A STANDARD DESIGN FRAME FOR VLSI CIRCUIT PROTOTYPING

R. H. Katz
S. Weiss

Computer Sciences Technical Report #485

October 1982

A Standard Design Frame for VLSI Circuit Prototyping

R. H. Katz and S. Weiss
Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706

ABSTRACT: The VLSI Design Community needs a standard frame in which to design circuits. The usefulness of custom designed prototype circuits is greatly enhanced if they can be "plugged" directly into a frame providing system capabilities (memory, I/O ports, timers, standard industry bus, etc.). The testing and debugging of fabricated chips at the systems level is also greatly improved. We describe the design and implementation of a frame, based on an easily obtained microcomputer board and a custom designed control circuit. We have validated that the control circuit works by fabricating an interfaced test circuit and the controller on the same die, and plugging the package into the board. The test circuit succeeded in accessing system components (memory) through the controller circuit.

1. Introduction

The structured design method of Mead and Conway [MEAD80] enables designers with little previous experience in integrated circuit technology to undertake the implementation of VLSI Systems. While many interesting projects have been rapidly carried through to fabrication (e.g., RISC [FITZ81], SCHEME-79 [STEE80], Pixel-Planes [FUCH81], Geometry Engine [CLAR80], RSA Cipher [RIVE80]), little has yet been done to enable a designer to rapidly incorporate his prototype into a real computer system.

There are several reasons for this. First, it is difficult to debug a VLSI chip, especially in a university environment where equipment and expertise are limited. Although excellent tools are available for design validation, e.g., the circuit simulators and design rule checkers of [BRYA80, BAKE80], the designer is left on his own once he receives his chip. Further, no amount of validation will insure that the processing run completed successfully or that the chip itself has escaped from disabling defects. Tools for debugging are needed if prototype chips are to be used in systems. The question is how to make debugging more tractable in the university environment.

Secondly, although it sible to achieve quick turnaround of designs into chips [CONW80, BROO81], e chip is not the same as a prototype system. It must first be surrounded chips for memory and communications. The additional effort needed to c tem is substantial and time consuming, and is a major stumbling block to u ototypes in practical situations.

The VLSI Design Co ds a standard *design frame* for VLSI prototype systems, i.e., an environm vides adequate memory and input/output capabilities, comes in a readil package", and is easy to interface to prototype circuits. An appropriately ser circuit becomes a prototype system by simply inserting it into the fr

In the remainder of , we describe our approach for constructing a design frame. The frame esigned, implemented, and tested, and the concept has been demonstr e successful test of a simple prototype circuit.

2. Requirements for a e

A design frame (1) ort a standard bus structure, so a variety of devices and processors ca ed with the prototype over the bus, (2) provide adequate memory for pr and configuration information, and (3) have basic elements that are easy ur design frame is based on the Intel iSBC 86/12A Single Board Computer. rd board contains an 8086 microcomputer, 32K bytes of dynamic RAM e to 64K), an RS-232 interface, three programmable I/O ports, program an interrupt controller, provision for 16K bytes of ROM, and MULTIBUS int ol logic. A feature of the iSBC 86/12A is that the memory is dual porte e on-board processor and the MULTIBUS. Thus a master 8086 on one b osit and examine the memory contents on a slave board.

The iSBC 86/12A e basic "hardware" environment for the frame. The "software" is a circuit es the 8086's own bus interface unit. We have

implemented this controller for inclusion on the same die as a user's prototype circuit. When an I/O or memory access operation is requested by the prototype circuit, the bus controller acknowledges the request and generates bus control signals that are identical to those of the 8086. The other subsystems on the board (I/O ports, memory chips, MULTIBUS controller, etc.) behave as though they are interfaced with an 8086. Since the user circuit/bus controller combination chip must be pin compatible with the 8086 package, only a few pins that are not needed by the board are available to the user circuit. The package is inserted into the board in place of the 8086 to surround the user circuit with system components, thus creating the prototype system. Our design frame is a cannibalized iSBC 86/12A board and the CIF code for the bus controller circuit (see figures 1 and 2).

In the following sections, we provide details on the interface to the design frame, the design and implementation of the 8086 compatible bus controller, and

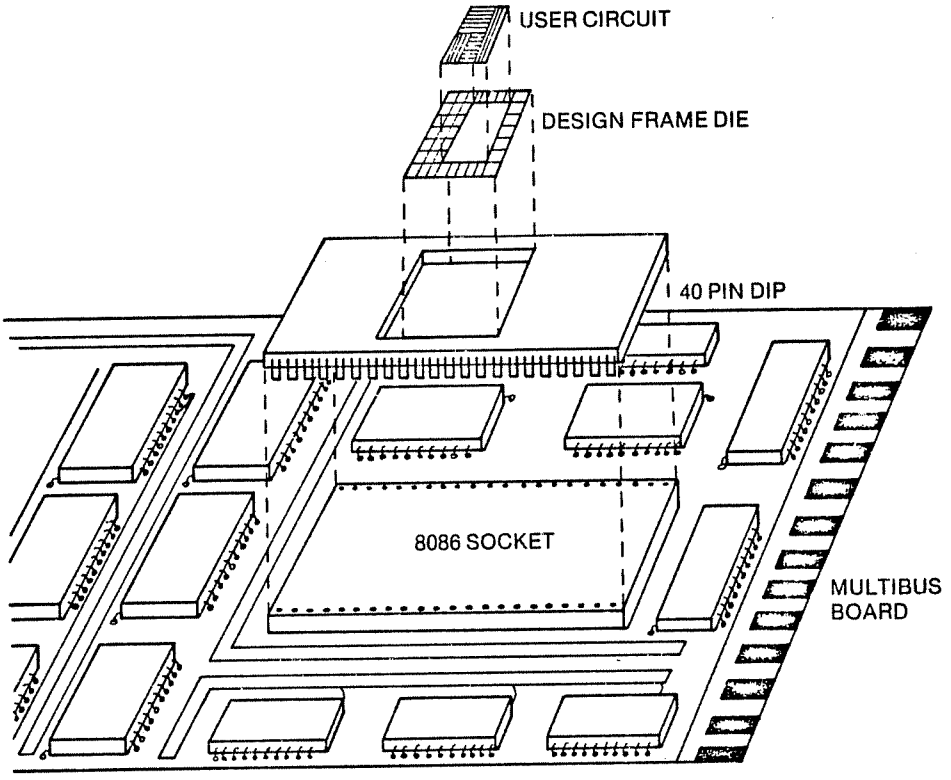

Figure 1. The Design Frame

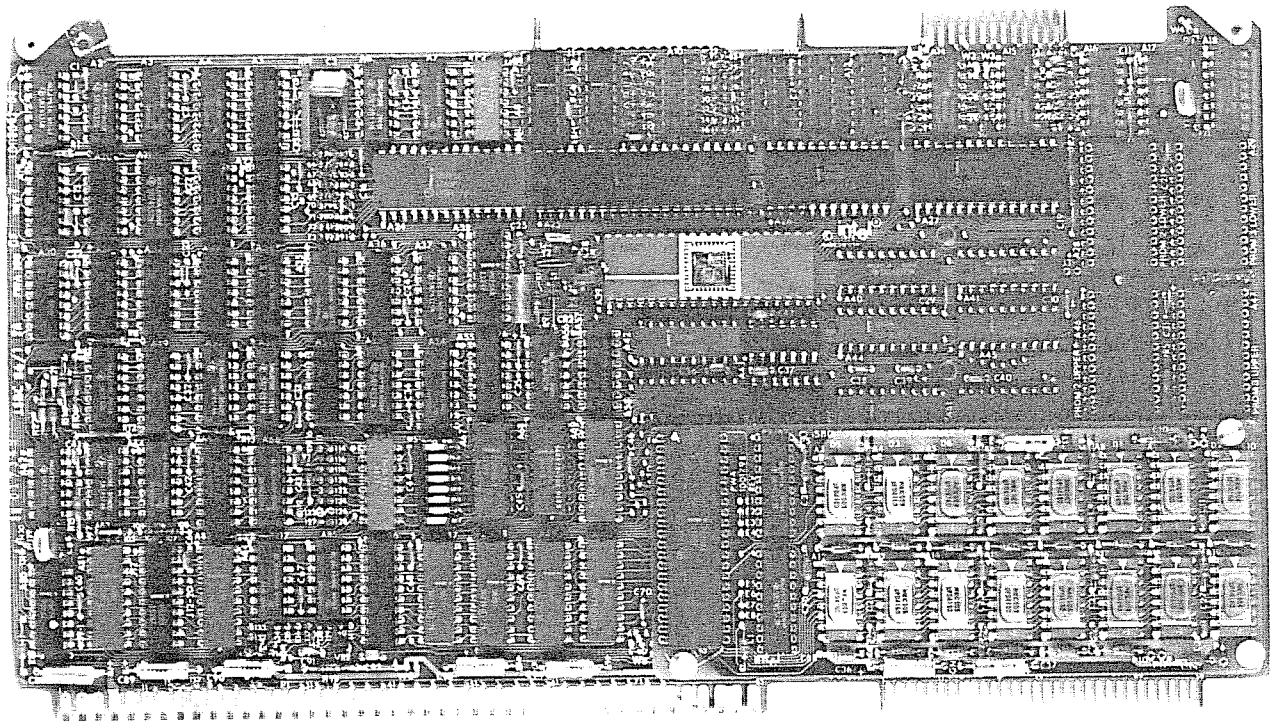

Figure 2. Photograph of Design Frame

describe how we tested the design frame with a simple test circuit. We discuss future directions in the conclusions.

3. Frame Interface

The user circuit/bus controller interface obeys a simple four cycle signaling scheme (see figure 3). A request is preceded by placing a memory or I/O device address in the interface's Address Register, data in the Data Register (if output), and setting the operation bits in the Status Register (read or write to memory or I/O device, or interrupt acknowledge). A request is initiated by setting the Request bit. When the bus controller recognizes the request, it initiates the operation, and shows that the operation has completed by setting the Acknowledge bit. The handshake is

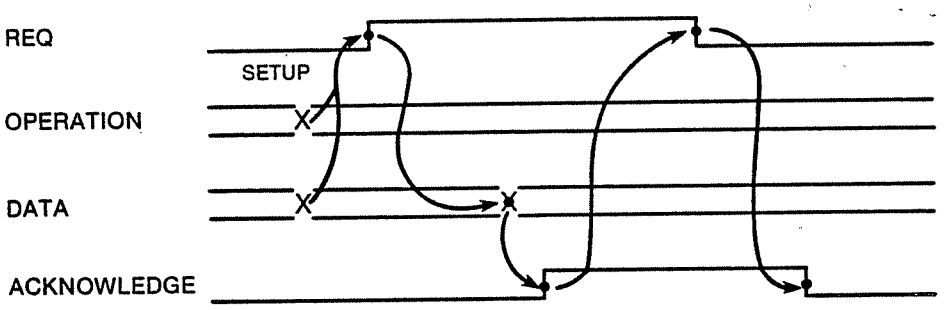

Figure 3. User/Controller Handshake

completed when the user circuit resets Request, causing the bus controller to reset Acknowledge. A new request can now be made. A register level description of the interface is given in figure 4.

4. Bus Controller Design

Since address and data are multiplexed on the 8086 processor bus, the 8086 Bus Interface Unit, residing with the Execution Unit on the CPU chip, must generate signals to latch addresses and to control the data bus transceivers. INTEL introduced the 8288 Bus Controller Chip to offload the generation of these signals from the processor (this frees some pins on the 8086 for other functions). A "maximum" mode 8086 is one that is configured for interface to the 8288. The latter generates all the necessary bus control signals, based on its decoding of the 8086 processor state. The state is encoded in the signals S2, S1, S0, which are available on pins of the 8086 connected to the 8288. An 8086 compatible bus controller need only generate these signals and provide address and data on the processor bus at the appropriate times. The 8288 generates the balance of the signals needed to control the bus, thus simplifying the design of the on-chip bus controller.

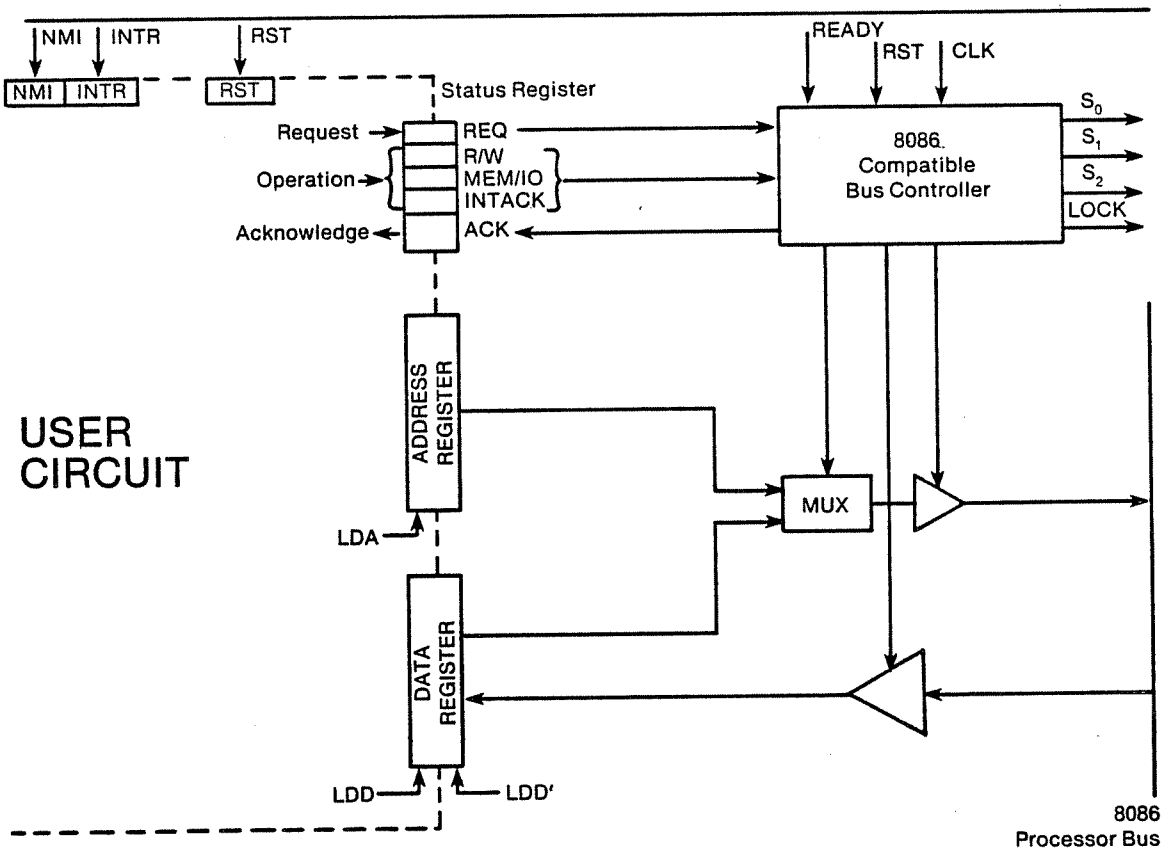

Figure 4. Register Diagram of Circuit/Controller Interface

However, several aspects of the signal timing of the 8086 complicate the signal generation. The detailed 8086 bus timing is as follows. The 8086 system clock is single phase. A major CPU cycle is divided into at least four minor cycles, each corresponding to a single period of the system clock. During cycle T1, the address is placed on the processor bus. The state of the processor is also revealed in the S2, S1, SO signals. During T2, output data is placed on the bus if in a write cycle, otherwise the processor bus is floated. A T3 cycle is inserted to give memory and I/O devices time to satisfy a read/write request. If a request cannot be performed in a single clock period, the 8086 Ready signal is held low by the addressed device, thus inserting wait states Tw between the T3 to T4 transition until the operation is complete. The 8086 alerts the 8288 bus controller to enter the T4 cycle by driving its

state signals high during the *last* Tw cycle (defined as the T3 or Tw cycle during which READY returns to the asserted state). On the transition to T4, input data is valid and is latched. The 8288 has to synchronize with the processor through changes in the processor status signals because it does not contain logic to interprete the READY signal. It enters T1 when at least one status signal goes low. It enters T4 when these signals are all driven high. Proper synchronization can only occur if the signals change within a critical timing window around the clock transitions. A simplified bus timing diagram is given in figure 5.

Interrupts are handled by the user circuit. It acknowledges an interrupt by requesting the bus controller to execute an interrupt acknowledge sequence. The sequence consists of two consecutive bus cycles. The bus LOCK signal is generated at the T1 to T2 transition during the first cycle, and is held through T1 of the second cycle. The INTA signal is generated during T2 and T3 of both cycles. During the

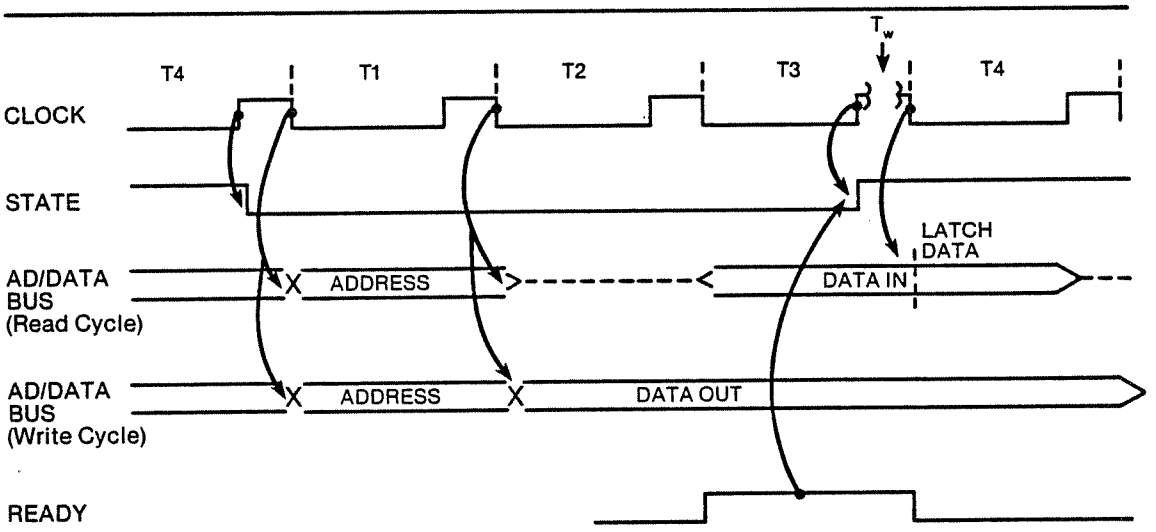

Figure 5. Simplified Timing Diagram for 8086 Compatible Bus Controller

second cycle, a byte of data from the interrupt controller is latched into the interface's data register. The user circuit is responsible for interpreting this byte.

The bus timing is complicated by the fact that some output signals appear on the falling edge of the clock, while others appear on the rising edge. We prefer to use the non-overlapping/two phase clocking scheme of [MEAD80], in which state machines read their inputs during phase 1 (PHI1) while all outputs become valid during phase 2 (PHI2).

These problems are handled by (1) generating a two phase clock from the system clock and (2) using a non-standard design for state machines. The timing diagrams for a derived two phase clock are shown in figure 6, and are based on the circuitry described in [MEAD80] and provided by the PadClockBar pad of the Stanford nMOS Cell Library [NEWK81]. PHI1 is high while the system clock is low, and PHI2 is high during the high time of the system clock. Signals appear at the rising or falling edge of the system clock by building the bus controller finite state machine with outputs that can change during either PHI1 or PHI2 (since the bus controller consists of a single finite state machine, there is no problem of reading outputs at the wrong time). The block diagram of the controller is given in figure 7. Outputs not conditioned by PHI1 will change close to the rising edge of the system clock, while those that are will only change close to the falling edge.

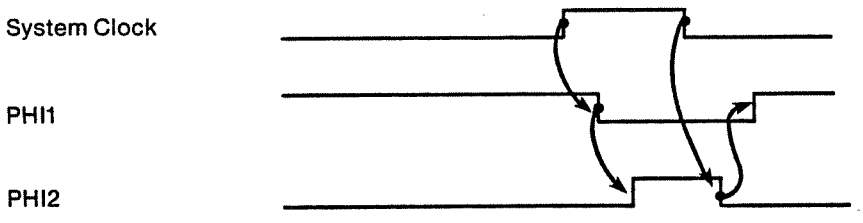

Figure 6. Derived Two Phase Clock

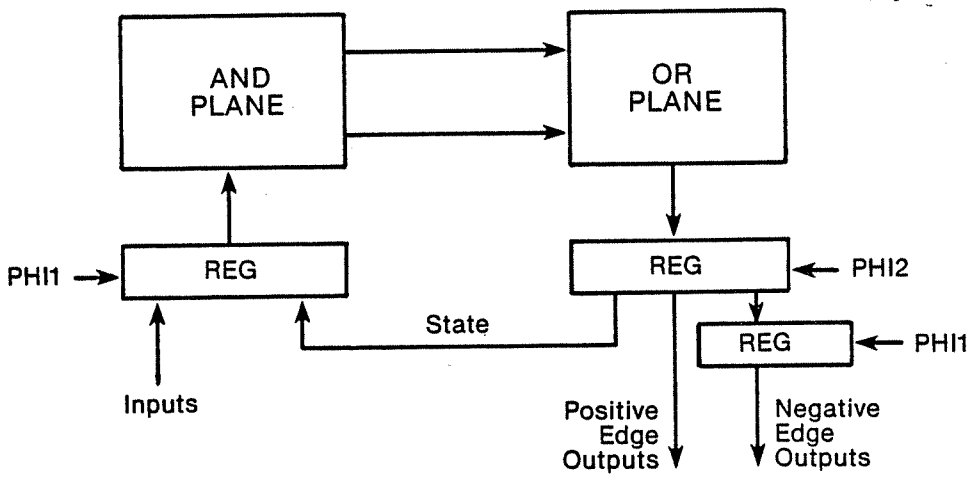

Figure 7. Bus Controller Block Diagram

The state diagram for the controller is given in figure 8. Since state changes occur at the positive edge of the system clock, signals appearing during cycle Ti are specified on the Ti-1 to Ti state transition. An up arrow shows that the signal appears on the rising edge, while a down arrow shows that it appears on the falling edge. Tr is the reset state, Ti, and Ta are idle states. Detailed timing is given in figure 9.

5. Bus Controller Implementation

The bus controller is implemented as a PLA-based finite state machine. The data, address, and control registers are conventional dynamic registers. The active area of the bus controller chip is 3.5 mm by 3.5 mm, however most of this is committed to the forty pads needed for 8086 pin compatibility. The area used by the bus controller and interface registers is modest. Because of the additional complexity of interrupt processing, it is not supported in the initial implementation of the bus controller. We

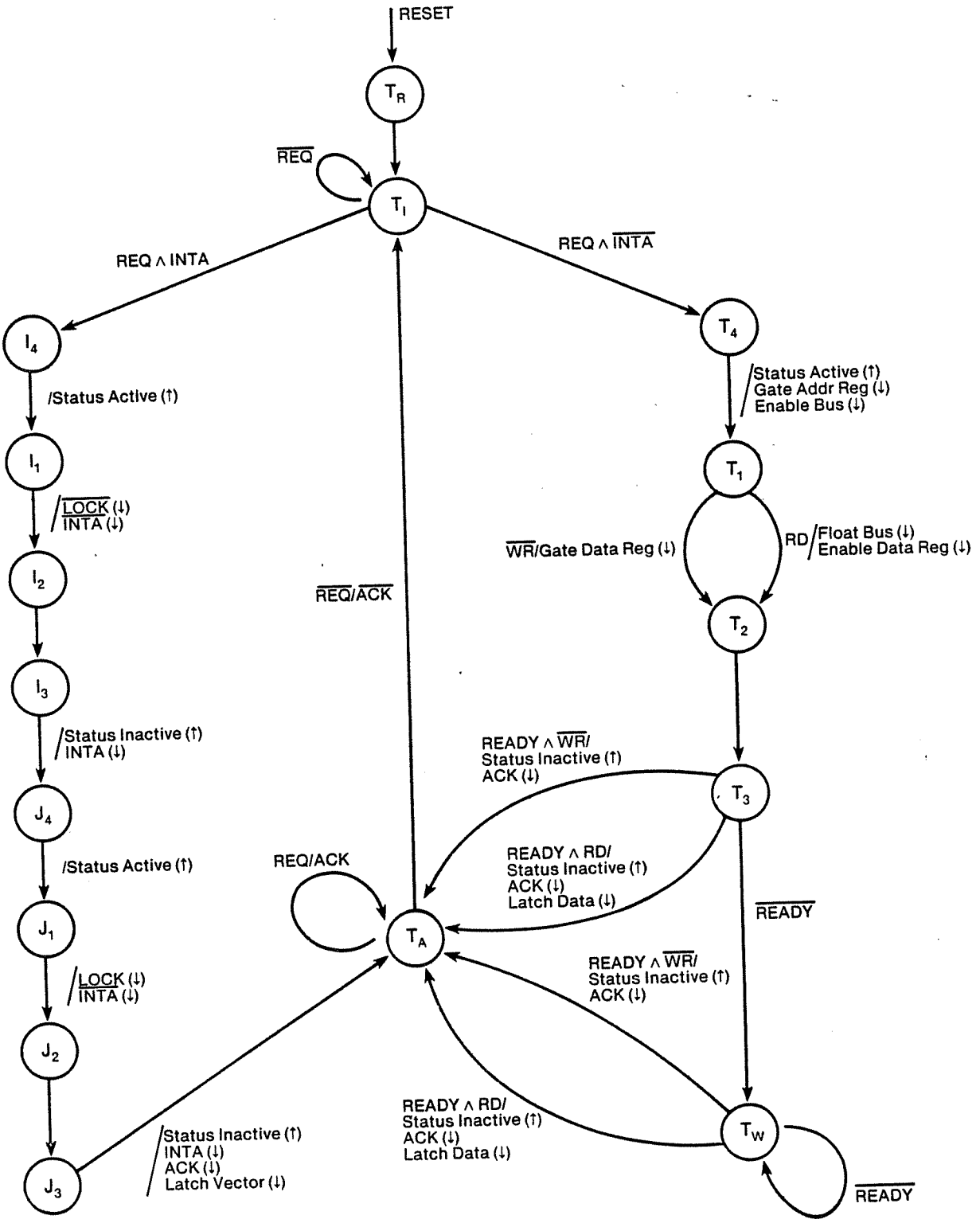

Figure 8. State Diagram for Bus Controller

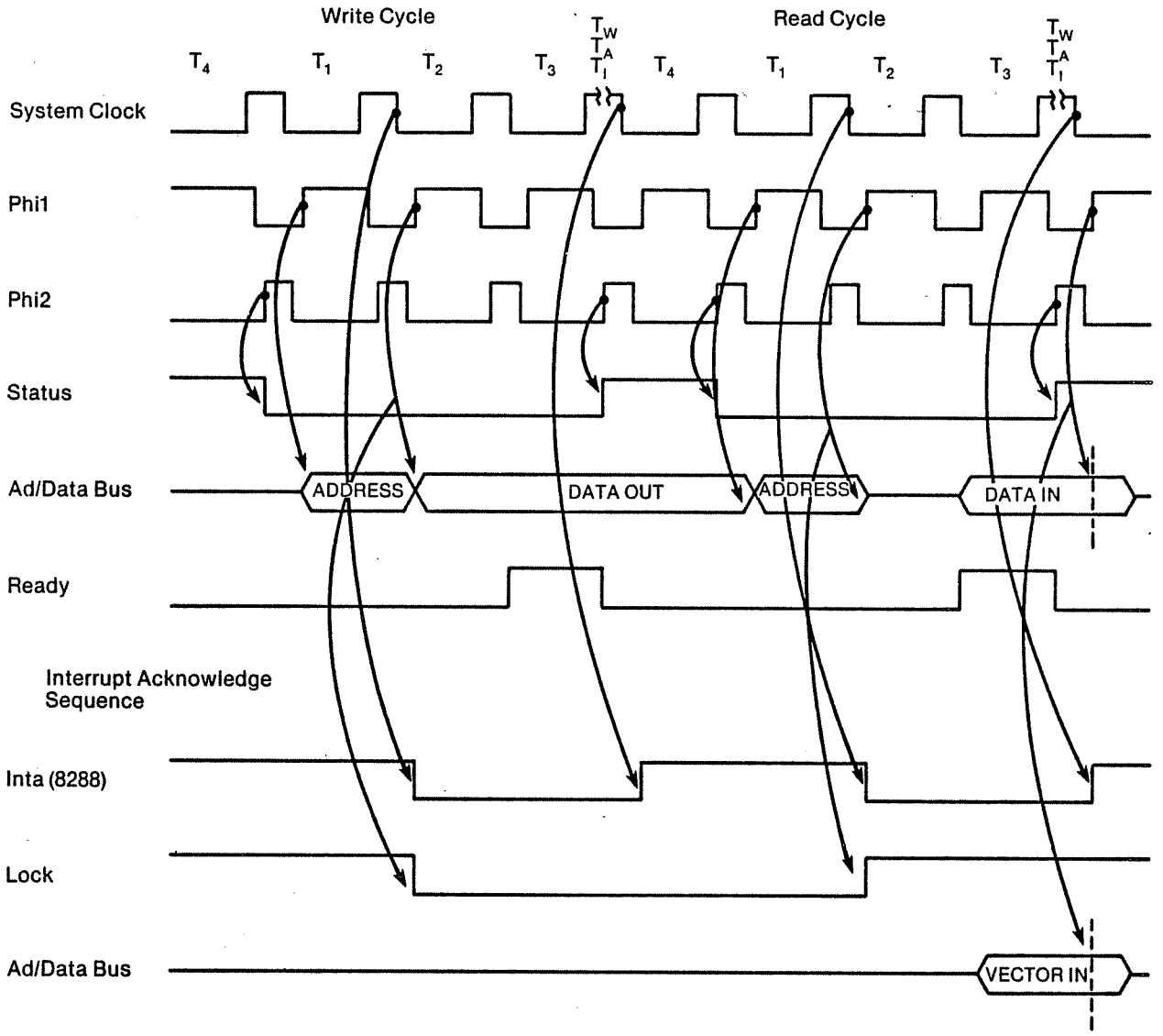

Figure 9. Detailed Timing Diagram for Read/Write/Interrupt Acknowledge Cycles

felt that the ability to read and write memory was a sufficient test. We have chosen a simple test circuit that reads a word from a fixed memory location and writes it back to another location. The bus controller was included on the die with the test circuit. A photomicrograph of the test circuit and bus controller is shown in figure 10.

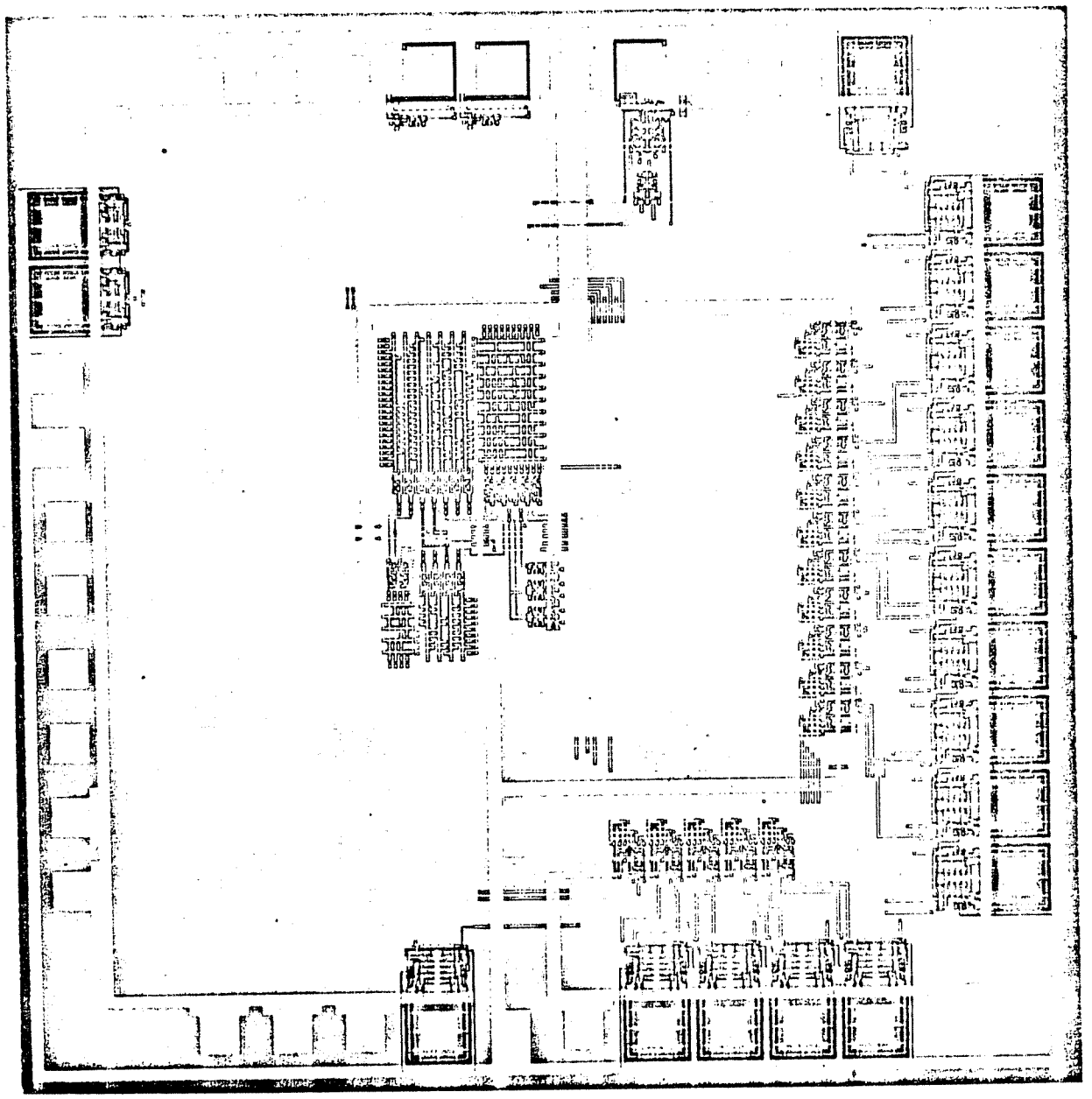

Figure 10. Bus Controller Photomicrograph

We implemented two versions of the bus controller. The only difference between them is the clock circuitry. In the first design, the standard clock pad from the cell library is used. In the second, instead of feeding back the opposite phase within the pad (this is done in the pad design to insure that the two phases are non-

overlapping), the feedback path is taken from the furthest extent in the circuit that the clock signal has to travel. Thus, the clock signal reaches all points in the circuit before returning to the pad, where the feedback causes it to go low, allowing the opposite phase to come high. This guarantees that clock phases will never overlap because of clock distribution delays. It is a standard technique used at Xerox PARC and was suggested to us by them.

During implementation, the major unanswered question was whether the bus controller circuit could operate fast enough to generate the state signals within the time critical windows (SPICE simulations were not available to us at the time). We were prepared to run the boards at slower speeds if necessary to lengthen the critical timing windows to eliminate the problem.

6. Design History and Testing of First Silicon

The concept of the design frame was formulated during late Fall of 1981. The bus controller was designed during December and January, and implementation was completed during the Spring of 1982. Designs were submitted through Xerox at the end of May, but an extensive simulation done after the submission revealed a bug in the programming of the PLA. This was corrected, and the design was resubmitted during the Summer. First silicon arrived at the beginning of September. Two chips with the standard clock pad and four chips with the modified pad were delivered.

Our curiosity got the better of us, and we tested the chips immediately by plugging one into an 8086 board configured as a slave. We used the monitor on a master 8086 board to load the slave input memory location with data. Fortunately for us, the first chip worked and the data value was transferred to the slave output memory location, which we were able to examine with the monitor.

After more thorough testing, we determined that of the six chips received, only one approximated correct behavior (the first one we tried!). The working chip had one data bit stuck at zero. We subsequently discovered a missing metal-polysilicon

contact that had eluded our simulations and connectivity checks (since these were ultimately connected to a tri-state pad, we had thought the problem was due to the simulator's inability to handle the high impedance state). Since this was connected to the data bit stuck at zero, the chip would have worked except for the missing contact. Running the remaining chips at a slower clock rate did not improve their behavior.

From these initial tests, we have verified that our design is able to run at the standard 5 Mhz clock rate of an 8086, while generating the bus control signals in the required time windows to read from and write to memory. Since the interrupt cycle is nothing more than two consecutive bus cycles, it should not be difficult to add this to the design. We believe the feedback clock technique to be instrumental in our success.

7. Conclusions and Future Directions

In this report, we have described the design, implementation, and testing of a "design frame" for VLSI System prototyping. The frame is based on a readily available microcomputer board and a custom designed bus controller circuit.

The idea of a standard frame is an important one. The frame provides many system capabilities needed by prototype circuits. System prototyping is completed more rapidly because these capabilities do not have to be "lashed up" from scratch for each design. Debugging at the systems level is also improved dramatically, since the behavior of the frame is well-understood. Since the frame includes a standard industry bus, working prototypes can be quickly surrounded with even more capabilities (more memory, secondary storage, etc.). Once shown to work, a prototype in its design frame can be used as a system immediately. A design frame also simplifies the task of interfacing a circuit subsystem to the rest of the system, since that interface is defined by the MULTIBUS specification. Obviously, not every VLSI prototype circuit can make use of our design frame. However, we believe that design frames

can be developed for generic classes of circuits. Ours is most appropriate for circuits that need memory and input/output operations, like conventional Von Neumann processors. High performance, bit-serial circuits for signal processing may require a very different design frame.

The iSBC 86/12A's dual ported memory provides the basis for a powerful VLSI prototype testing facility. A prototype board can be configured as a slave to another board with a standard 8086 processor. The master board, communicating through the MULTIBUS, can download data into the slave's memory and read data placed in the memory by the slave. This makes it possible to configure the prototype circuit with parameters, input data, microcode, instructions, and to examine diagnostic data produced by it. We intend to further explore the possibility of implementing a development system for the 8086 that will monitor the operation of circuits within our design frame.

We have completed the design and implementation of an initial bus controller circuit and have verified that it works. The next step is to interface more complicated prototype circuits to the bus controller, such as a simple microprocessor for image processing applications currently being designed at the University of Wisconsin. We are also interested in enhancing the design frame with testability features, such as scan in/scan out logic for the input/output pads and the interface's registers. The incorporation of such facilities is a good example of how a design frame can help to simplify the job of debugging prototype circuits. Another project underway is to build a design frame for direct interface to the MULTIBUS. While the bus protocol is simpler than the 8086 processor bus, the design is more complicated because a system on the bus can either be a slave or a master or both (the 8086 bus controller is the only master of the 8086 processor bus). A design frame that directly interfaces to the MULTIBUS will make it easier to create special purpose subsystems whose primary need is to communicate on a standard bus, but which do not need the other facilities of the 8086 board.