chips in total and obtained 30 fully operational chips. The throughput rates (bytes/second) are 10.9 MHz maximum, 4.76 MHz minimum and 5.96 MHz average. The average latency is about 1 microsecond, and average power consumption at full speed is around 75 mW.

Traditionally, self-timed circuits have been built using a large amount of hardware, because of this, and also the use of the reset signaling protocol, it also takes a proprotionally large amount of time to operate. Within the domain of VLSI, certain basic assumptions can be relaxed to reflect the realism of integrated circuits. By assuming that gate delays are bounded and can be matched reasonably well in certain cases, one can design self-timed systems with highly acceptable performance. In the router design, except for a few places in data circuits (e.g. the *delay registers*) where delay assumptions are explicitly made and extra care is exercised, all is self-timed and *speed-independent*. This claim can be partly verified by noting that, for example, the STG specification for the Register module contains no specific timing metric but only precedence constraints between signal changes. These self-timed modules, as seen, are much more efficient both in terms of the amount of hardware and speed in comparison to other traditional approaches. Therefore, this approach may provide a method for managing the timing complexity of large systems yet to be implemented using VLSI technology.

Acknowledgments

The author would like to thank Prof. Dennis for his continuing support and guidance. Clement Leung and Jim Vallenga contributed ideas to this paper. Tom Wanuga simulated the Distributed Structure using Terman's RNL. Andy Boughton, Willie Lim and Bill Ackerman gave valuable comments and suggestions and read the drafts. People in FLA group at MIT are thanked for making layout tools available. Walid Hamdy tested the router chips.

References

1. Boughton, G A, Routing Network for Packet Communication Systems. PhD thesis, EECS Department, MIT, August 1984.

2. Chaney, T J and Molnar, C E . Anomalous Behavior of Synchronizer and Arbiter Circuits, *IEEE-TC*, Vol. C-22, April 1973.

3. Chu, T. The Design, Implementation and Testing of a Self-timed Two by Two Packet Router. *Computation Structures Group Memo 225*, Laboratory for Computer Science, Feb 1983.

4. Chu, T. Designing VLSI Self-timed Systems using Signal Transition Graphs. Submitted for publication, 1985 Chapel Hill Conference on VLSI.

5. Dennis, J B. Modular, Asynchronous Control Structures for a High Performance Processor. *Record of the Project MAC Conference on Concurrent Systems and Parallel Computation*, pp 55 – 80.

6. Dennis, J B. Data Flow Supercomputers. *IEEE Computers*, Nov 1980.

7. Krambeck R. H. et al. High-speed Compact Circuits with CMOS. *IEEE JSSC* pp 614 – 619, Vol. SC-17. No. 3, June 1982.

8. Miller, R E . Switching Theory, Vol. 2, Chapter 10, Wiley and Sons, NY 1965.

9. Muller, D E and Bartky, W S . A Theory of Asynchronous Circuits. *Proceedings of an International Symposium on the Theory of Switching, the Annals of the Computation Laboratory of Harvard University, Vol 29, Part 1,* Harvard University Press, Cambridge, Mass. 1959, pp 204 – 243.

10. Seitz, C L . System Timing, Chapter 7 of *Introduction to VLSI Systems* by Mead and Conway, Addison Wesley 1980.