Title
Behavioral modeling of the Intel 8255A/8255A-5 programmable peripheral interface

Authors
Chaiyakul, Viraphol
Dutt, Nikil D.
Gajski, Daniel D.

Publication Date
1991-03-18

Peer reviewed

Behavioral Modeling of The Intel 8255A/8255A-5 Programmable Peripheral Interface

Viraphol Chaiyakul
Nikil D. Dutt
Daniel D.Gajski

Technical Report #91-27
March 18, 1991

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717
(714) 856-8059

viraphol@ics.uci.edu

Contents

List of Figures

1 Introduction

This report describes the behavioral modeling of the Intel 8255A, Programmable Peripheral Interface, which is designed for the use with Intel micro processors. We have used the Intel data book [Intel87] description as the primary source of information. The information provided in this data book includes the chip's functionality according to its input/output, timing and operational characteristics, and a functional block diagram. The report illustrates a modeling approach in detail using state-charts, flowcharts and VHDL. The resultant model was coded in VHDL and tested with the Vantage Analysis Systems VHDL simulator [Vant89].

This report is divided into 6 sections. Section 2 summarizes the functionalities and structural view of the Intel 8255A. Section 3 discusses the modeling approach. Section 4 describes the behavioral model of the chip using SpecCharts [VaNaGa], flowcharts and pseudo-code. Section 5 outlines the testing strategy for the resultant model, and a description of each test scenario. Appendix I contains detailed flowcharts for the model. Appendix II contains test vectors and test waveforms for those test scenarios describes in Section 5. And finally, Appendix III provides a listing of the actual VHDL code for our behavioral model of the Intel 8255A.

2 The Intel 8255A

The Intel 8255A is a general purpose programmable peripheral interface device designed for use with Intel micro processors. Its function is to interface peripheral equipment to the microcomputer system bus, as shown in Figure 1.

The Intel 8255A has 24 I/O pins which can be individually configured into 2 groups of 12 or 3 groups of 8 and used in 3 major modes of operation.

The configuration is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

2.1 Functional Description of the Intel 8255A

Figure 2 shows the block diagram for the 8255A. It contains three 8-bit ports(A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255A.

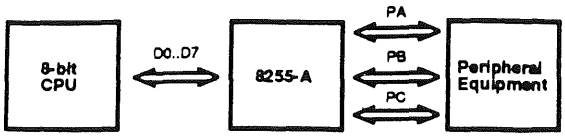
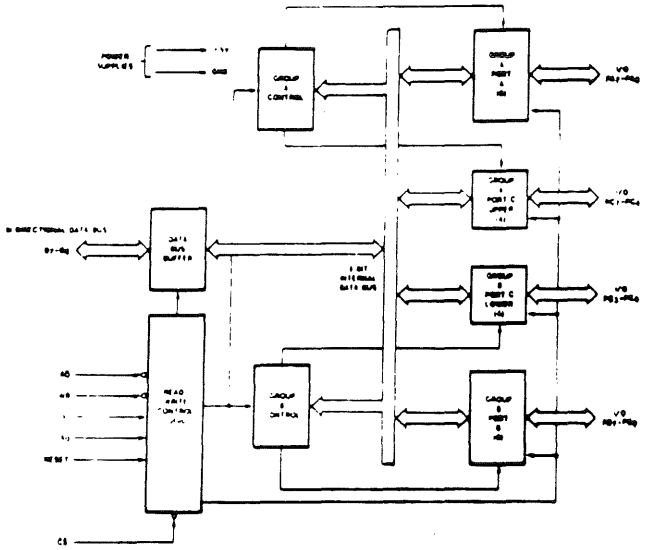
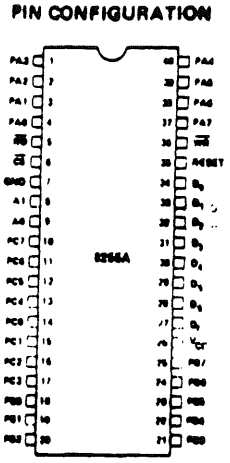


Figure 1: Typical Application for the 8255A.





PIN NAMES

D_7-D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

Figure 2: 8255A Block Diagram and Pin Configuration.

- PortA consists of one 8-bit data output latch/buffer and one 8-bit data input latch.
- PortB consists of one 8-bit data input/output latch/buffer and one 8-bit data input buffer.
- PortC consists of one 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.

These 24 I/O pins may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.

2.2 Structural Description of the Intel 8255A

- Data Bus Buffer

  This 3-state bidirectional 8-bit buffer is used to interface the 8255A to the system bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the micro processor. Control words and status information are also transferred through the data bus buffer.

- CS_BAR (chip select)

  A "low" on this input pin enables the communication between the 8255A and the micro processor.

- RD_BAR (read)

  A "low" on this input pin enables the 8255A to send the data or status information to the micro processor on the data bus. In essence, it allows the micro processor to "read from" the 8255A.

- WR_BAR (write)

  A "low" on this input pin enables the micro processor to write data or control words into the 8255A.

- A0_MODE and A1_MODE (port select 0 and port select 1)

  These input signals, in conjunctions with the RD_BAR and WR_BAR inputs, control the selection of one of the three ports or the control word registers. They are normally connected to the least significant bits of the address bus.

- RESET

  A "high" on this input clears the control registers and all ports (A,B,C) are set to the input mode.

- I/O PORTS (port A, B, and C)

  These ports can be configured individually in a wide variety of functional characteristics by the system software.

3 Modeling Approach

We have divided the modeling task into three major phases of incremental development. During the first phase, behavior of the Intel 8255A was described using state-charts, flowcharts and pseudo-code. Description of the Intel 8255A chip from Intel data book [Intel87] is used as a main source of specification. Since many aspects of the chip's functionality are not clearly described in the Intel data sheet, several assumptions have been made. More explanation on these assumptions can be found later in this section.

In the second phase, each state and its corresponding flowchart is then converted into VHDL code. The state is primarily modeled with process and block structures. The guard at each entry to each block signifies the conditions under which a state is to be entered.

In the final phase, the VHDL code is compiled, simulated, and its resultant waveforms are compared to those given in the Intel data book [Intel87].

3.1 Assumptions of Some Functionalities

The Intel data sheet description of the Intel 8255A was the only available source of specification we could locate. Most of the chip's functionalities can be extracted from the data sheet. But some of these functions are either vaguely described or are entirely missing. Hence, we made logical assumptions to fill these gaps in the behavior. These assumptions are listed below:

- Reset Operation

The data sheet does not clearly say what happens if the chip is reset while in the middle of a wait operation (e.g. waiting for handshaking signals). In our model, we assume that if a RESET signal is activated, the chip will abort any encountered waiting operations while trying to complete the current I/O operation. The chip then puts itself into the RESET state. Hence, in the scenario where the chip is waiting on a handshaking signal, the chip will abort the wait, finish the current operation and enter the RESET state.

To model this process, an additional check for the RESET signal is added to every waiting operation in every state. The condition of the wait should be satisfied if the RESET signal is activated. (See the actual VHDL code in the Appendix III for more detail).

- Chip Enable Operation

In our model, we assume that if the chip is disabled, communication between the 8255A and the micro processor will be disabled. However, in some modes, communication between the 8255A and external devices is allowed. For example, in mode 1 input (handshaking operation), latching of data from external devices is performed regardless of whether the chip is enabled or disabled. The data will remain in the latch until the chip (8255A) is enabled again. Similarly, during a handshaking output operation, if the chip is disabled after the data is latched from the micro processor data bus, the data will remain in the latch until the external device acknowledges the acceptance.

- Mode Select Operation

A new setting of the chip's configuration will only take effect at the beginning of any read/write cycle. In another word, if the chip is in the middle of a read/write operation

when its configuration is altered, the chip will maintain the old configuration until it completes that read/write cycle. Subsequently, it will be configured to the new configuration setting.

3.2 Treatment of Timing Behavior

Timing specifications of the Intel 8255A from the Intel data book [Intel87] represent physical characteristics of the real chip. All timing delays are coded as inertial delays in the final VHDL model. However timing constraints (e.g., setup time, hold time etc.) are controlled by VHDL assertions. Error messages are reported when timing constraints are violated.

3.3 Simulator Specifics

The VHDL model given in Appendix III has been successfully simulated under the Vantage Analysis Systems, version 1.08 [Vant89]. The waveforms given in Appendix II are direct screen dumps from the Sun workstation.

4 Behavioral Model of The Intel 8255A

The behavioral of the Intel 8255A can be modeled using SpecCharts [VaNaGa] consisting of 2 primary states: RESET state and ACTIVE state (Figure 3).

A brief description of each state is given below. Appendix I shows detailed flowcharts for each state of the SpecCharts in Figure 3.

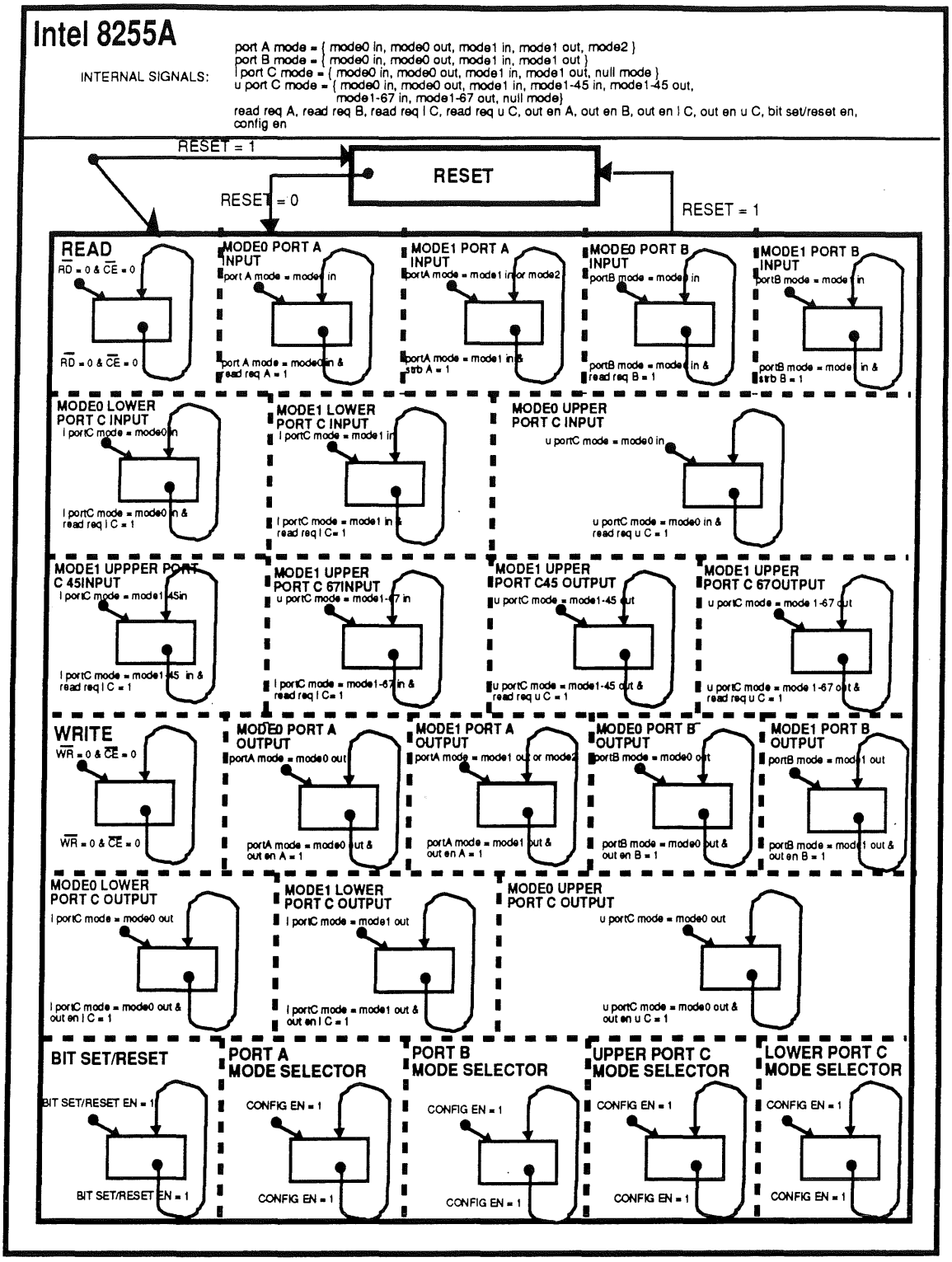


Figure 3: Intel 8255A SpecCharts

- RESET State

  This state performs the reset operation. It can be invoked at anytime by activating the reset signal. In this state, control registers are cleared and all ports (A,B,C) are set to the input mode 0. The chip will remain in this state until the reset signal is deactivated.

- ACTIVE State

  This state consists of 25 concurrent states. They are modeled concurrently because external devices can communicate with the 8255A through any of its 3 ports **simultaneously**. But since a port can only operate in one mode at a time, a *port mode* internal signal is assigned to each port as a control signal. The only exception is port C, which has two signals, since its upper and lower half can be configured differently. These control signals are described below.

  Note that among these 25 concurrent states, there is no special state to handle mode2 configuration of the chip. This is because we have modeled the configuration of mode2 as a combination of operations from other modes.

The model consists of several internal signals. These signals are used as control signals to determine the activation of each state in the model. Descriptions for these signals are given below:

- port A mode

  This signal indicates the mode configuration for port A. The *port A mode* signal can be set to only one value/mode as specified in Figure 3 (mode 0 input, mode 0 output,

mode 1 input, mode 1 output, mode 2). Hence, I/O operations from/to port A need to consult *port A mode* signal before initiating the operation.

- port B mode

  This signal indicates the mode configuration for port B. The *port B mode* signal can be set to only one value/mode as specified in Figure 3 (mode 0 input, mode 0 output, mode 1 input, mode 1 output). Hence, I/O operations from/to port B need to consult *port B mode* signal before initiating the operation.

- l(ower) port C mode

  This signal indicates the mode configuration for the lower 4 bits (C0-3) of port C. The *l port C mode* signal can be set to only one value/mode as specified in Figure 3 (mode 0 input, mode 0 output, mode 1 input, mode 1 output, null mode). *null mode* occurs when all the lower 4 bits of port C are used as handshaking signals (e.g. combination of mode 1). Hence, I/O operations from/to lower 4 bits of port C need to consult the *l port C mode* signal before initiating the operation.

- u(pper) port C mode

  This signal indicates the mode configuration for the upper 4 bits (C4-7) of port C. The *u port C mode* signal can be set to only value/mode as specified in Figure 3 (mode 0 input, mode 0 output, mode 1-45 input (only bit 4 and 5 are activated), mode 1-45 output, mode 1-67 input (only bit 6 and 7 are activated), mode 1-67 output, null mode). *null mode* occurs when all upper 4 bits of port C are used as handshaking signals (e.g. mode 2). Hence, I/O operations from/to upper 4 bits of port C need to consult the *u port C mode* signal before initiating the operation.

- read req(uest) (port) A

  This signal is controlled by the READ process. The signal is activated when the micro processor issues a read command. When active, the *read req A* signal indicates the transfer of data from port A onto the micro processor data bus.

- read req(uest) (port) B

  This signal is controlled by the READ process. The signal is activated when the micro processor issues a read command. When active, the *read req B* signal indicates the transfer of data from port B onto the micro processor data bus.

- read req(uest) l(lower) (port) C

  This signal is controlled by the READ process. The signal is activated when the micro processor issues a read command. When active, the *read req l C* signal indicates the transfer of data from the lower 4 bits (C0-C3) of port C onto the micro processor data bus.

- read req(uest) u(pper) (port) C

  This signal is controlled by the READ process. The signal is activated when the micro processor issues a read command. When active, the *read req u C* signal indicates the transfer of data from the upper 4 bits (C4-C7) of port C onto the micro processor data bus.

- out en(able) (port) A

  This signal is controlled by the WRITE process. The signal is activated when the micro processor issues a write command. When active, the *out en A* indicates the transfer

of data from the micro processor data bus into the internal latch of the 8255A or into the external device data bus through port A, depending on the port configuration.

- out en(able) (port) B

  This signal is controlled by the WRITE process. The signal is activated when the micro processor issues a write command. When active, the *out en B* indicates the transfer of data from the micro processor data bus into the internal latch of the 8255A or into the external device data bus through port B, depending on the port configuration.

- out en(able) l(ower) (port) C

  This signal is controlled by the WRITE process. The signal is activated when the micro processor issues a write command. When active, the *out en l C* indicates the transfer of data from the micro processor data bus into the internal latch of the 8255A or into the external device data bus through the lower 4 bits (C0-C3) of port C, depending on the port configuration.

- out en(able) u(pper) (port) C

  This signal is controlled by the WRITE process. The signal is activated when the micro processor issues a write command. When active, *out en u C* indicates the transfer of data from the micro processor data bus into the internal latch of the 8255A or into the external device data bus through the upper 4 bits (C4-C7) of port C, depending on the port configuration.

- bit set/reset en(able)

  This signal is controlled by the WRITE process. The signal is activated when the micro processor requests any of the eight bits of port C to be set or reset. This feature

reduces software requirements in control-based applications.

- config(uration) en(able)

  This signal is controlled by the WRITE process. The signal is activated when the micro processor requires alteration of the chip's configuration.

4.1 ACTIVE State

A brief discussion of the 25 concurrent states within the ACTIVE state are provided as follow:

- READ State

  This state is responsible for controlling read operations. During a read cycle, the read enable signal is activated from this state to the port specified by the micro processor (through the signals A0 and A1). This results in the transfer of data from the internal latch or external devices onto the micro processor data bus.

- MODE 0 PORT A INPUT State

  This state is activated if port A is configured to *mode 0 input* and a read request signal is received. Data from port A is transfered onto the micro processor data bus on completion of this state.

- MODE 0 PORT B INPUT State

  This state is activated if port B is configured to *mode 0 input* and a read request signal is received. Data from port B is transfered onto the micro processor data bus on completion of this state.

- MODE 1 PORT A INPUT State

  This state is activated if port A is configured to *mode 1 input* or *mode2*, and strobe signal is received from the external device. This state performs the handshaking input operation with the external device that is connected to port A.

- MODE 1 PORT B INPUT State

  This state is activated if port B is configured to *mode 1 input* and strobe signal is received from the external device. This state performs handshaking input operation with the external device that is connected to port B.

- MODE 0 LOWER PORT C INPUT State

  This state is activated if the lower port C is configured to *mode 0 input* and a read request signal is received. The lower 4 bits of data from port C (C0-C3) are transfered onto the micro processor data bus on completion of this state.

- MODE 1 LOWER PORT C INPUT State

  This state is activated if lower the port C is configured to *mode 1 input* and a read request signal is received. Data from bits 0-2 of port C are transfered onto the micro processor data bus on completion of this state (this state happens in mode2 configuration).

- MODE 0 UPPER PORT C INPUT State

  This state is activated if the upper port C is configured to *mode 0 input* and a read request signal is received. The upper 4 bits data from port C (C4-C7) are transfered onto the micro processor data bus on completion of this state.

- MODE 1 UPPER PORT C-45 INPUT State

This state is activated if the upper port C is configured to *mode 1 bit 4-5 input* and a read request signal is received. Data from bits 4 and 5 of port C are transfered onto the micro processor data bus on completion of this state.

- MODE 1 UPPER PORT C-67 INPUT State

  This state is activated if the upper port C is configured to *mode 1 bit 6-7 input* and a read request signal is received. Data from bits 6 and 7 of port C are transfered onto the micro processor data bus on completion of this state.

- WRITE State

  This state is responsible for controlling write operations. During a write cycle, *out en(able)* signal is activated from this state to the port specified by the micro processor (through A0 and A1 signals). This results in the transfer of data from the micro processor data bus into the 8255A latches or onto the external device data bus.

  In the case of a mode selection operation, the configuration enable signal will be activated from this state. And in the case of a bit set/reset operation, the bit set/reset enable signal will be activated from this state.

- BIT SET/RESET State

  This state is responsible for setting and resetting any of the eight bits of port C and interrupt flip-flops. It is activated by the *bit set/reset en(able)* signal which is controlled by the WRITE state.

- PORT A MODE SELECTOR State

  This state waits for the *config(uration) en(able)* signal to be activated by the WRITE state. Upon the activation, it reads the data bus and sets the internal control signal

port A mode to the mode which is specified by the configuration given on the micro processor data bus.

- PORT B MODE SELECTOR State

  This state waits for the *config(uration) en(able)* signal to be activated by the WRITE state. Upon the activation, it reads the data bus and sets the internal control signal *port B mode* to the mode which is specified by the configuration given on the micro processor data bus.

- UPPER PORT C MODE SELECTOR State

  This state waits for the *config(uration) en(able)* signal to be activated by the WRITE state. Upon the activation, it reads the data bus and sets the internal control signal *u(pper) port C mode* to the mode which is specified by the configuration given on the micro processor data bus.

- LOWER PORT C MODE SELECTOR State

  This state waits for the *config(uration) en(able)* signal to be activated by the WRITE state. Upon the activation, it reads the data bus and sets the internal control signal *l(ower) port C mode* to the mode which is specified by the configuration given on the micro processor data bus.

- MODE 0 PORT A OUTPUT State

  This state is activated if the control *port A mode* signal is set to *mode 0 output* and *out en(able)* signal for port A is activated. Subsequently, it transfers data from the micro processor data bus onto the external device bus connected to port A.

- MODE 0 PORT B OUTPUT State

This state is activated if the control *port B mode* signal is set to *mode 0 output* and *out en(able)* signal for port B is activated. It then transfers data from the micro processor data bus onto the external device bus connected to port B.

- MODE 1 PORT A OUTPUT State

  This state is activated if the control *port A mode* signal is set to *mode 1 output* or *mode 2*, and *out en(able)* signal for port A is activated. This state performs handshaking communication with the external device.

- MODE 1 PORT B OUTPUT State

  This state is activated if the control *port B mode* signal is set to *mode 1 output* and *out en(able)* signal for port B is activated. This state performs handshaking communication with the external device.

- MODE 0 LOWER PORT C OUTPUT State

  This state is activated if the control *l(ower) port C* signal is set to *mode 0 output* and *out en(able)* signal for lower port C is activated. It then transfers data from the micro processor data bus onto the external device connected to lower 4 bits of port C.

- MODE 1 LOWER PORT C OUTPUT State

  This state is activated if the control *l(ower) port C* signal is set to *mode 1 output* and *out en(able)* signal for lower port C is activated. It then transfers data in bit 0-2 of the micro processor data bus onto bit 0-2 of the external device connected to port C (this situation happens in mode2 configuration).

- MODE 0 UPPER PORT C OUTPUT State

This state is activated if the control *l(ower) port C* signal is set to *mode 1 output* and *out en(able)* signal for lower port C is activated. It then transfers lower 4 bits data from the micro processor data bus onto the external device which is connected to port C.

- MODE 1 UPPER PORT C-45 OUTPUT State

  This state is activated if the control *u(pper) port C* signal is set to *mode 1 C-45 output* and *out en(able)* signal for upper port C is activated. It then transfers data in bits 4 and 5 of the micro processor data bus onto bits 4 and 5 of port C.

- MODE 1 UPPER PORT C-67 OUTPUT State

  This state is activated if the control *u(pper) port C* signal is set to *mode 1 C-67 output* and *out en(able)* signal for upper port C is activated. It then it transfers data in bits 6 and 7 of the micro processor data bus onto bits 6 and 7 of port C.

4.2 8255 Operational Behavior

A typical sequence of operations for the Intel 8255A would involve enabling the chip (CE_BAR = 0) and then resetting the chip (RESET = 1). Subsequently, data can be read from or written to external ports according to the current port configuration.

For data to be written to external ports from the micro processor, the following sequence of events occurs: The micro processor places the data on the data bus and sets the control signals specifying which port is to be written into. Then, the micro processor sets the WRITE_BAR signal low, which causes data to be latched into the internal registers. At

the same time as when WRITE_BAR goes low, the chip enters Write state, decodes the controlled signals and sends the *out en(able)* signal to the port which is specified by the controlled signals. This activates one of the concurrent OUPUT states to perform its function depending on the current mode setting.

For a data to be read from external ports into the micro processor data bus, a sequence of the following events occur: The micro processor gives a combination of control signals which specify which port is to be read from. Then it places a low signal on the READ_BAR line. Upon sensing the low on READ_BAR, the chip enters the Read State, decodes the port control signals and sends a *read req(uest)* which wakes up one of the concurrent INPUT States to perform the read operation.

To change the configuration of the chip, the micro processor places a control word on the data bus and invokes the Write process by giving low signal on the WRITE_BAR line. In the Write state the chip decodes the control words and detects a mode changing operation. This enables the MODE SELECTOR states (by activating the *config(uration) en(able)* signal) to change the mode according to the given control word.

To set or reset the interrupt masking register or handshaking signals, the micro processor places a control word on the micro processor data bus and set a low signal on the WRITE_BAR line. When WRITE_BAR goes low, the chip enters the Write state. If it detects the control word to be a Bit Set/Reset operation, it enables the *bit set/reset en(able)* signal and thereby brings itself into the BIT SET/RESET State. In this state, the signals and registers are set/reset depending on the given control word.

5 Testing

This section describes seven typical operational scenarios for the Intel 8255A chip. These test scenarios are derived from typical operational sequences specified in the data book description of the 8255A [Intel87]. Each of these scenarios (labeled as "Example") is accompanied by a test vector, in the Vantage Analysis Systems' simulator format, and waveforms showing the result of the simulation on the Vantage Analysis Systems VHDL Simulator[Vant89]. (Test vectors and waveforms can be found in Appendix II.)

5.1 Example 1

In this example all ports are set to mode 0. Then a sequence of data is set on the data bus to be written to each of the 3 ports in the following sequence:

Data Bus $\longrightarrow$ Port C

Data Bus $\longrightarrow$ Port B

Data Bus $\longrightarrow$ Port A

The test vector for this example is provided in Figure 30 and its corresponded wave form is given in Figure 31.

5.2 Example 2

In this example all ports are set to Mode 0. Then a sequence of data is read in from each of the 3 ports in the following sequence:

Port A $\longrightarrow$ Data Bus

Port B $\longrightarrow$ Data Bus

Port C $\longrightarrow$ Data Bus

The test vector for this example is provided in Figure 32 and its corresponded wave form is given in Figure 33.

5.3 Example 3

In this example the test vectors are set up to test the Mode 1 read operation of Port A.

Port A $\longrightarrow$ Data Bus (MODE 1)

The test vector for this example is provided in Figure 34 and its corresponded wave form is given in Figure 35.

5.4 Example 4

In this example the test vectors are set up to test the Mode 1 write operation of Port A.

Data Bus $\longrightarrow$ Port A (MODE 1)

The test vector for this example is provided in Figure 36 and its corresponded wave form is given in Figure 37.

5.5 Example 5

In this example the test vectors are set up to test the Mode 1 read operation of Port B.

Port B $\longrightarrow$ Data Bus (MODE 1)

The test vector for this example is provided in Figure 38 and its corresponded wave form is given in Figure 39.

5.6 Example 6

In this example the test vectors are set up to test the Mode 1 write operation of Port B.

Data Bus $\longrightarrow$ Port B (MODE 1)

The test vector for this example is provided in Figure 40 and its corresponded wave form is given in Figure 41.

5.7 Example 7

In this example the test vectors are set up to test both read and write operations in Mode 2 of Port A, in the following sequence:

Data Bus $\longrightarrow$ Port A (MODE 2)

Port A $\longrightarrow$ Data Bus (MODE 2)

The test vector for this example is provided in Figure 42 and its corresponded wave form is given in Figure 43.

6 Summary

This report described the behavioral model of a commercial chip: the Intel 8255A. The modeling was derived from the Intel data book description as the initial specification. The behavioral model was developed using SpecCharts, flowcharts and pseudo-code. Subsequently, these flowcharts were coded in VHDL and tested on a commercial VHDL simulator (Vantage Analysis Systems). To verify the correctness of the behavior, a set of seven typical operational test cases were used as stimuli in the simulation and resultant waveforms were compared to those given in the Intel data book.

7 Acknowledgements

The authors of this report would like to thank Sanjiv Narayan, Joe Lis, and Rajesh Gupta for their help. We gratefully acknowledge the support of SRC under contract 90-DJ-146 and NSF under grant MIP-9009239.

8 References

[Intel87] Intel data book, November 1987, pp. 6-63 to 6-86, Order Number: 231308-002.

[Vant89] Vantage Analysis Systems, Inc, Fremont, CA 1989.

[VaNaGa] Vahid, F., Narayan, S., and Gajski, D.D., "*SpecCharts: A Language for System Level Specification and Synthesis*", University of California, Irvine, Technical report 90-19, July 1990.

[VHDL87] *IEEE Standard VHDL Language Reference Manual*, IEEE, 1987.

9 Appendix I

This appendix contains detailed flowcharts for behavior model of the Intel 8255A. Explanation for each of this flowchart is given in Section 4.

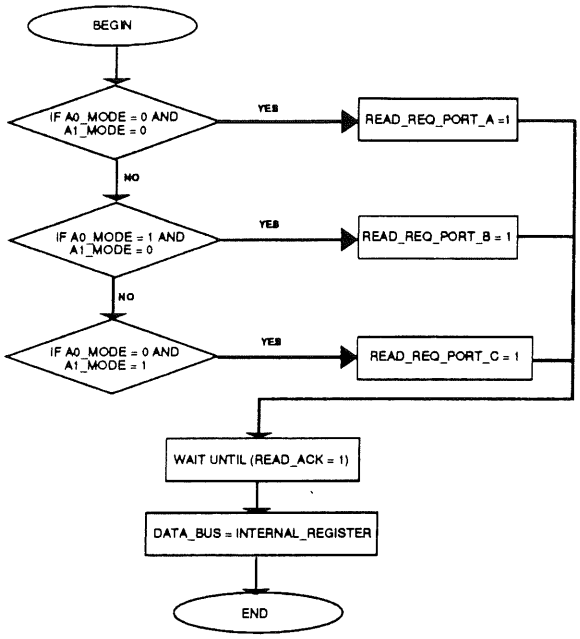


Figure 4: Flowchart For READ State

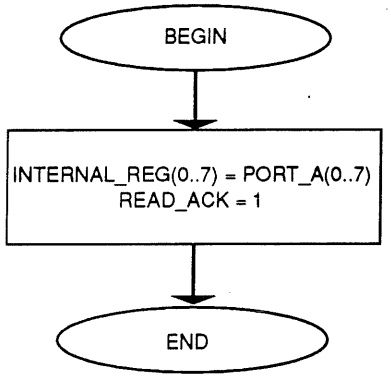


Figure 5: Flowchart For MODE 0 PORT A INPUT State

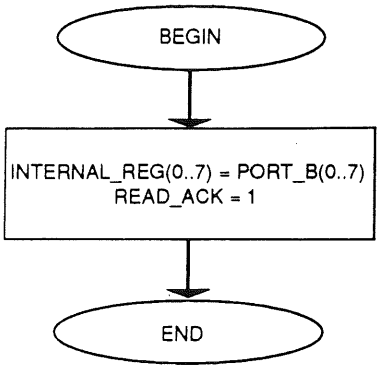


Figure 6: Flowchart For MODE 0 PORT B INPUT State

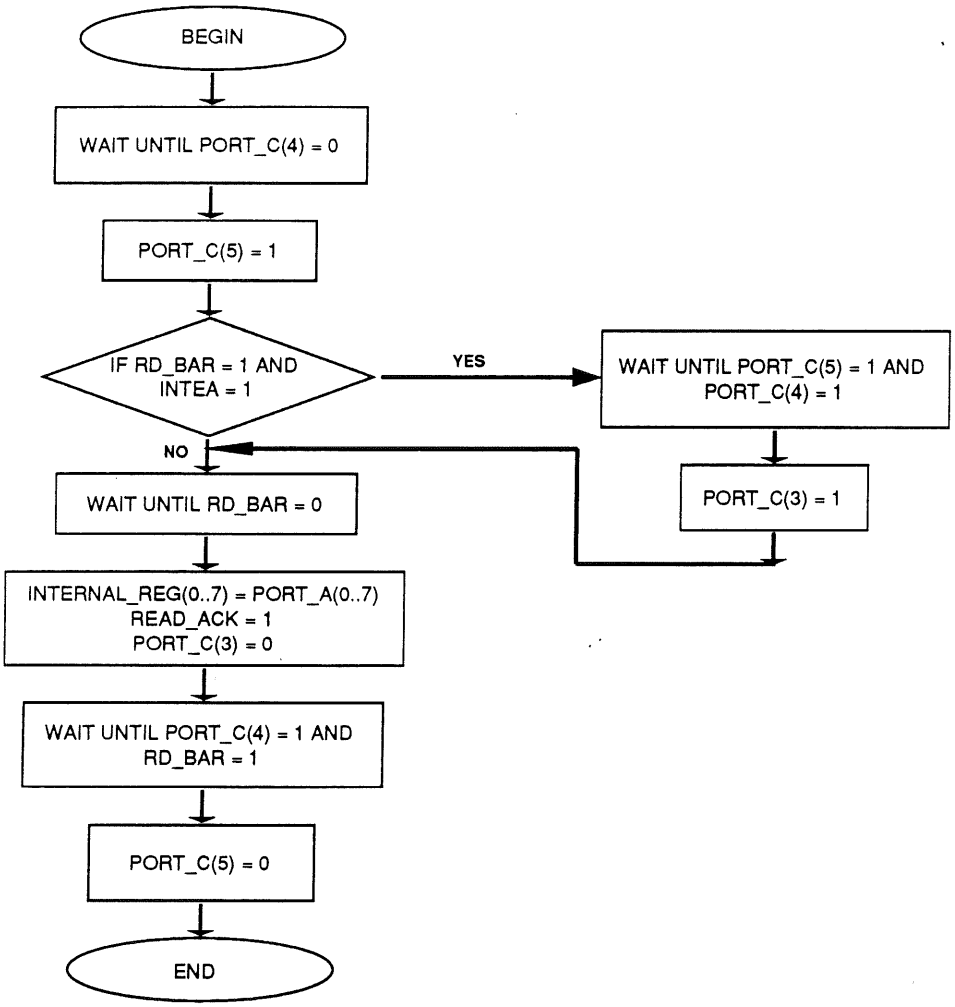


Figure 7: Flowchart For MODE 1 PORT A INPUT State

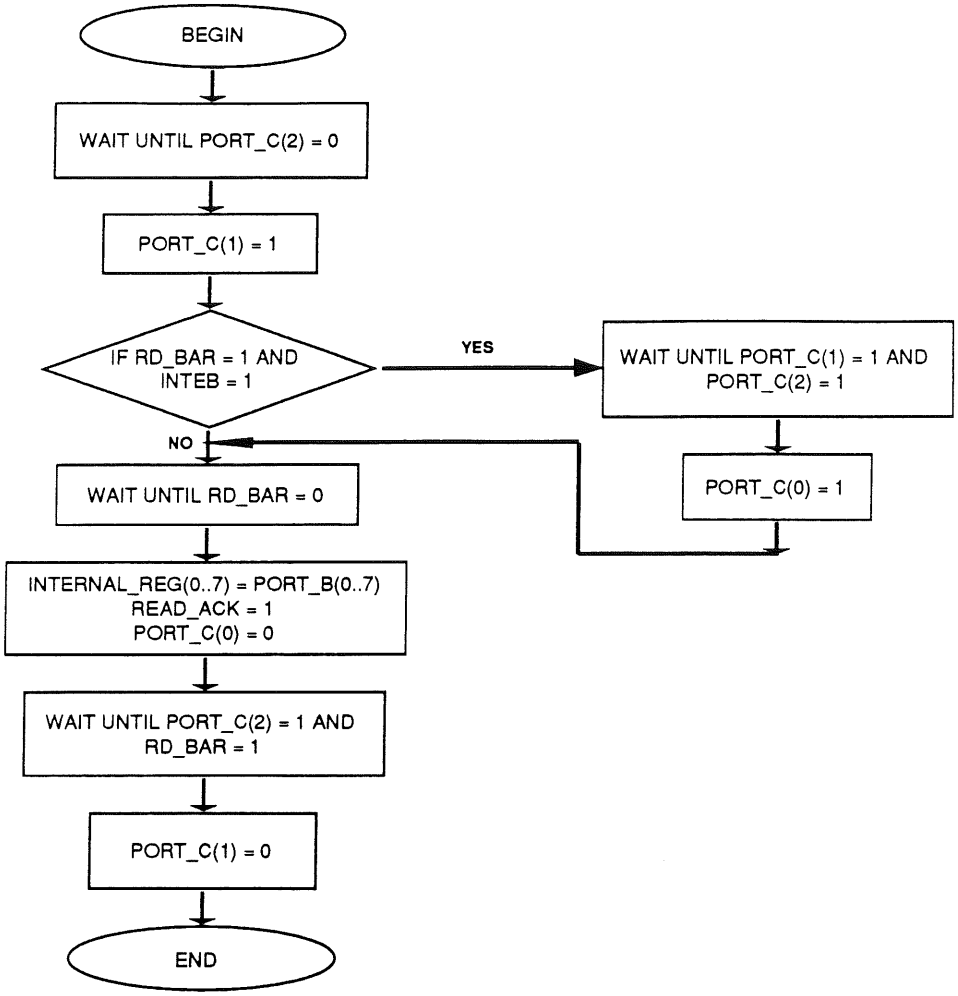


Figure 8: Flowchart For MODE 1 PORT B INPUT State

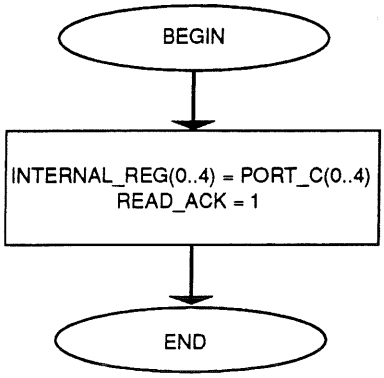


Figure 9: Flowchart For MODE 0 LOWER PORT C INPUT State

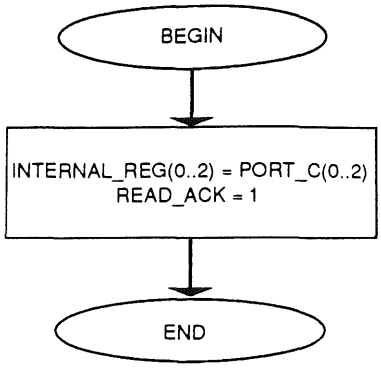


Figure 10: Flowchart For MODE 1 LOWER PORT C INPUT State

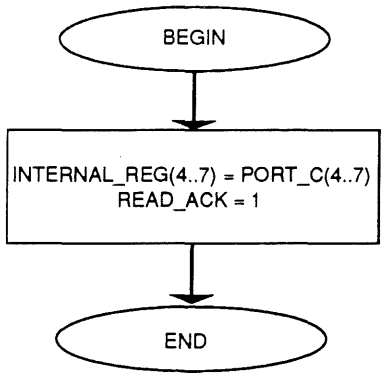


Figure 11: Flowchart For MODE 0 UPPER PORT C INPUT State

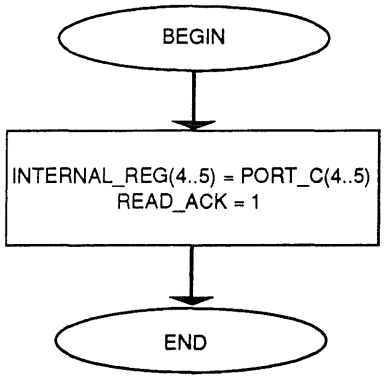


Figure 12: Flowchart For MODE 1 UPPER PORT C-45 INPUT State

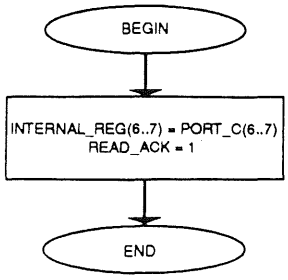


Figure 13: Flowchart For MODE 1 UPPER PORT C-67 INPUT State

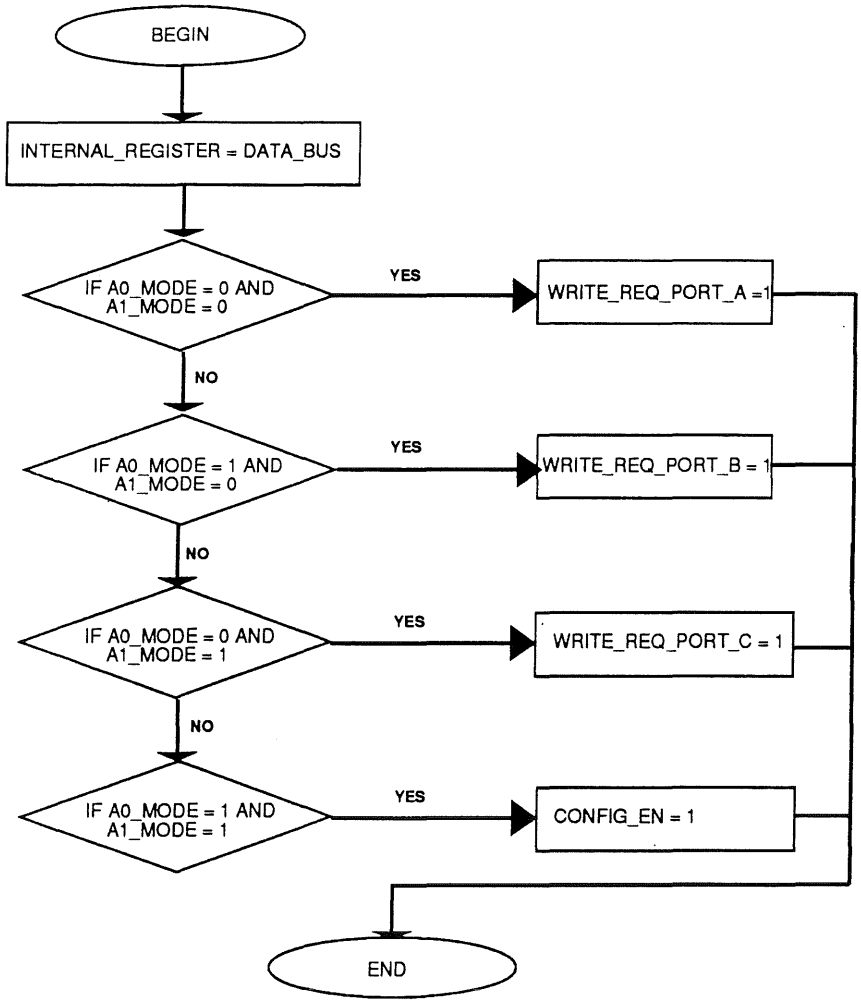


Figure 14: Flowchart For WRITE State

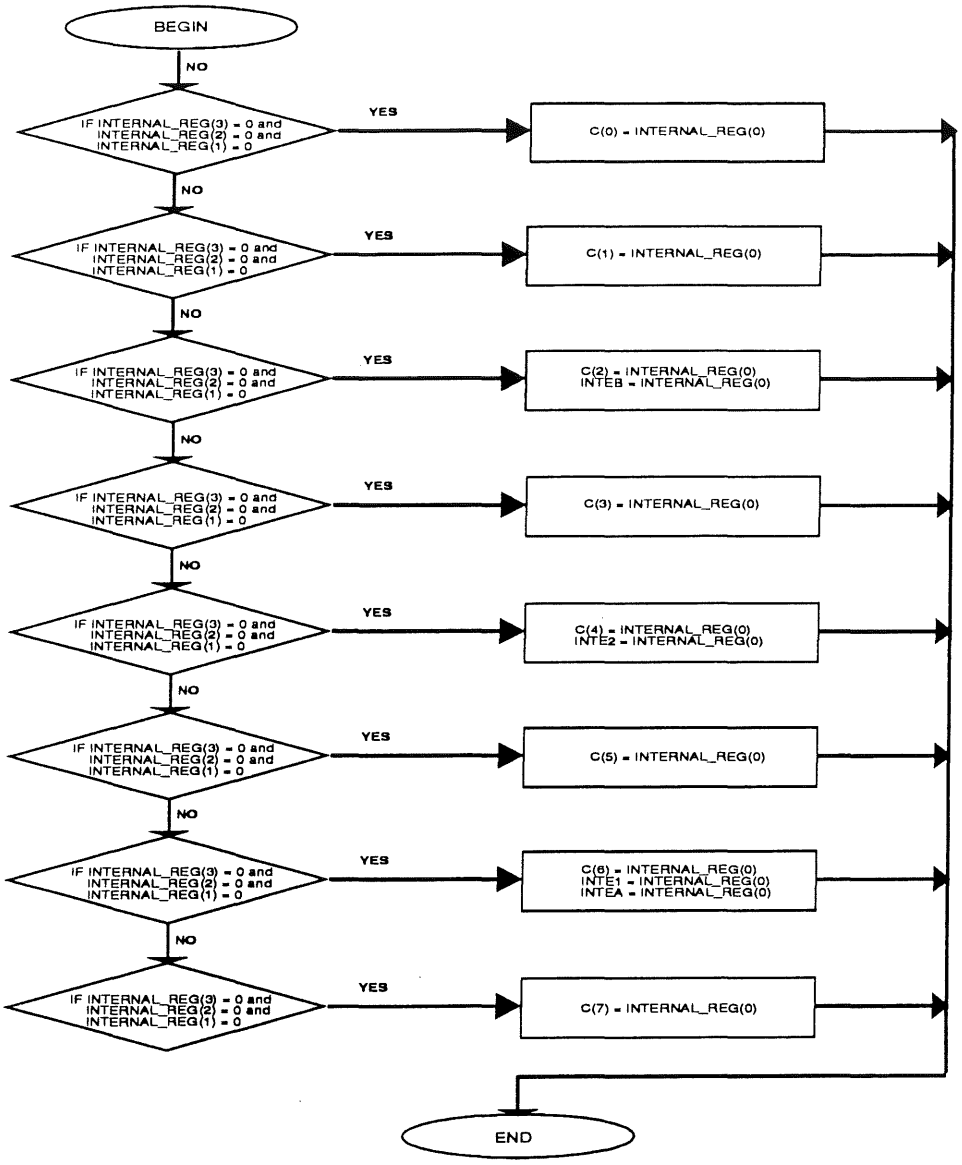


Figure 15: Flowchart For BIT SET State

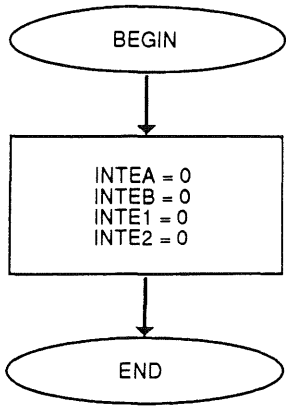


Figure 16: Flowchart For BIT RESET State

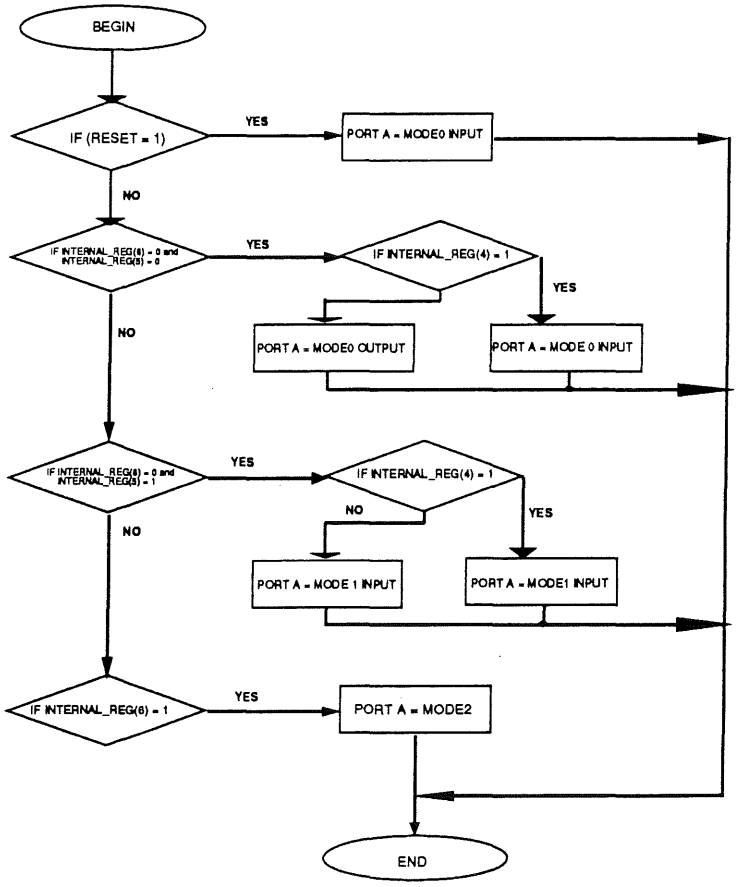


Figure 17: Flowchart For PORT A MODE SELECTOR State

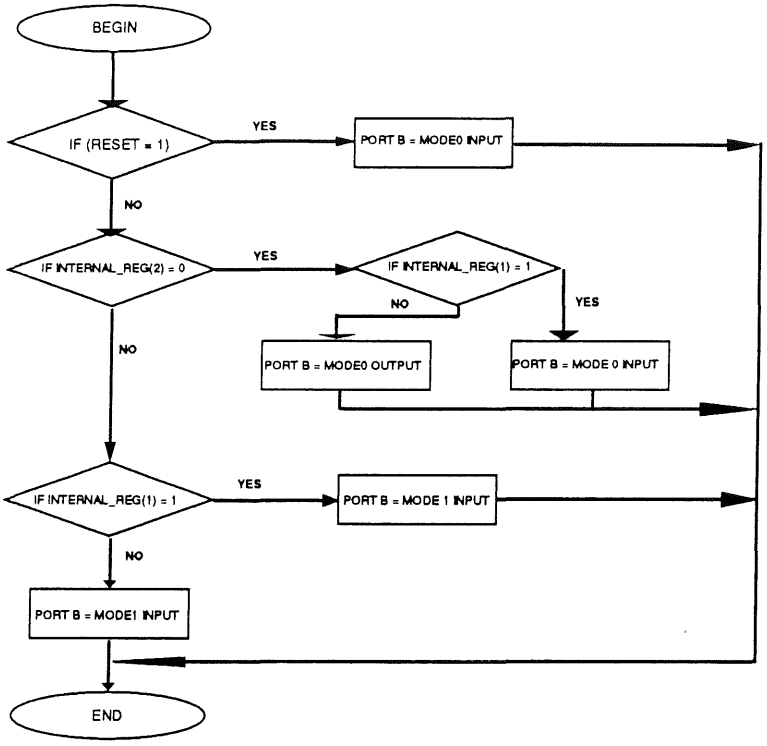


Figure 18: Flowchart For PORT B MODE SELECTOR State

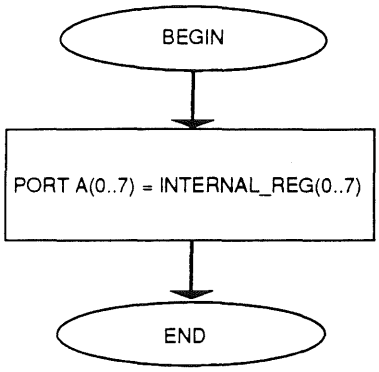


Figure 21: Flowchart For MODE 0 PORT A OUTPUT State

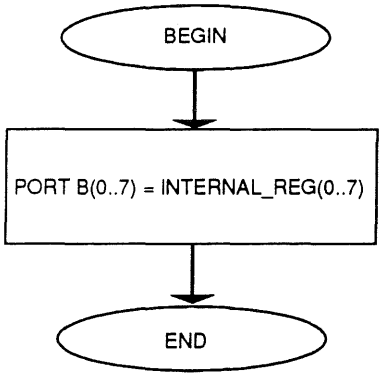


Figure 22: Flowchart For MODE 0 PORT B OUTPUT State

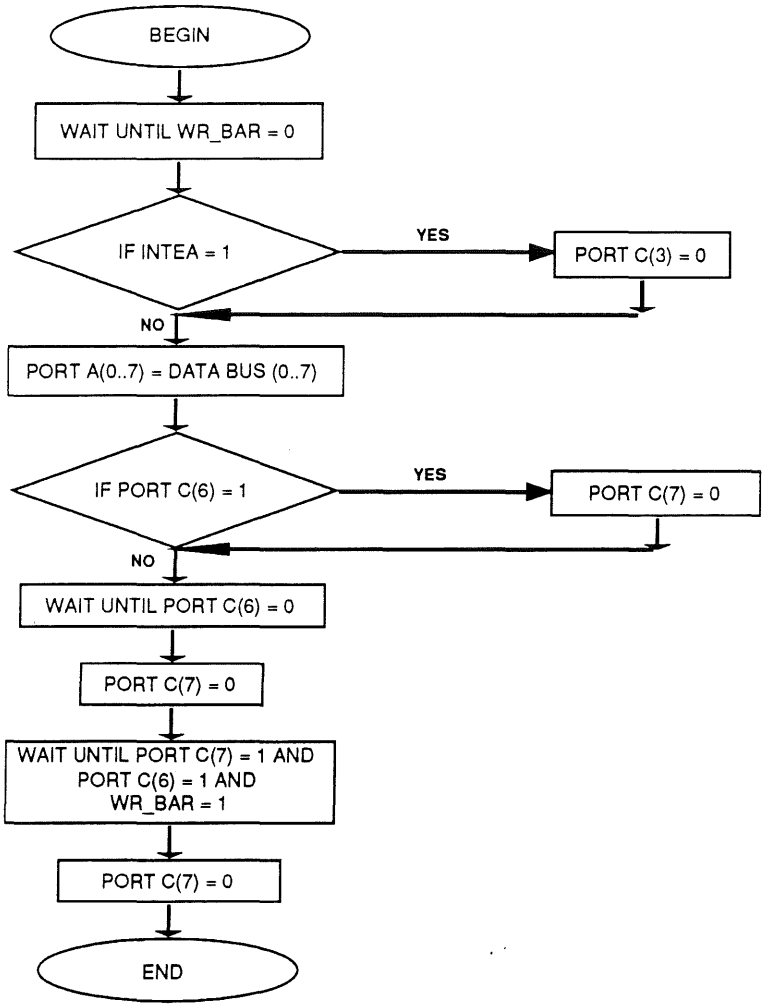


Figure 23: Flowchart For MODE 1 PORT A OUTPUT State

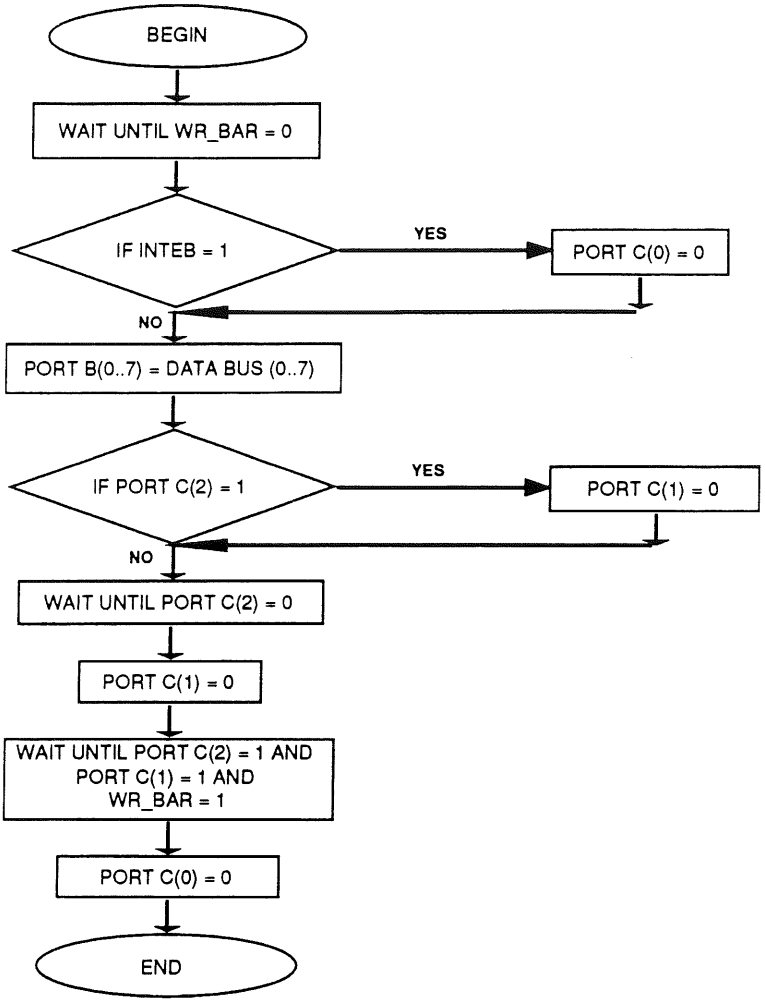


Figure 24: Flowchart For MODE 1 PORT B OUTPUT State

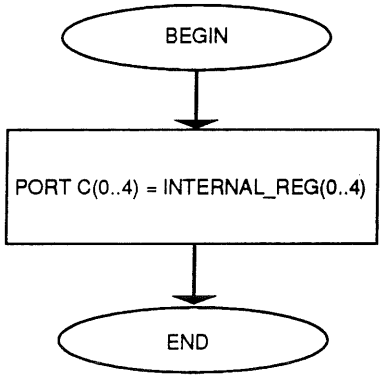


Figure 25: Flowchart For MODE 0 LOWER PORT C OUTPUT State

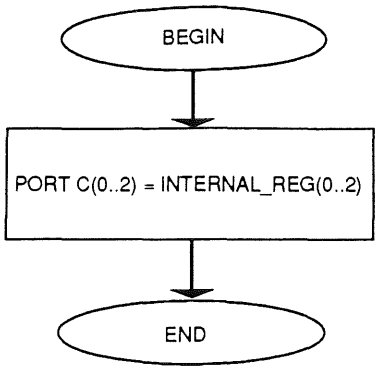


Figure 26: Flowchart For MODE 1 LOWER PORT C OUTPUT State

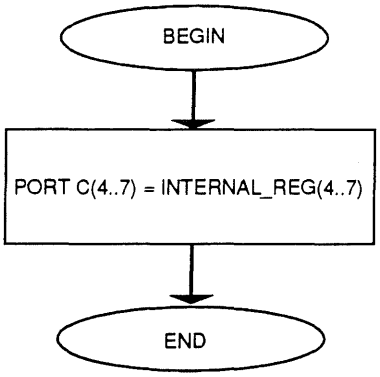


Figure 27: Flowchart For MODE 0 UPPER PORT C OUTPUT State

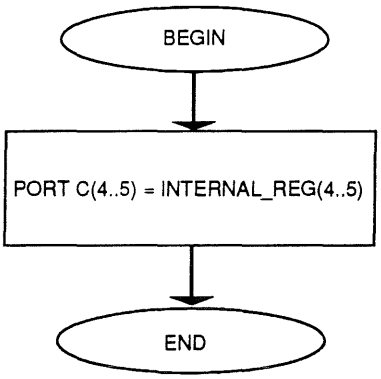


Figure 28: Flowchart For MODE 1 UPPER PORT C-45 OUTPUT State

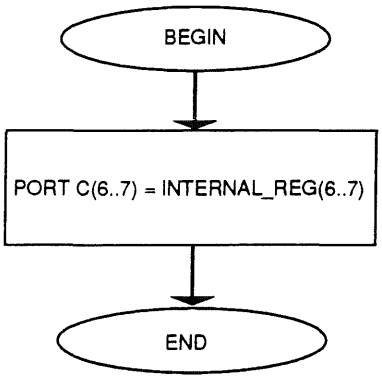


Figure 29: Flowchart For MODE 1 UPPER PORT C-67 OUTPUT State

10 Appendix II

This appendix contains test vectors and test waveforms for those test scenario describes in Section 5.

```
addev[CS_BAR]:signal {[F0 @ 0 ns]};
addev[WR_BAR]:signal {[F1 @ 0 ns], [F0 @ 100 ns], [F1 @ 600 ns],
                      [F0 @ 700 ns],
                      [F1 @ 1200 ns], [F0 @ 1300 ns],[F1 @ 1800 ns],
      [F0 @ 1900 ns], [F1 @ 2400 ns] };

addev[RD_BAR]:signal {[F1 @ 0 ns]};

addev[RESET]: signal {[F0 @ 0 ns], [F1 @ 1 ns], [F0 @ 5 ns] };

addev[D0]: signal {[ZX @ 0 ns],[F0 @ 10 ns],[ZX @ 650 ns],[F0 @ 680 ns],
                   [F1 @ 1850 ns]};
addev[D1]: signal {[ZX @ 0 ns],[F0 @ 10 ns],[ZX @ 650 ns],[ F0 @ 680 ns]};
addev[D2]: signal {[ZX @ 0 ns],[F0 @ 10 ns],[ZX @ 650 ns],[F0 @ 680 ns],
                   [F1 @ 1250 ns]};
addev[D3]: signal {[ZX @ 0 ns],[F0 @ 10 ns],[ZX @ 650 ns],[F0 @ 680 ns]};
addev[D4]: signal {[ZX @ 0 ns],[F0 @ 10 ns],[ZX @ 650 ns], [F1 @ 680 ns],
                   [F1 @ 1850 ns]};
addev[D5]: signal {[ZX @ 0 ns],[F0 @ 10 ns],[ZX @ 650 ns], [F1 @ 680 ns],
                   [F0 @ 1250 ns]};
addev[D6]: signal {[ZX @ 0 ns],[F0 @ 10 ns],[ZX @ 650 ns], [F1 @ 680 ns],
                   [F0 @ 1850 ns]};
addev[D7]: signal {[ZX @ 0 ns],[F1 @ 10 ns],[ZX @ 650 ns],[F1 @ 680 ns],
                   [F0 @ 1850 ns]};

addev[A0_MODE]: signal {[F1 @ 0 ns], [F0 @ 670 ns], [F0 @ 1280 ns],
                  [F1 @ 1890 ns]};
addev[A1_MODE]: signal {[F1 @ 0 ns], [F1 @ 670 ns],[F0 @ 1280 ns]};
```

Figure 30: Test Vector For Example 1

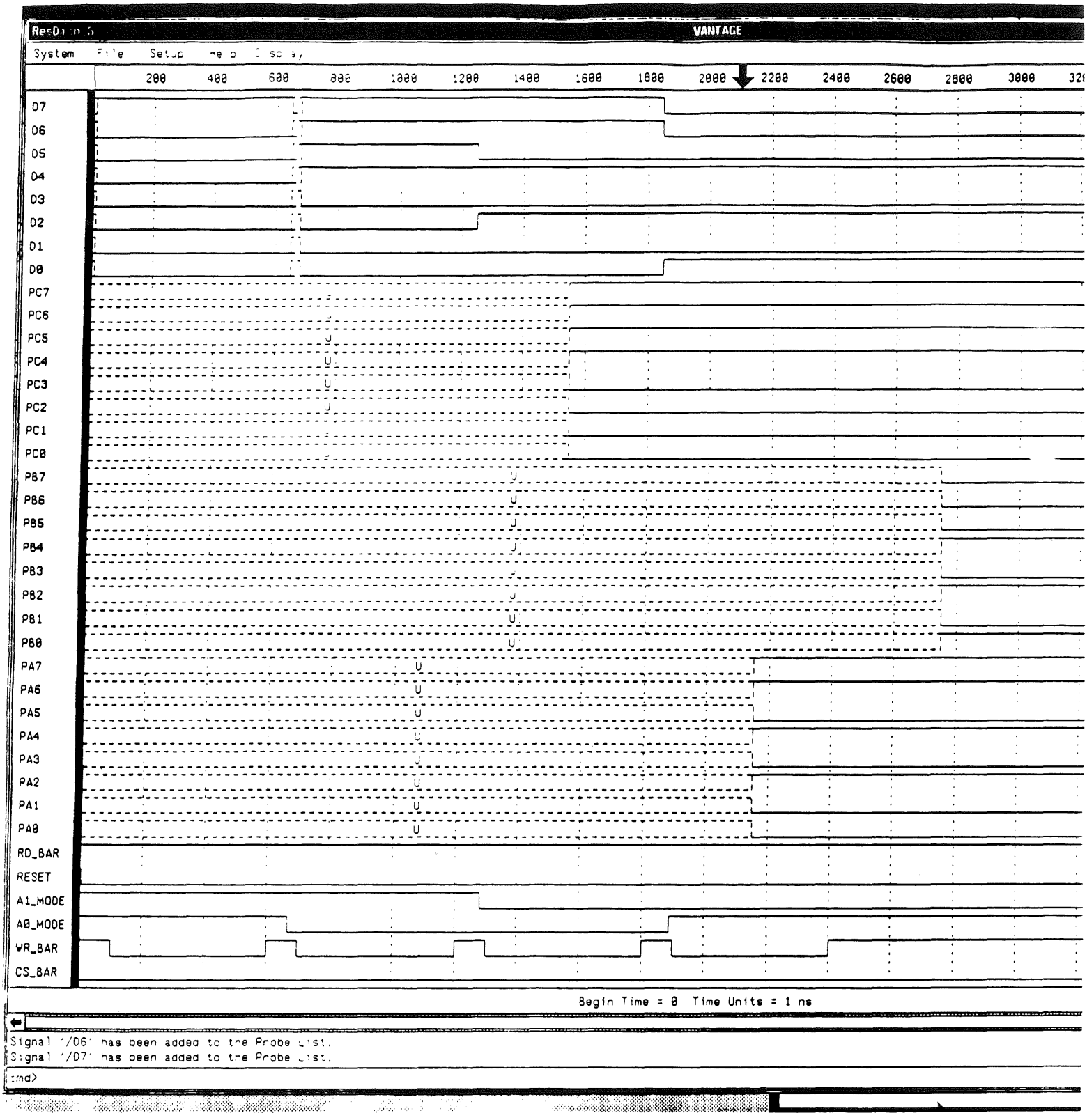


Figure 31: Simulation Result For Example 1

```
addev[CS_BAR]:signal {[F0 @ 0 ns]};
addev[WR_BAR]:signal {[F1 @ 0 ns], [F0 @ 100 ns], [F1 @ 600 ns]};
addev[RD_BAR]:signal {[F1 @ 0 ns],[F0 @ 1800 ns],
                      [F1 @ 2200 ns], [F0 @ 2700 ns],[F1 @ 3000 ns],
      [F0 @ 3300 ns], [F1 @ 3800 ns] };
addev[RESET]: signal {[F0 @ 0 ns], [F1 @ 1 ns], [F0 @ 5 ns] };

addev[D0]: signal {[ZX @ 0 ns],[F1 @ 70 ns],[ZX @ 700 ns]};
addev[D1]: signal {[ZX @ 0 ns],[F1 @ 70 ns],[ ZX @ 700 ns]};
addev[D2]: signal {[ZX @ 0 ns],[F0 @ 70 ns],[ZX @ 700 ns]};
addev[D3]: signal {[ZX @ 0 ns],[F1 @ 70 ns],[ZX @ 700 ns]};
addev[D4]: signal {[ZX @ 0 ns],[F1 @ 70 ns], [ZX @ 700 ns]};
addev[D5]: signal {[ZX @ 0 ns],[F0 @ 70 ns], [ZX @ 700 ns]};
addev[D6]: signal {[ZX @ 0 ns],[F0 @ 70 ns], [ZX @ 700 ns]};
addev[D7]: signal {[ZX @ 0 ns],[F1 @ 70 ns],[ZX @ 700 ns]};

addev[PA0]: signal {[F1 @ 10 ns]};
addev[PA1]: signal {[F1 @ 10 ns]};
addev[PA2]: signal {[F0 @ 10 ns]};
addev[PA3]: signal {[F0 @ 10 ns]};
addev[PA4]: signal {[F1 @ 10 ns]};
addev[PA5]: signal {[F0 @ 10 ns]};
addev[PA6]: signal {[F0 @ 10 ns]};
addev[PA7]: signal {[F1 @ 10 ns]};

addev[PB0]: signal {[F1 @ 10 ns]};
addev[PB1]: signal {[F1 @ 10 ns]};
addev[PB2]: signal {[F0 @ 10 ns]};
addev[PB3]: signal {[F1 @ 10 ns]};
addev[PB4]: signal {[F1 @ 10 ns]};
addev[PB5]: signal {[F0 @ 10 ns]};
addev[PB6]: signal {[F0 @ 10 ns]};
addev[PB7]: signal {[F0 @ 10 ns]};

addev[PC0]: signal {[F0 @ 10 ns]};
addev[PC1]: signal {[F0 @ 10 ns]};
addev[PC2]: signal {[F0 @ 10 ns]};
addev[PC3]: signal {[F1 @ 10 ns]};
addev[PC4]: signal {[F1 @ 10 ns]};
addev[PC5]: signal {[F1 @ 10 ns]};
addev[PC6]: signal {[F1 @ 10 ns]};
addev[PC7]: signal {[F1 @ 10 ns]};

addev[A0_MODE]: signal {[F1 @ 0 ns], [F0 @ 810 ns], [F1 @ 2300 ns],
                        [F0 @ 3100 ns]};
addev[A1_MODE]: signal {[F1 @ 0 ns], [F0 @ 810 ns], [F1 @ 3100 ns]};
```

Figure 32: Test Vector For Example 2

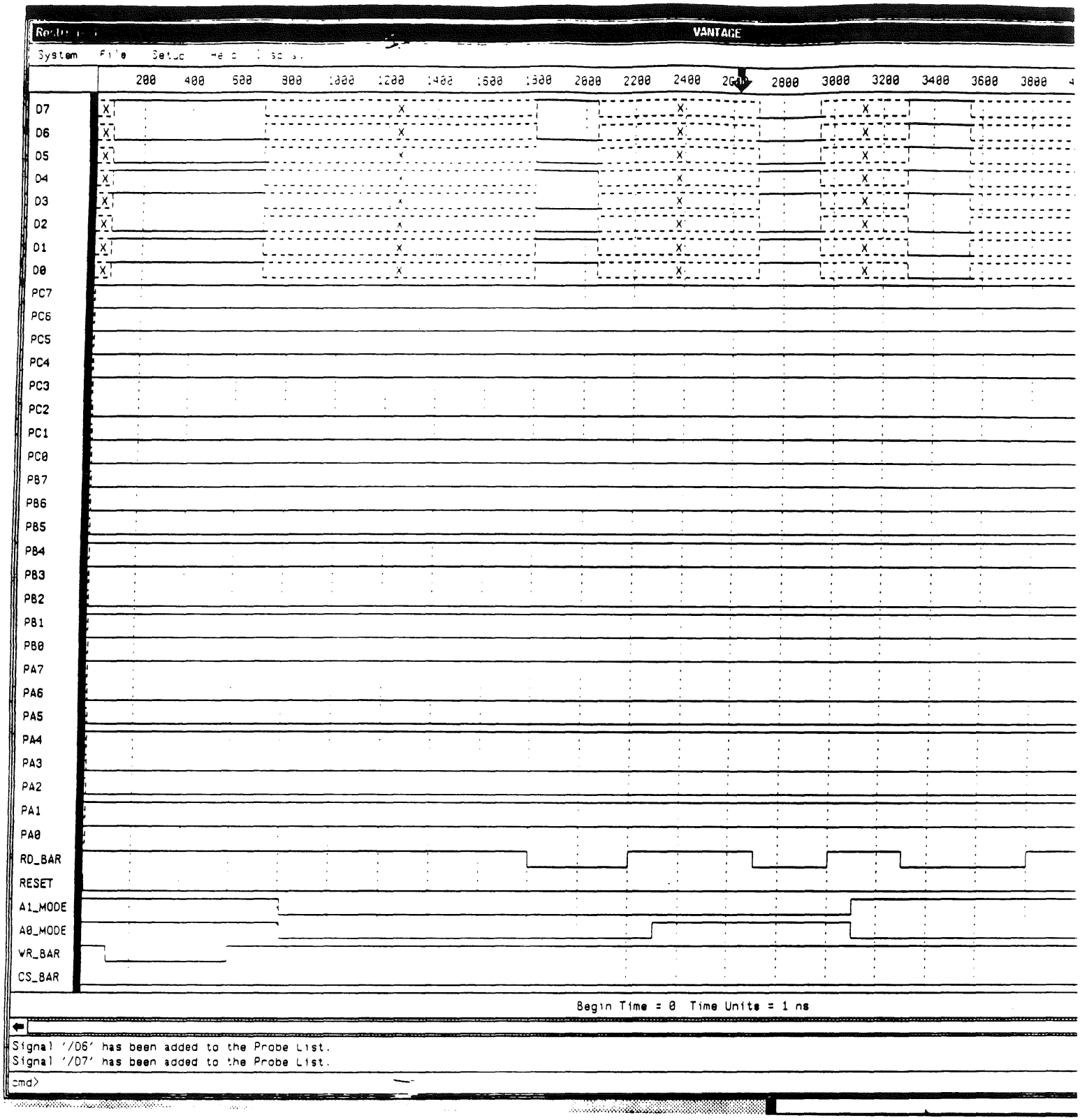


Figure 33: Simulation Result For Example 2

```
addev[PA0]:signal  {[F1 @ 0 ns]};
addev[PA1]:signal  {[F1 @ 0 ns]};
addev[PA2]:signal  {[F1 @ 0 ns]};
addev[PA3]:signal  {[F1 @ 0 ns]};
addev[PA4]:signal  {[F0 @ 0 ns]};
addev[PA5]:signal  {[F0 @ 0 ns]};
addev[PA6]:signal  {[F0 @ 0 ns]};
addev[PA7]:signal  {[F0 @ 0 ns]};


addev[D0]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F1 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns]};
addev[D1]:signal  {[ZX @ 0 ns],[F1 @ 100 ns],[F1 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns]};
addev[D2]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F1 @ 1000 ns],[F1 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns]};
addev[D3]:signal  {[ZX @ 0 ns],[F1 @ 100 ns],[F0 @ 1000 ns],[F1 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns]};
addev[D4]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns]};
addev[D5]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns]};
addev[D6]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns]};
addev[D7]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns]};

addev[CS_BAR]:signal {[F0 @ 0 ns]};
addev[WR_BAR]:signal {[F1 @ 0 ns], [F0 @ 200 ns], [F1 @ 700 ns],
                      [F0 @ 1200 ns], [F1 @ 2000 ns],[F0 @ 2300 ns],
                      [F1 @ 3000 ns],[F0 @ 3300 ns],[F1 @ 4000 ns]};
addev[RESET]:signal {[F0 @ 0 ns], [F1 @ 7 ns], [F0 @ 9 ns]};
addev[A0_MODE]:signal {[F1 @ 0 ns], [ F0 @ 4200 ns]};
addev[A1_MODE]:signal {[F1 @ 0 ns], [ F0 @ 4200 ns]};
addev[PC4]:signal {[F1 @ 0 ns],[F0 @ 5000 ns],[F1 @ 6000 ns]};
addev[RD_BAR]:signal {[F1 @ 0 ns],[F0 @ 7000 ns],[F1 @ 7900 ns]};
```

Figure 34: Test Vector For Example 3

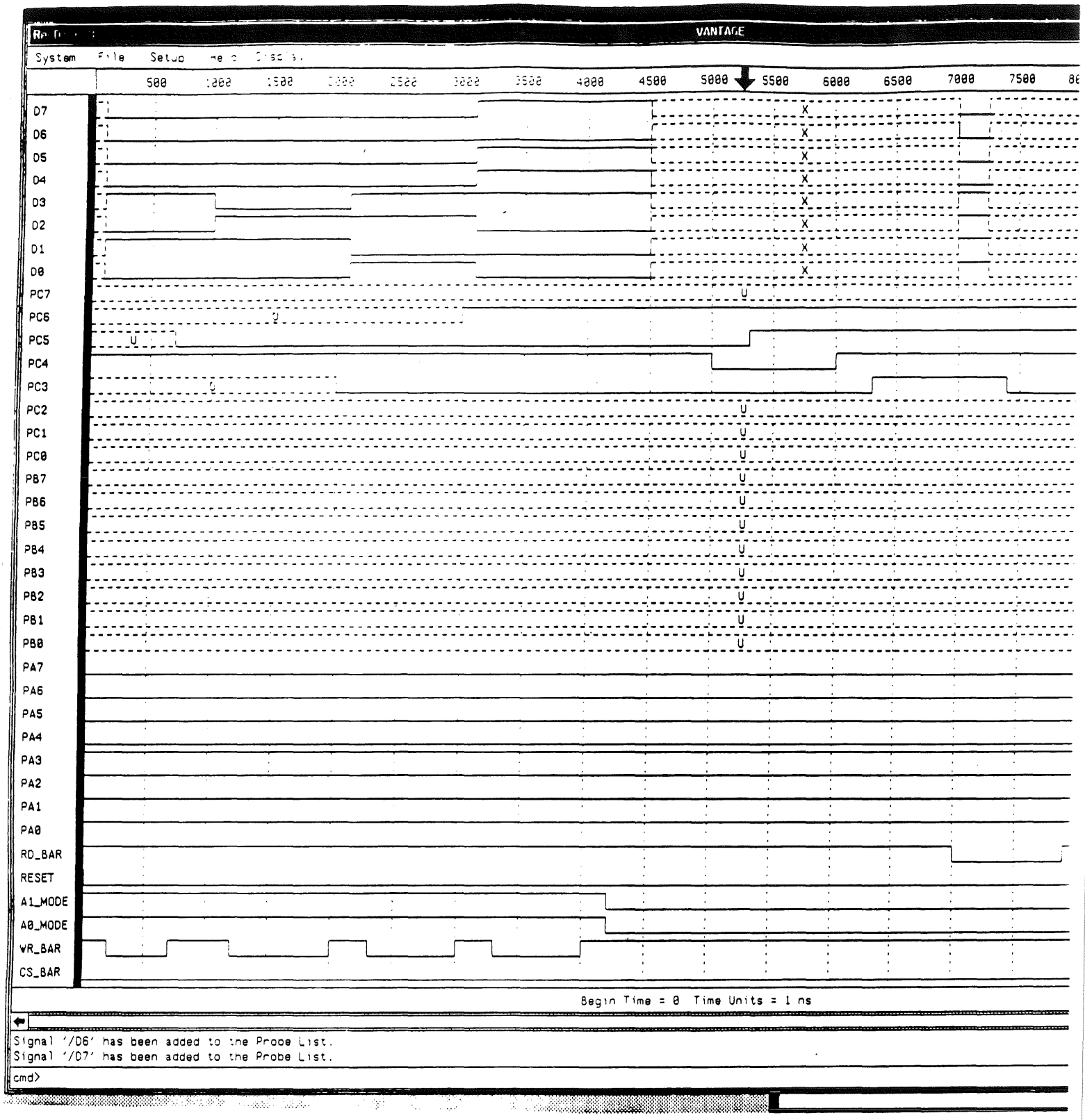


Figure 35: Simulation Result For Example 3

```
addev[PA0]:signal  {[ZX @ 0 ns]};
addev[PA1]:signal  {[ZX @ 0 ns]};
addev[PA2]:signal  {[ZX @ 0 ns]};
addev[PA3]:signal  {[ZX @ 0 ns]};
addev[PA4]:signal  {[ZX @ 0 ns]};
addev[PA5]:signal  {[ZX @ 0 ns]};
addev[PA6]:signal  {[ZX @ 0 ns]};
addev[PA7]:signal  {[ZX @ 0 ns]};


addev[D0]:signal  {[ZX @ 0 ns],[F1 @ 100 ns],[F1 @ 1000 ns],[F1 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns],[F0 @ 4700 ns]};
addev[D1]:signal  {[ZX @ 0 ns],[F1 @ 100 ns],[F1 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns],[F0 @ 4700 ns]};
addev[D2]:signal  {[ZX @ 0 ns],[F1 @ 100 ns],[F1 @ 1000 ns],[F1 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};
addev[D3]:signal  {[ZX @ 0 ns],[F1 @ 100 ns],[F0 @ 1000 ns],[F1 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};
addev[D4]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};
addev[D5]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};
addev[D6]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};
addev[D7]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};

addev[CS_BAR]:signal {[F0 @ 0 ns]};
addev[WR_BAR]:signal {[F1 @ 0 ns], [F0 @ 200 ns], [F1 @ 700 ns],
                      [F0 @ 1200 ns], [F1 @ 2000 ns],[F0 @ 2300 ns],
                      [F1 @ 3000 ns],[F0 @ 3300 ns],[F1 @ 4000 ns],
                      [F0 @ 5000 ns],[F1 @ 5800 ns]};
addev[RESET]:signal {[F0 @ 0 ns], [F1 @ 7 ns], [F0 @ 9 ns]};
addev[A0_MODE]:signal {[F1 @ 0 ns], [ F0 @ 4200 ns]};
addev[A1_MODE]:signal {[F1 @ 0 ns], [ F0 @ 4200 ns]};
addev[PC6]:signal {[F1 @ 0 ns],[F0 @ 9200 ns],[F1 @ 9800 ns]};
addev[RD_BAR]:signal {[F1 @ 0 ns]};
```

Figure 36: Test Vector For Example 4

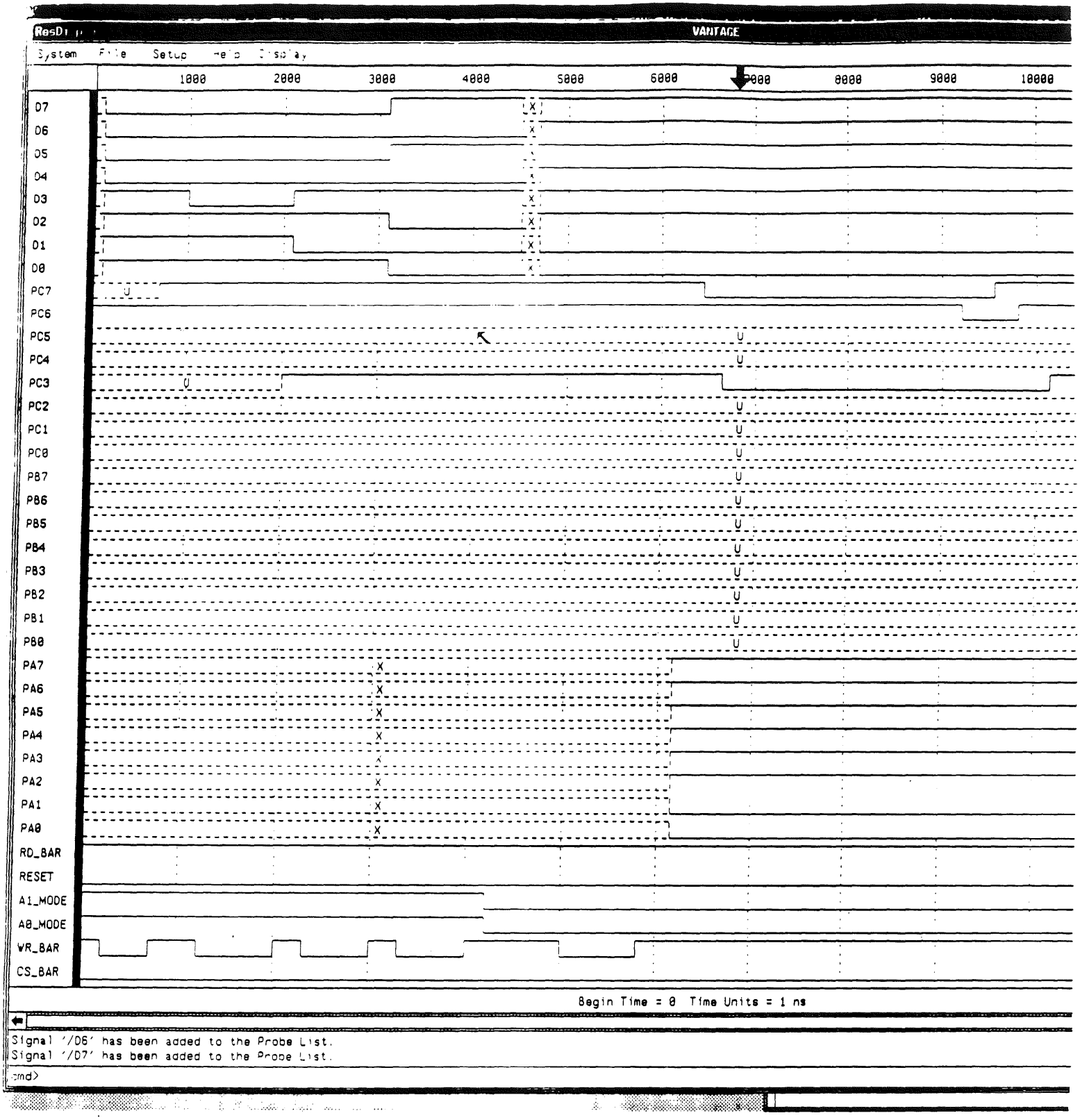


Figure 37: Simulation Result For Example 4

```
addev[PB0]:signal  {[F1 @ 0 ns]};
addev[PB1]:signal  {[F1 @ 0 ns]};
addev[PB2]:signal  {[F1 @ 0 ns]};
addev[PB3]:signal  {[F1 @ 0 ns]};
addev[PB4]:signal  {[F0 @ 0 ns]};
addev[PB5]:signal  {[F0 @ 0 ns]};
addev[PB6]:signal  {[F0 @ 0 ns]};
addev[PB7]:signal  {[F0 @ 0 ns]};


addev[D0]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F1 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns]};
addev[D1]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F1 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns]};
addev[D2]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F1 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns]};
addev[D3]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns]};
addev[D4]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns]};
addev[D5]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns]};
addev[D6]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns]};
addev[D7]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns]};

addev[CS_BAR]:signal {[F0 @ 0 ns]};
addev[WR_BAR]:signal {[F1 @ 0 ns], [F0 @ 200 ns], [F1 @ 700 ns],
                      [F0 @ 1200 ns], [F1 @ 2000 ns],[F0 @ 2300 ns],
                      [F1 @ 3000 ns],[F0 @ 3300 ns],[F1 @ 4000 ns]};
addev[RESET]:signal {[F0 @ 0 ns], [F1 @ 7 ns], [F0 @ 9 ns]};
addev[A0_MODE]:signal {[F1 @ 0 ns], [ F1 @ 4200 ns]};
addev[A1_MODE]:signal {[F1 @ 0 ns], [ F0 @ 4200 ns]};
addev[PC2]:signal {[F1 @ 0 ns],[F0 @ 5000 ns],[F1 @ 6000 ns]};
addev[RD_BAR]:signal {[F1 @ 0 ns],[F0 @ 7000 ns],[F1 @ 7900 ns]};
```

Figure 38: Test Vector For Example 5

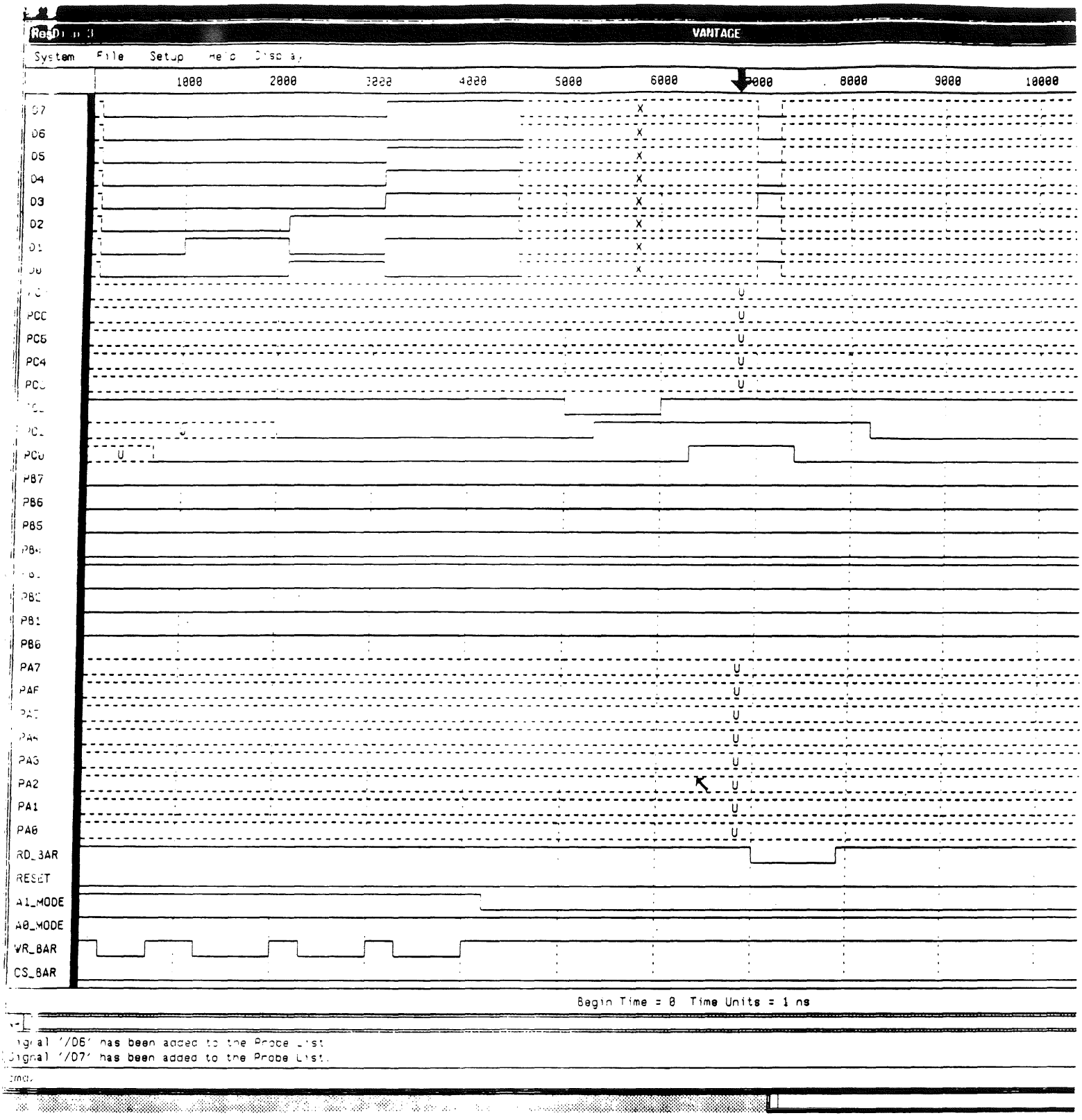


Figure 39: Simulation Result For Example 5

```
addev[PA0]:signal  {[ZX @ 0 ns]};
addev[PA1]:signal  {[ZX @ 0 ns]};
addev[PA2]:signal  {[ZX @ 0 ns]};
addev[PA3]:signal  {[ZX @ 0 ns]};
addev[PA4]:signal  {[ZX @ 0 ns]};
addev[PA5]:signal  {[ZX @ 0 ns]};
addev[PA6]:signal  {[ZX @ 0 ns]};
addev[PA7]:signal  {[ZX @ 0 ns]};


addev[D0]:signal  {[ZX @ 0 ns],[F1 @ 100 ns],[F1 @ 1000 ns],[F1 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns],[F0 @ 4700 ns]};
addev[D1]:signal  {[ZX @ 0 ns],[F1 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns],[F0 @ 4700 ns]};
addev[D2]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F1 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};
addev[D3]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};
addev[D4]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};
addev[D5]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};
addev[D6]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};
addev[D7]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F1 @ 3100 ns],[ZX @ 4500 ns],[F1 @ 4700 ns]};

addev[CS_BAR]:signal {[F0 @ 0 ns]};
addev[WR_BAR]:signal {[F1 @ 0 ns], [F0 @ 200 ns], [F1 @ 700 ns],
                      [F0 @ 1200 ns], [F1 @ 2000 ns],[F0 @ 2300 ns],
                      [F1 @ 3000 ns],[F0 @ 3300 ns],[F1 @ 4000 ns],
                      [F0 @ 5000 ns],[F1 @ 5800 ns]};
addev[RESET]:signal {[F0 @ 0 ns], [F1 @ 7 ns], [F0 @ 9 ns]};
addev[A0_MODE]:signal {[F1 @ 0 ns], [ F1 @ 4200 ns]};
addev[A1_MODE]:signal {[F1 @ 0 ns], [ F0 @ 4200 ns]};
addev[PC2]:signal {[F1 @ 0 ns],[F0 @ 9200 ns],[F1 @ 9800 ns]};
addev[RD_BAR]:signal {[F1 @ 0 ns]};
```

Figure 40: Test Vector For Example 6

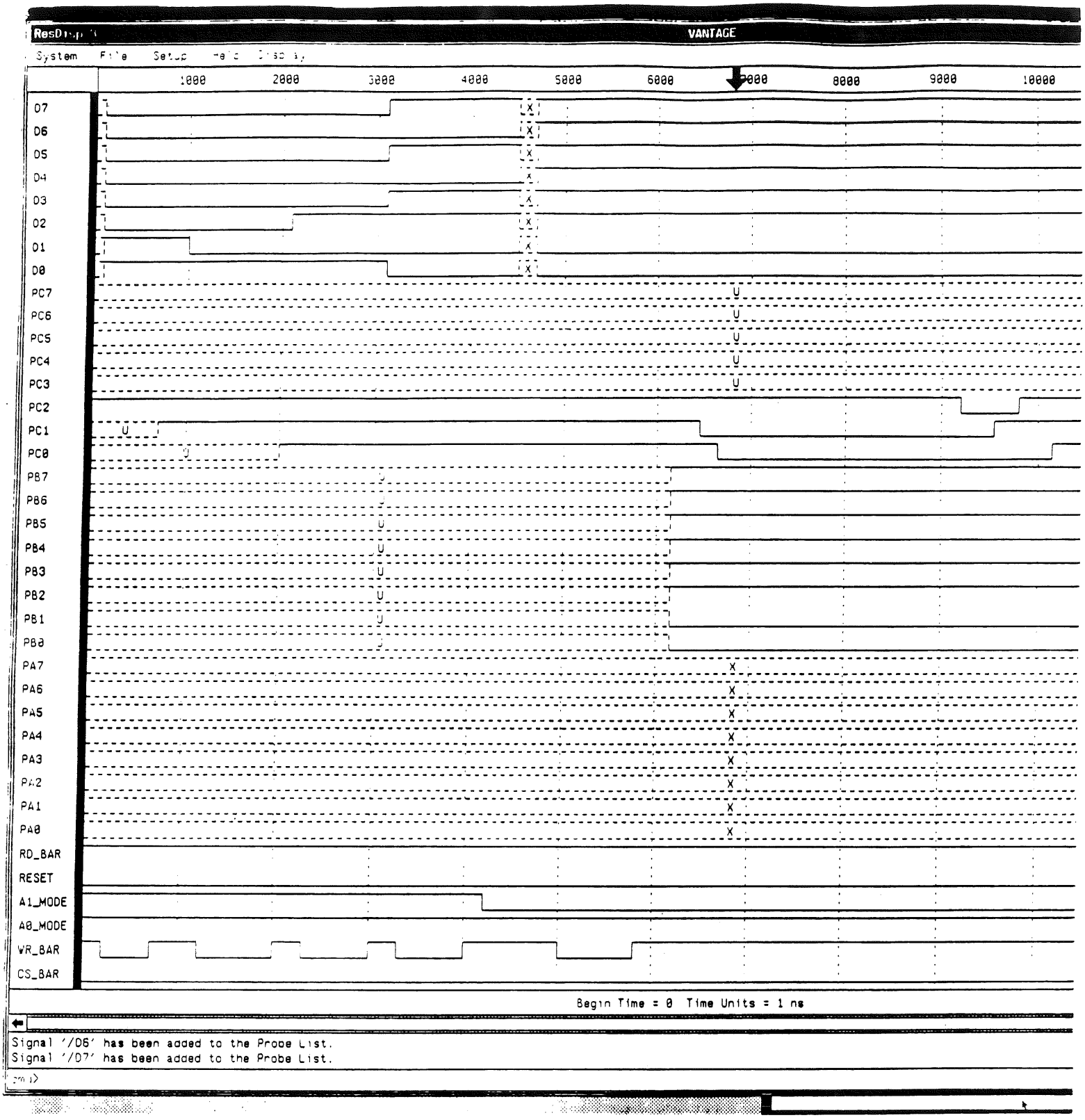


Figure 41: Simulation Result For Example 6

```
addev[PA0]:signal  {[ZX @ 0 ns],[F1 @ 15000 ns]};
addev[PA1]:signal  {[ZX @ 0 ns],[F1 @ 15000 ns]};
addev[PA2]:signal  {[ZX @ 0 ns],[F1 @ 15000 ns]};
addev[PA3]:signal  {[ZX @ 0 ns],[F1 @ 15000 ns]};
addev[PA4]:signal  {[ZX @ 0 ns],[F1 @ 15000 ns]};
addev[PA5]:signal  {[ZX @ 0 ns],[F1 @ 15000 ns]};
addev[PA6]:signal  {[ZX @ 0 ns],[F1 @ 15000 ns]};
addev[PA7]:signal  {[ZX @ 0 ns],[F1 @ 15000 ns]};

addev[D0]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F1 @ 1000 ns],[F1 @ 2100 ns],
[F1 @ 3100 ns],[F1 @ 4600 ns],[F1 @ 5800 ns],[F1 @ 7500 ns],
                [ZX @ 15000 ns]};
addev[D1]:signal  {[ZX @ 0 ns],[F1 @ 100 ns],[F1 @ 1000 ns],[F1 @ 2100 ns],
[F0 @ 3100 ns],[F0 @ 4600 ns],[F1 @ 5800 ns],[F0 @ 7500 ns],
                [ZX @ 15000 ns]};
addev[D2]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F1 @ 1000 ns],[F1 @ 2100 ns],
[F0 @ 3100 ns],[F1 @ 4600 ns],[F1 @ 5800 ns],[F0 @ 7500 ns],
                [ZX @ 15000 ns]};
addev[D3]:signal  {[ZX @ 0 ns],[F1 @ 100 ns],[F0 @ 1000 ns],[F1 @ 2100 ns],
[F1 @ 3100 ns],[F1 @ 4600 ns],[F1 @ 5800 ns],[F0 @ 7500 ns],
                [ZX @ 15000 ns]};
addev[D4]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[F0 @ 4600 ns],[F1 @ 5800 ns],[F0 @ 7500 ns],
                [ZX @ 15000 ns]};
addev[D5]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[F0 @ 4600 ns],[F1 @ 5800 ns],[F1 @ 7500 ns],
                [ZX @ 15000 ns]};
addev[D6]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[F0 @ 4600 ns],[F1 @ 5800 ns],[F1 @ 7500 ns],
                [ZX @ 15000 ns]};
addev[D7]:signal  {[ZX @ 0 ns],[F0 @ 100 ns],[F0 @ 1000 ns],[F0 @ 2100 ns],
[F0 @ 3100 ns],[F0 @ 4600 ns],[F1 @ 5800 ns],[F1 @ 7500 ns],
                [ZX @ 15000 ns]};

addev[CS_BAR]:signal {[F0 @ 0 ns]};
addev[WR_BAR]:signal {[F1 @ 0 ns], [F0 @ 200 ns], [F1 @ 700 ns],
                      [F0 @ 1200 ns], [F1 @ 2000 ns],[F0 @ 2300 ns],
                      [F1 @ 3000 ns],[F0 @ 3300 ns],[F1 @ 4000 ns],
                      [F0 @ 4800 ns],[F1 @ 5500 ns],
                      [F0 @ 6000 ns],[F1 @ 7000 ns],
                      [F0 @ 8000 ns],[F1 @ 8800 ns]};
addev[RESET]:signal {[F0 @ 0 ns], [F1 @ 7 ns], [F0 @ 9 ns]};
addev[A0_MODE]:signal {[F1 @ 0 ns], [ F0 @ 7500 ns]};
addev[A1_MODE]:signal {[F1 @ 0 ns], [ F0 @ 7500 ns]};
addev[PC4]:signal {[F1 @ 0 ns],[F0 @ 17000 ns],[F1 @ 18000 ns]};
addev[PC6]:signal {[F1 @ 0 ns],[F0 @ 12000 ns],[F1 @ 13000 ns]};
addev[RD_BAR]:signal {[F1 @ 0 ns],[F0 @ 20000 ns],[F1 @ 21000 ns]};
```

Figure 42: Test Vector For Example 7

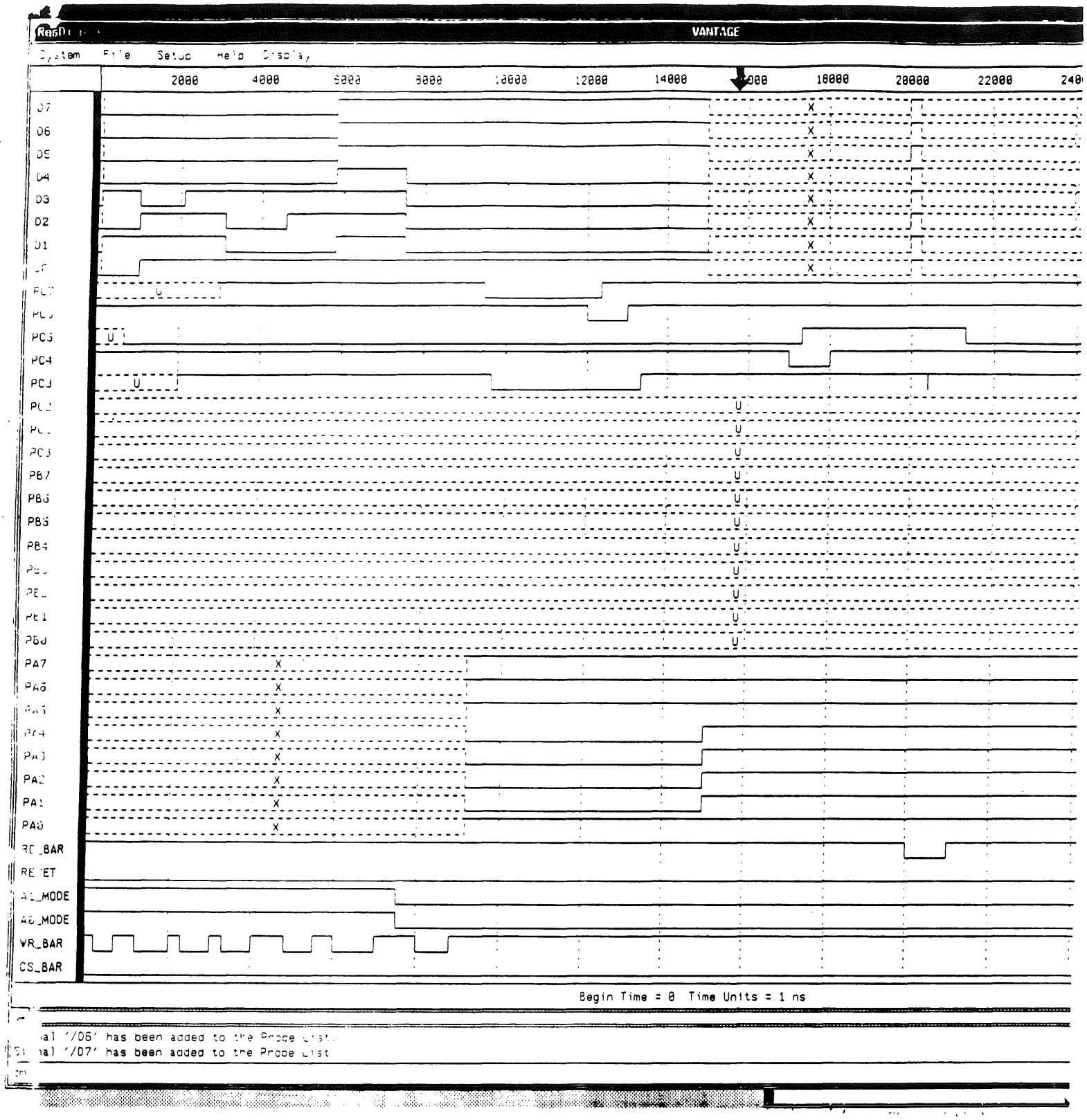


Figure 43: Simulation Result For Example 7

11 Appendix III

This appendix contains listing of the actual VHDL behavioral model code for the Intel 8255A. This code is implemented such that it is correctly simulated under the the Vantage Analysis Systems, version 1.08 [Vant89].

```
--
--      8255A_behavior.vhdl
--
--      Description:
--              This file contains VHDL behavioral description of the
--              Intel 8255A.  The code has been successfully compiled
--              and simulated using Vantage Analysis Systems (ver 1.108).
--
--      DATE:
--              6/12/90
--
--      COMMENT:
--              1) All timing information in this file is obtained from
--                 the TTL data book (issue date November 1987).
--              2) To compile this file, first compile the file "common.vhdl"
--                 which contains user-defined data structures which are used
--                 in this file.
--
--      BY:
--              Viraphol Chaiyakul
--
```

```
-- use pre-defined Vantage's standard logic type

use work.std_logic.all;

-- Also use user-defined data structure in the "common" package.  This
-- package is defined in the file "common.vhld"

use work.common.all;

-- Entity declaration of the Intel 8255a

entity i8255a_e is

-- The following timing information is obtained from the TTL data book

  generic (

  -- WR enable to output delay
  tWB : time := 350 ns;

  -- Data valid from read delay
  tRD : time := 250 ns;

  -- Data float after read delay
  tDF : time := 10 ns;

  -- WR = 0 to INTR = 0 delay
  tWIT : time := 850 ns;
```



```
  -- RD = 0 to INTR = 0 delay
  tRIT : time := 400 ns;

  -- RD = 1 to IBF= 0 delay
  tRIB : time := 300 ns;

  -- STB = 1 to INTR = 1 delay
  tSIT : time := 300 ns;

  -- STB = 0 to IBF = 1 delay
  tSIB : time := 300 ns;

  -- ACK = 0 to OBF = 1 delay
  tAOB : time := 350 ns;

  -- ACK = 1 to INTR = 1 delay
  tAIT : time := 350 ns;

  -- WR = 1 to OBF = 0 delay
  tWOB : time := 650 ns;

  -- Address stable before READ
  tAR : time := 0 ns;

  -- Address stable after READ
  tRA : time := 0 ns;

  -- READ pulse width
  tRR : time := 300 ns;

  -- Time between READs and/or WRITEs
  tRV : time := 850 ns;

  -- Address stable before WRITE
  tAW : time := 0 ns;

  -- Address stable after WRITE
  tWA : time := 20 ns;

  -- WRITE pulse width
  tWW : time := 400 ns;

  -- Data valid to WRITE
  tDW : time := 100 ns;

  -- Data valid after WRITE
  tWD : time := 30 ns;

  -- Peripheral data before RD
  tIR : time := 0 ns;

  -- Peripheral data after RD
  tHR : time := 0 ns;

  -- ACK pulse width
  tAK : time := 300 ns;
```

```
  -- STB pulse width
  tST : time := 500 ns;

  -- Peripheral data before T.E. of STB
  tPS : time := 0 ns;

  -- Peripheral data after T.E. of STB
  tPH : time := 180 ns;

  -- ACK = 0 to output
  tAD : time := 300 ns;

  -- ACK = 1 to output float
  tKD : time := 20 ns);

-- Ports declaration.  Note that the Vantage's pre-defined logic is used
-- as the basic-data type.  This pre-defined data type is defined in the
-- Vantage's STD_LOGIC package together with its bus resolution functions
-- (refer to the "STD_LOGIC VHDL Package" manual, dated April 3,1990, by
-- Vantage Analysis System, Inc.)

  port (

    -- control signals
    signal CS_BAR, WR_BAR, A0_MODE, A1_MODE, RESET , RD_BAR: in t_wlogic;

    -- Output Port A interface
    signal PA0,PA1,PA2,PA3,PA4,PA5,PA6,PA7 : inout t_wlogic bus;

    -- Output Port B interface
    signal PB0,PB1,PB2,PB3,PB4,PB5,PB6,PB7 : inout t_wlogic bus;

    -- Output Port C interface
    signal PC0,PC1,PC2,PC3,PC4,PC5,PC6,PC7 : inout t_wlogic bus;

    -- Input Data bus interface
    signal D0,D1,D2,D3,D4,D5,D6,D7 : inout t_wlogic bus);

end i8255a_e;


-- Architecture declaration

architecture i8255a_a of i8255a_e is

-- Internal signals declaration

  -- pres_mode_a tells the current mode for the port A operations
  signal pres_modea : mode_port_a;

  -- pres_mode_b tells the current mode for the port B operations
  signal pres_modeb : mode_port_b;

  -- pres_mode_upc tells the current mode for the upper 4 bits of port C
  -- operations.
```



```
  signal pres_mode_upc : mode_port_up_c;

  -- pres_mode_lowc tells the current mode for the lower 4 bits of port C
  -- operations.
  signal pres_mode_lowc : mode_port_low_c;

  -- enable signals for write operations for all 3 ports (enable high)
  signal out_en_a,out_en_b,out_en_c : t_wlogic;

  -- read request signal for read operations for all 3 ports (enable high)
  signal req_read_a,req_read_b,req_read_c : t_wlogic;

  -- internal register for transfer between input and output operations.
  -- These signal should have been implemented as bus.  The reason it is
  -- declared as register because we want to retain the value on these signals
  -- even though their drivers have been shut off.
  signal inb0, inb1, inb2, inb3, inb4, inb5, inb6, inb7:  t_wlogic register;

  -- enable controling the changing of current mode status (enable high)
  signal cntl_en: t_wlogic;

  -- read acknowledgement (active high)
  signal ack_read : t_wlogic bus;

  -- control for bit set and reset operation (active high)
  signal bit_set, bit_reset: t_wlogic;

  -- internal registers holding current value on the bus interface for each
  -- port
  signal C0,C1,C2,C3,C4,C5,C6,C7 : t_wlogic register;
  signal A0,A1,A2,A3,A4,A5,A6,A7 : t_wlogic register;
  signal B0,B1,B2,B3,B4,B5,B6,B7 : t_wlogic register;

  -- interrupt signal for hand-shaking modes (check TTL data book)
  signal  INTEA : t_wlogic register;
  signal  INTEB : t_wlogic register;
  signal  INTE1 : t_wlogic register;
  signal  INTE2 : t_wlogic register;

begin

  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- READ CONTROL PROCESS
  --
  -- This process controls the read operations.  When it senses the activation
  -- of the read process (RD_BAR active low), it will send read request signal
  -- to the port specified by the A0_MODE and A1_MODE control lines. Upon the
  -- completion of the read operation from the specified port, it transfers
  -- the data from internal transfer register to the CPU data bus.
  -- The process also can be activated by the chip RESET signal, upon which,
  -- it will float the CPU data bus interface lines and reset all read request
  -- signals.
  process
  begin
    wait on RD_BAR,RESET;
```

```
    -- process will perform operation only if the chip select line is active
    if (CS_BAR = '0') then

      -- if reset then float the data bus and reset all read request lines.
      if (RESET = '1') then
        D0 <= ZX;
        D1 <= ZX;
        D2 <= ZX;
        D3 <= ZX;
        D4 <= ZX;
        D5 <= ZX;
        D6 <= ZX;
        D7 <= ZX;
        req_read_a <= F0;
        req_read_b <= F0;
        req_read_c <= F0;

      -- if not reseting, then if the READ operation is active then sends
      -- required signals to the specified port.
      elsif (RD_BAR'EVENT) and (RD_BAR = '0') then

        -- Port A read request
        if (A1_MODE = '0') and (A0_MODE = '0') then
          req_read_a <= F1, F0 after tRD;

        -- Port B read request
        elsif (A1_MODE = '0') and (A0_MODE = '1') then
          req_read_b <= F1, F0 after tRD;

        -- Port C read request
        elsif (A1_MODE = '1') and (A0_MODE = '0') then
          req_read_c <= F1, F0 after tRD;
        end if;

        -- wait until the ports control finished the read operations, then
        -- placed signals from the internal transfer register to the CPU
        -- data bus interface.
        wait until (ack_read = '1') or (RESET = '1');
        if (RESET = '0') then
          D0 <= inb0 , ZX after tRD;
          D1 <= inb1, ZX after tRD;
          D2 <= inb2 , ZX after tRD;
          D3 <= inb3 , ZX after tRD;
          D4 <= inb4 , ZX after tRD;
          D5 <= inb5 , ZX after tRD;
          D6 <= inb6 , ZX after tRD;
          D7 <= inb7 , ZX after tRD;
        end if;
      end if;
    end if;
  end process;


  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- WRITE CONTROL PROCESS
  --
```



```
  -- This process controls the write operations.  When it senses the activation
  -- of the write process (WR_BAR active low), it will place the signals value
  -- from the CPU data bus interface onto the internal transfer register.
  -- Then it sends signal to the port specified by the A0_MODE and A1_MODE
  -- control lines to perform the output operations.  In the case of bit set
  -- or reset operation, it will activate the control line for that operation.
  -- The mode configuration also can be altered by the write process by invoking
  -- a different combination of control lines.  In this case, the process
  -- simply enable the mode control line so the mode selector process can
  -- alter the current configuration of the chip.
  process
  begin
    wait on WR_BAR, RESET;

    -- Process will only function if the chip select is enabled
    if (CS_BAR = '0') then

      -- If it is a reset operation then disable all internal transfer register
      -- drivers and all control lines
      if (RESET = '1') then
        inb0 <= null;
        inb1 <= null;
        inb2 <= null;
        inb3 <= null;
        inb4 <= null;
        inb5 <= null;
        inb6 <= null;
        inb7 <= null;
        out_en_a <= F0;
        out_en_b <= F0;
        out_en_c <= F0;
        bit_set <= F0;
        bit_reset <= F1, F0 after 2 ns;

      -- If it is a write operation then transfer the value on the data bus
      -- onto internal register and invoke necessary control lines.
      elsif (WR_BAR'EVENT) and (WR_BAR = '1') then

        -- transfer data from the CPU data bus interface to the internal
        -- register
        inb0 <= D0, null after tWD;
        inb1 <= D1, null after tWD;
        inb2 <= D2, null after tWD;
        inb3 <= D3, null after tWD;
        inb4 <= D4, null after tWD;
        inb5 <= D5, null after tWD;
        inb6 <= D6, null after tWD;
        inb7 <= D7, null after tWD;

        -- If bit set/reset operation then enable the bit_set line
        if (A1_MODE = '1') and (A0_MODE = '1') and (D7 = '0') then
          bit_set <= F1, F0 after tDF;

        -- enable control lines for specified output port
        elsif (A1_MODE = '0') and (A0_MODE = '0') then
          out_en_a <= F1, F0 after tDF;
```

```
      elsif (A1_MODE = '0') and (A0_MODE = '1') then
        out_en_b <= F1, F0 after tDF;
      elsif (A1_MODE = '1') and (A0_MODE = '0') then
        out_en_c <= F1, F0 after tDF;

      -- If it is the chip's configuration alteration process then enable
      -- the mode selector control line
      elsif (A1_MODE = '1') and (A0_MODE = '1') then
        cntl_en <= F1, F0 after tDF;
      end if;
    end if;
  end if;
end process;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- BITRESET PROCESS
--
-- This process control the bit reset operation.  When active, it will reset
-- all the interrupt control registers
BITRESET: block (bit_reset = '1')
begin
process
  begin

   -- if active then reset all interrupt control registers.  Note that the
   -- delay before turning off the driver is set to be a constant (5 ns.)
      because there is no such delay specified in the TTL data book.
  if guard then
    INTEA <= F0, null after 5 ns;
    INTEB <= F0, null after 5 ns;
    INTE1 <= F0, null after 5 ns;
    INTE2 <= F0, null after 5 ns;
  else
    INTEA <= null;
    INTEB <= null;
    INTE1 <= null;
    INTE2 <= null;
  end if;
  wait on guard;
  end process;
end  block BITRESET;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- BITSET PROCESS
--
-- This process control the bit set operation.  When active, it will set
-- all interrupt control registers and all the output of port C.
BITSET: block (bit_set = '1' or RESET = '1')
begin
process
  begin
  if guard then
```



```
      -- If it is a reset operation then disable all drivers.
      if (RESET = '1') then
        INTEA <= null;
        INTEB <= null;
        INTE1 <= null;
        INTE2 <= null;
        C0 <= null;
        C1 <= null;
        C2 <= null;
        C3 <= null;
        C4 <= null;
        C5 <= null;
        C6 <= null;
        C7 <= null;

      -- Else it is a bit set operation.  Bit is set according to the
      -- control signals which has been placed on the internal transfer
      -- register in the WRITE process.  Set bits specified by the control
      -- register.  Note that there is a delay before disable the driver
      -- which is set to be a constant (2 ns) becuase no such delay parameters
      -- are given in the TTL data book.
      else
        if (inb3 = '0') and (inb2 = '0') and (inb1 = '0') then
          C0 <= inb0,null after 2 ns;
        elsif (inb3 = '0') and (inb2 = '0') and (inb1 = '1') then
          C1 <= inb0,null after 2 ns;
        elsif (inb3 = '0') and (inb2 = '1') and (inb1 = '0') then
          C2 <= inb0,null after 2 ns;
          INTEB <= inb0, null after 5 ns;
        elsif (inb3 = '0') and (inb2 = '1') and (inb1 = '1') then
          C3 <= inb0,null after 2 ns;
        elsif (inb3 = '1') and (inb2 = '0') and (inb1 = '0') then
          C4 <= inb0,null after 2 ns;
          INTE2 <= inb0, null after 5 ns;
        elsif (inb3 = '1') and (inb2 = '0') and (inb1 = '1') then
          C5 <= inb0,null after 2 ns;
        elsif (inb3 = '1') and (inb2 = '1') and (inb1 = '0') then
          C6 <= inb0,null after 2 ns;
          INTEA <= inb0, null after 2 ns;
          INTE1 <= inb0, null after 2 ns;
        elsif (inb3 = '1') and (inb2 = '1') and (inb1 = '1') then
          C7 <= inb0,null after 2 ns;
        end if;
      end if;
    end if;
  wait on guard;
  end process;
end block BITSET;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- PORT A MODE-SELECTOR PROCESS
--
-- This is the mode selector process for port A.  It is activated when the
-- WRITE operation specified the alteration of mode configuration.
-- It will set the mode control for each port depending on the given
```

```
  -- configuration which has been written onto the internal transfer register
  -- in the WRITE process.
  process (cntl_en, RESET)
  begin

    -- Only functions if the chip select is enabled
    if (CS_BAR = '0') then

      -- if reset then put the chip to mode0 input
      if (RESET = '1') then
        pres_modea <= mode0a_input;

      -- otherwise, set the mode according to the given configuration
      elsif (cntl_en'EVENT) and (cntl_en = '1') then
        if (inb6 = '0') and (inb5 = '0') then
          if (inb4 = '1') then
            pres_modea <= mode0a_input;
          elsif (inb4 = '0') then
            pres_modea <= mode0a_output;
          end if;
        elsif (inb6 = '0') and (inb5 = '1') then
          if (inb4 = '1') then
            pres_modea <= modela_input;
          elsif (inb4 = '0') then
            pres_modea <= modela_output;
          end if;
        elsif (inb6 = '1') then
          pres_modea <= mode2a;
        end if;
      end if;
    end if;
  end process;


  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- PORT B MODE-SELECTOR PROCESS
  --
  -- This is the mode selector process for port B.  It is activated when the
  -- WRITE operation specified the alteration of mode configuration.
  -- It will set the mode control for each port depending on the given
  -- configuration which has been written onto the internal transfer register
  -- in the WRITE process.
  process (cntl_en, RESET)
  begin

    -- Only functions if the chip select is enabled
    if (CS_BAR = '0') then

      -- if reset then put the chip to mode0 input
      if (RESET = '1') then
        pres_modeb <= mode0b_input;

      -- otherwise, set the mode according to the given configuration
      elsif (cntl_en'EVENT) and (cntl_en = '1') then
        if (inb2 = '0')  then
          if (inb1 = '1') then
```



```
            pres_modeb <= mode0b_input;
          elsif (inb1 = '0') then
            pres_modeb <= mode0b_output;
          end if;
        elsif (inb2 = '1') then
          if (inb1 = '1') then
            pres_modeb <= modelb_input;
          elsif (inb1 = '0') then
            pres_modeb <= modelb_output;
          end if;
        end if;
      end if;
    end if;
  end process;


  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- UPPER PORT C MODE-SELECTOR PROCESS
  --
  -- This is the mode selector process for upper port C.  It is activated when
  -- the WRITE operation specified the alteration of mode configuration.
  -- It will set the mode control for each port depending on the given
  -- configuration which has been written onto the internal transfer register
  -- in the WRITE process.
  process (cntl_en, RESET)
  begin

    -- functions only if the chip select is enabled
    if (CS_BAR = '0') then

      -- if reset then put the upper 4 bits of port c into mode 0 input
      if (RESET = '1') then
        pres_mode_upc <= mode0upc_input;

      -- otherwise, set the upper 4 bits port c control to the mode specified
      -- by the configuration.
      elsif (cntl_en'EVENT) and (cntl_en = '1') then
        if (inb6 = '0') and (inb5 = '0') then
          if (inb3 = '1') then
            pres_mode_upc <= mode0upc_input;
          elsif (inb3 = '0') then
            pres_mode_upc <= mode0upc_output;
          end if;
        elsif (inb6 = '0') and (inb5 = '1') then
          if (inb4 = '1') then
            if (inb3 = '1') then
              pres_mode_upc <= modelupc67_input;
            elsif (inb3 = '0') then
              pres_mode_upc <= modelupc67_output;
            end if;
          elsif (inb4 = '0') then
            if (inb3 = '1') then
              pres_mode_upc <= modelupc45_input;
            elsif (inb3 = '0') then
              pres_mode_upc <= modelupc45_output;
            end if;
```

```
        end if;
      else
        pres_mode_upc <= null_mode;
      end if;
    end if;
  end if;
end process;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- LOWER PORT C MODE-SELECTOR PROCESS
--
-- This is the mode selector process for lower port C.  It is activated when
-- the WRITE operation specified the alteration of mode configuration.
-- It will set the mode control for each port depending on the given
-- configuration which has been written onto the internal transfer register
-- in the WRITE process.
process (cntl_en, RESET)
begin

  -- functions only if the chip select is enabled
  if (CS_BAR = '0') then

    -- if reset then put the lower 4 bits of port C into mode0 input mode
    if (RESET = '1') then
      pres_mode_lowc <= mode0lowc_input;

    -- otherwise, change the control of the lower 4 bits of port C according
    -- to the specified configuration.
    elsif (cntl_en'EVENT) and (cntl_en = '1') then
      if (inb2 = '0') and (inb6 = '0') and (inb5 = '0')  then
        if (inb0 = '1') then
          pres_mode_lowc <= mode0lowc_input;
        elsif (inb0 = '0') then
          pres_mode_lowc <= mode0lowc_output;
        end if;
      elsif (inb2 = '0') then
        if (inb1 = '1') then
          pres_mode_lowc <= modellowc_input;
        elsif (inb1 = '0') then
          pres_mode_lowc <= modellowc_output;
        end if;
      else
        pres_mode_lowc <= null_mode;
      end if;
    end if;
  end if;
end process;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- MODE0-PORT A-OUT PROCESS
--
-- This process perform the mode 0 output operation for port A.  This is
-- a simple output operation.  No "handshaking" is required, data is simply
-- written to the port.
```



```
mode0a_out:  block (pres_modea = mode0a_output and RESET = '0')

begin
  process
  begin
  if guard then

    -- wait until output enable signal is active then placed the content
    -- of the internal transfer register to the port.
    wait until (out_en_a='1');
    A0 <= inb0 after tWB, null after (tWB + 2 ns);
    A1 <= inb1 after tWB, null after (tWB + 2 ns);
    A2 <= inb2 after tWB, null after (tWB + 2 ns);
    A3 <= inb3 after tWB, null after (tWB + 2 ns);
    A4 <= inb4 after tWB, null after (tWB + 2 ns);
    A5 <= inb5 after tWB, null after (tWB + 2 ns);
    A6 <= inb6 after tWB, null after (tWB + 2 ns);
    A7 <= inb7 after tWB, null after (tWB + 2 ns);
  else
    A0 <= null;
    A1 <= null;
    A2 <= null;
    A3 <= null;
    A4 <= null;
    A5 <= null;
    A6 <= null;
    A7 <= null;
  end if;

  wait on guard;
  end process;
end block mode0a_out;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- MODE0-PORT A-IN PROCESS
--
-- This process perform the mode 0 input operation for port A.  This is
-- a simple input operation.  No "handshaking" is required, data is simply
-- read from the port and placed onto the internal transfer register.

mode0a_in: block (pres_modea = mode0a_input and RESET = '0' and
                  req_read_a = '1')
begin
  process
  begin

  -- place the content read from the port into the internal transfer register
  -- then enable the read acknowledge line so that the READ process can be
  -- awaken and place the content onto the CPU data bus.
  if guard then
    inb0 <= A0;
    inb1 <= A1;
    inb2 <= A2;
    inb3 <= A3;
    inb4 <= A4;
```

```
      inb5 <= A5;
      inb6 <= A6;
      inb7 <= A7;
      ack_read <= F1 , F0 after tRD,ZX after (tRD + 5 ns);
   else
     inb0 <= null;
     inb1 <= null;
     inb2 <= null;
     inb3 <= null;
     inb4 <= null;
     inb5 <= null;
     inb6 <= null;
     inb7 <= null;
     ack_read <= ZX;
   end if;
   wait on guard;
   end process;
end block mode0a_in;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- MODE1-PORT A-OUT PROCESS
-- This process performs the mode 1 output operations on port A.  This mode
-- provides a mean for transferring output data from the internal transfer
-- registers to the port A in conjunction with handshaking signals.

modela_out:  block ((pres_modea = modela_output or pres_modea = mode2a)
                    and RESET = '0' )
begin
  M1_out: process
    begin
    if guard then

      -- wait for the write enable
      wait until (out_en_a = '1' or RESET = '1');

      -- if interrupt masking register is set then send the interrupt to CPU
      if (INTEA = '1') then
        C3 <= F0 after tWIT, null after (tWIT + 2 ns);
      end if;

      -- placed output from data bus to the port
      A0 <= D0 after tWB, null after (tWB + 2 ns);
      A1 <= D1 after tWB, null after (tWB + 2 ns);
      A2 <= D2 after tWB, null after (tWB + 2 ns);
      A3 <= D3 after tWB, null after (tWB + 2 ns);
      A4 <= D4 after tWB, null after (tWB + 2 ns);
      A5 <= D5 after tWB, null after (tWB + 2 ns);
      A6 <= D6 after tWB, null after (tWB + 2 ns);
      A7 <= D7 after tWB, null after (tWB + 2 ns);

      -- if the data has been accepted (ACK_BAR) then indicate that the
      -- CPU has written data out to the port A (OBF_BAR).
      if (C6 = '1') then
        C7 <= F0 after tWOB;
      end if;
```



```
      -- deactivate the OBF_BAR line
      wait until (C6 = '0') or (RESET = '1');
      C7 <= F1 after tAOB;

      -- deactivate the interrupt line
      wait until ((C6 = '1') and (C7 = '1') and (WR_BAR = '1')) or
           (RESET = '1');
      C3 <= F1 after tAIT;
    else
      A0 <= null;
      A1 <= null;
      A2 <= null;
      A3 <= null;
      A4 <= null;
      A5 <= null;
      A6 <= null;
      A7 <= null;
      C3 <= null;
      C7 <= null;
    end if;
    wait on guard;
    end process M1_out;
end block modela_out;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- MODE1-PORT A-IN PROCESS
--
-- This process performs the mode 1 input operations on port A.  This mode
-- provides a means for transferring input data from port A to the internal
-- transfer registers in conjunction with strobes signals.
modela_in:  block ((pres_modea = modela_input or pres_modea = mode2a)
                    and RESET = '0' )
begin
  M1_in: process

    -- internal latches to hold the input data (latching effect)
    variable reg_a0,reg_a1,reg_a2,reg_a3,reg_a4,reg_a5,reg_a6,
             reg_a7 : t_wlogic;
    variable I : integer;
    begin
      if guard then

        -- wait until strobe signal (STB_BAR) active then read in value
        -- from port A into internal temporary register.

        wait until ((C4'EVENT) and (C4 = '0')) or (RESET = '1');
        reg_a0:= A0;
        reg_a1:= A1;
        reg_a2:= A2;
        reg_a3:= A3;
        reg_a4:= A4;
        reg_a5:= A5;
        reg_a6:= A6;
        reg_a7:= A7;
```

```
        -- indicate that the data has been loaded into the input latch;
        -- in essence, an acknowledgement. (IBF)
        C5 <= F1 after tSIB, null after (tSIB + 2 ns);

        -- interrupt the CPU when the device request the service.
        if (RD_BAR = '1') and (INTEA = '1') then
          wait until ((C5 = '1') and (C4 = '1')) or (RESET = '1');
          C3 <= F1 after tSIT, null after (tSIT + 2 ns);
        end if;

        -- wait until READ line is activated then transfer data from the
        -- internal latch to the internal transfer register.  Then send
        -- a read acknowledgement to activate READ process.

        wait until (read_req_a = '1') or (RESET = '1');
        inb0 <= reg_a0,null after tRD;
        inb1 <= reg_a1,null after tRD;
        inb2 <= reg_a2,null after tRD;
        inb3 <= reg_a3,null after tRD;
        inb4 <= reg_a4,null after tRD;
        inb5 <= reg_a5,null after tRD;
        inb6 <= reg_a6,null after tRD;
        inb7 <= reg_a7,null after tRD;
        ack_read <= F1 , F0 after tRD;

        -- deactivate the interrupt and buffer indicator lines.
        C3 <= F0 after tRIT, null after (tRIT + 2 ns);
        wait until ((C4 = '1') and (RD_BAR = '1')) or (RESET = '1');
        C5 <= F0 after tRIB, null after (tRIB + 2 ns);
      else
        C3 <= null;
        C5 <= null;
        inb0 <= null;
        inb1 <= null;
        inb2 <= null;
        inb3 <= null;
        inb4 <= null;
        inb5 <= null;
        inb6 <= null;
        inb7 <= null;
        ack_read <= null;
      end if;
      wait on guard;
    end process M1_in;
  end block modela_in;


  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- MODE0-PORT B-IN PROCESS
  --
  -- This process perform the mode 0 input operation for port B.  This is
  -- a simple input operation.  No "handshaking" is required, data is simply
  -- read from the port and placed onto the internal transfer register.
  mode0b_in: block (pres_modeb = mode0b_input and RESET = '0' and
                    req_read_b = '1')
```



```
  begin
    process
    begin

      -- place the content read from the port into the internal transfer
      -- registers then enable the read acknowledge line so that the READ
      -- process can be awaken and place the content onto the CPU data bus.

      if guard then
         inb0 <= B0;
         inb1 <= B1;
         inb2 <= B2;
         inb3 <= B3;
         inb4 <= B4;
         inb5 <= B5;
         inb6 <= B6;
         inb7 <= B7;
         ack_read <= F1 , F0 after tRD,ZX after (tRD + 5 ns);
      else
        inb0 <= null;
        inb1 <= null;
        inb2 <= null;
        inb3 <= null;
        inb4 <= null;
        inb5 <= null;
        inb6 <= null;
        inb7 <= null;
        ack_read <= ZX;
      end if;
      wait on guard;
      end process;
  end block mode0b_in;


  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- MODE0-PORT B-OUT PROCESS
  --
  -- This process perform the mode 0 output operation for port B.  This is
  -- a simple output operation.  No "handshaking" is required, data is simply
  -- written to the port.
  mode0b_out:  block (pres_modeb = mode0b_output and RESET = '0')

  begin
    process
    begin

      if guard then

         -- wait until output enable signal is active then placed the content
         -- of the internal transfer register to the port.
         wait until ((out_en_b='1') or (RESET = '1'));
         B0 <= inb0 after tWB, null after (tWB + 2 ns);
         B1 <= inb1 after tWB, null after (tWB + 2 ns);
         B2 <= inb2 after tWB, null after (tWB + 2 ns);
         B3 <= inb3 after tWB, null after (tWB + 2 ns);
         B4 <= inb4 after tWB, null after (tWB + 2 ns);
```

```
      B5 <= inb5 after tWB, null after (tWB + 2 ns);
      B6 <= inb6 after tWB, null after (tWB + 2 ns);
      B7 <= inb7 after tWB, null after (tWB + 2 ns);
    else
      B0 <= null;
      B1 <= null;
      B2 <= null;
      B3 <= null;
      B4 <= null;
      B5 <= null;
      B6 <= null;
      B7 <= null;
    end if;

    wait on guard;
    end process;
  end block mode0b_out;

  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- MODE1-PORT B-OUT PROCESS
  --
  -- This process performs the mode 1 output operations on port B.  This mode
  -- provides a means for transferring output data from the internal transfer
  -- registers to the port A in conjunction with handshaking signals.

  modelb_out:  block (pres_modeb = modelb_output and RESET = '0' )
  begin
    Mlb_out: process
      begin
      if guard then

        -- wait for the write enable
        wait until (out_en_b = '1' or RESET = '1');

        -- if interrupt masking register is set then send the interrupt to CPU
        if (INTEB = '1') then
          C0 <= F0 after tWIT, null after (tWIT + 2 ns);
        end if;

        -- placed output from data bus to the port
        B0 <= D0 after tWB, null after (tWB + 2 ns);
        B1 <= D1 after tWB, null after (tWB + 2 ns);
        B2 <= D2 after tWB, null after (tWB + 2 ns);
        B3 <= D3 after tWB, null after (tWB + 2 ns);
        B4 <= D4 after tWB, null after (tWB + 2 ns);
        B5 <= D5 after tWB, null after (tWB + 2 ns);
        B6 <= D6 after tWB, null after (tWB + 2 ns);
        B7 <= D7 after tWB, null after (tWB + 2 ns);

        -- if the data has been accepted (ACK_BAR) then indicate that the
        -- CPU has written data out to the port B (OBF_BAR).
        if (C2 = '1') then
          C1 <= F0 after tWOB;
        end if;

        -- deactivate the OBF_BAR line
```



```
        wait until (C2 = '0') or RESET = '1';
        C1 <= F1 after tAOB;

        -- deactivate the interrupt line
        wait until ((C2 = '1') and (C1 = '1') and (WR_BAR = '1')) or RESET='1';
        C0 <= F1 after tAIT;
      else
        B0 <= null;
        B1 <= null;
        B2 <= null;
        B3 <= null;
        B4 <= null;
        B5 <= null;
        B6 <= null;
        B7 <= null;
        C0 <= null;
        C1 <= null;
      end if;
      wait on guard;
      end process Mlb_out;
  end block modelb_out;

  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- MODE1-PORT B-IN PROCESS
  --
  -- This process performs the mode 1 input operations on port B.  This mode
  -- provides a means for transferring input data from port B to the internal
  -- transfer registers in conjunction with strobes signals.
  modelb_in:  block (pres_modeb = modelb_input and RESET = '0')
  begin
    Mlb_in: process

      -- internal latches to hold the input data (latching effect)
      variable reg_b0,reg_b1,reg_b2,reg_b3,reg_b4,reg_b5,reg_b6,
               reg_b7 : t_wlogic;
      variable I : integer;
      begin
        if guard then

          -- wait until strobe signal (STB_BAR) active then read in value
          -- from port B into internal temporary register.

          wait until ((C2'EVENT) and (C2 = '0')) or (RESET = '1');
          reg_b0:= B0;
          reg_b1:= B1;
          reg_b2:= B2;
          reg_b3:= B3;
          reg_b4:= B4;
          reg_b5:= B5;
          reg_b6:= B6;
          reg_b7:= B7;

          -- indicate that the data has been loaded into the input latch;
          -- in essence, an acknowledgement. (IBF)
          C1 <= F1 after tSIB, null after (tSIB + 2 ns);
```

```
        -- interrupt the CPU when the device request the service.
        if (RD_BAR = '1') and (INTEB = '1') then
          wait until ((C1 = '1') and (C2 = '1')) or (RESET = '1');
          C0 <= F1 after tSIT, null after (tSIT + 2 ns);
        end if;

        -- wait until READ line is activated then transfer data from the
        -- internal latch to the internal transfer register.  Then send
        -- a read acknowledgement to activate READ process.
        wait until ((RD_BAR = '0') or (RESET = '1'));
        inb0 <= reg_b0,null after tRD;
        inb1 <= reg_b1,null after tRD;
        inb2 <= reg_b2,null after tRD;
        inb3 <= reg_b3,null after tRD;
        inb4 <= reg_b4,null after tRD;
        inb5 <= reg_b5,null after tRD;
        inb6 <= reg_b6,null after tRD;
        inb7 <= reg_b7,null after tRD;
        ack_read <= F1 , F0 after tRD;

        -- deactivate the interrupt and buffer indicator lines.
        C0 <= F0 after tRIT, null after (tRIT + 2 ns);
        wait until ((C2 = '1') and (RD_BAR = '1')) or (RESET = '1');
        C1 <= F0 after tRIB, null after (tRIB + 2 ns);
      else
        C0 <= null;
        C1 <= null;
        inb0 <= null;
        inb1 <= null;
        inb2 <= null;
        inb3 <= null;
        inb4 <= null;
        inb5 <= null;
        inb6 <= null;
        inb7 <= null;
        ack_read <= null;
      end if;
      wait on guard;
    end process M1b_in;
end block mode1b_in;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- MODE0-LOWER PORT C-IN PROCESS
--
-- This process performs the mode 0 input operation for lower port C.  This is
-- a simple input operation.  No "handshaking" is required, data is simply
-- read from the port and placed onto the internal transfer register.
mode0lowc_in: block (pres_mode_lowc = mode0lowc_input and RESET = '0' and
                     req_read_c = '1')
begin
  process
  begin

    -- place the content read from the port into the internal transfer
    -- registers then enable the read acknowledge line so that the READ
    -- process can be awaken and place the content onto the CPU data bus.
```



```
    if guard then
       inb0 <= C0;
       inb1 <= C1;
       inb2 <= C2;
       inb3 <= C3;
       ack_read <= F1 , F0 after tRD,ZX after (tRD + 5 ns);
    else
      inb0 <= null;
      inb1 <= null;
      inb2 <= null;
      inb3 <= null;
      ack_read <= ZX;
    end if;
    wait on guard;
    end process;
end block mode0lowc_in;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- MODE0-LOWER PORT C-OUT PROCESS
--
-- This process performs the mode 0 output operation for lower port C.
-- This is a simple output operation.  No "handshaking" is required, data is
-- simply written to the port.
mode0lowc_out:  block (pres_mode_lowc = mode0lowc_output and RESET = '0' )

begin
  process
  begin
  if guard then

     -- wait until output enable signal is active then placed the content
     -- of the internal transfer register to the port.
     wait until (out_en_c = '1');
     C0 <= inb0 after tWB, null after (tWB + 2 ns);
     C1 <= inb1 after tWB, null after (tWB + 2 ns);
     C2 <= inb2 after tWB, null after (tWB + 2 ns);
     C3 <= inb3 after tWB, null after (tWB + 2 ns);
   else
     C0 <= null;
     C1 <= null;
     C2 <= null;
     C3 <= null;
   end if;

   wait on guard;
   end process;
end block mode0lowc_out;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- MODE1-LOWER PORT C-IN PROCESS
--
-- This process performs mode 1 input operation for lower port C.  Basically,
-- in this mode only two bits of lower port C, namely bit 0 and bit 1, will
-- acts as the output port.  This is a simple input operation.
modellowc_in: block (pres_mode_lowc = modellowc_input and RESET = '0' and
                     req_read_c = '1')
```

```
  begin
    process
    begin
    if guard then
       inb0 <= C0;
       inb1 <= C1;
       inb2 <= C2;
       ack_read <= F1 , F0 after tRD,ZX after (tRD + 5 ns);
    else
      inb0 <= null;
      inb1 <= null;
      inb2 <= null;
      ack_read <= ZX;
    end if;
    wait on guard;
    end process;
  end block modellowc_in;


  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- MODE1-LOWER PORT C-OUT PROCESS
  --
  -- This process performs mode 1 output operation for lower port C.  Basically,
  -- in this mode only two bits of lower port C, namely bit 0 and bit 1, will
  -- acts as the input port.  This is a simple output operation.
  modellowc_out:  block (pres_mode_lowc = modellowc_output and RESET = '0' and
                         out_en_c = '1')
  begin
    process
    begin
    if guard then
       C0 <= inb0;
       C1 <= inb1;
       C2 <= inb2;
     else
       C0 <= null;
       C1 <= null;
       C2 <= null;
     end if;

    wait on guard;
    end process;
  end block modellowc_out;

  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- MODE0-UPPER PORT C-IN PROCESS
  --
  -- This process performs the mode 0 input operation for upper port C. This is
  -- a simple input operation.  No "handshaking" is required, data is simply
  -- read from the port and placed onto the internal transfer register
  mode0upc_in: block (pres_mode_upc = mode0upc_input and RESET = '0' and
                      req_read_c = '1')
  begin
    process
    begin
```



```
    -- place the content read from the port into the internal transfer
    -- registers then enable the read acknowledge line so that the READ
    -- process can be awaken and place the content onto the CPU data bus.
    if guard then
       inb4 <= C4;
       inb5 <= C5;
       inb6 <= C6;
       inb7 <= C7;
       ack_read <= F1 , F0 after tRD,ZX after (tRD + 5 ns);
    else
       inb4 <= null;
       inb5 <= null;
       inb6 <= null;
       inb7 <= null;
       ack_read <= ZX;
    end if;
    wait on guard;
    end process;
  end block mode0upc_in;

  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- MODE0-UPPER PORT C-OUT PROCESS
  --
  -- This process performs the mode 0 output operation for upper port C.
  -- This is a simple output operation.  No "handshaking" is required, data is
  -- simply written to the port.
  mode0upc_out:  block (pres_mode_upc = mode0upc_output and RESET = '0')

  begin
    process
    begin
    if guard then

       -- wait until output enable signal is active then placed the content
       -- of the internal transfer register to the port.
       wait until (out_en_c = '1') or RESET = '1';
       C4 <= inb4 after tWB, null after (tWB + 2 ns);
       C5 <= inb5 after tWB, null after (tWB + 2 ns);
       C6 <= inb6 after tWB, null after (tWB + 2 ns);
       C7 <= inb7 after tWB, null after (tWB + 2 ns);
     else
       C4 <= null;
       C5 <= null;
       C6 <= null;
       C7 <= null;
     end if;

    wait on guard;
    end process;
  end block mode0upc_out;


  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- MODE1-UPPER PORT C BIT4,5-IN PROCESS
  --
  -- This process performs mode 1 input operation for upper port C.  Basically,
```

```
-- in this mode only two bits of upper port C, namely bit 4 and bit 5, will
-- acts as the output port.  This is a simple input operation.
modelupc45_in: block (pres_mode_upc = modelupc45_input and RESET = '0' and
                      req_read_c = '1')
begin
  process
  begin
  if guard then
     inb4 <= C4;
     inb5 <= C5;
     ack_read <= F1 , F0 after tRD,ZX after (tRD + 5 ns);
  else
     inb4 <= null;
     inb5 <= null;
     ack_read <= ZX;
  end if;
  wait on guard;
  end process;
end block modelupc45_in;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- MODE1-UPPER PORT C BIT4,5-OUT PROCESS
--
-- This process performs mode 1 output operation for upper port C. Basically,
-- in this mode only two bits of upper port C, namely bit 4 and bit 5, will
-- acts as the input port.  This is a simple output operation.
modelupc45_out:  block (pres_mode_upc = modelupc45_output and RESET = '0' and
                        out_en_c = '1')
begin
  process
  begin
  if guard then
     C4 <= inb4;
     C5 <= inb5;
   else
     C4 <= null;
     C5 <= null;
   end if;

   wait on guard;
   end process;
end block modelupc45_out;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- MODE1-UPPER PORT C BIT6,7-IN PROCESS
-- This process performs mode 1 input operation for upper port C.  Basically,
-- in this mode only two bits of upper port C, namely bit 6 and bit 7, will
-- acts as the output port.  This is a simple input operation.
modelupc67_in: block (pres_mode_upc = modelupc67_input and RESET = '0' and
                      req_read_c = '1')
begin
  process
  begin
  if guard then
     inb6 <= C6;
     inb7 <= C7;
```



```
     ack_read <= F1 , F0 after tRD,ZX after (tRD + 5 ns);
  else
     inb6 <= null;
     inb7 <= null;
     ack_read <= ZX;
  end if;
  wait on guard;
  end process;
end block modelupc67_in;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- MODE1-UPPER PORT C BIT6,7 OUT PROCESS
--
-- This process performs mode 1 output operation for upper port C.  Basically,
-- in this mode only two bits of upper port C, namely bit 6 and bit 7, will
-- acts as the output port.  This is a simple output operation.
modelupc67_out:  block (pres_mode_upc = modelupc67_output and RESET = '0' and
                        out_en_c = '1')
begin
  process
  begin
  if guard then
     C6 <= inb6;
     C7 <= inb7;
   else
     C6 <= null;
     C7 <= null;
   end if;

   wait on guard;
   end process;
end block modelupc67_out;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- PORT A-INTERFACE PROCESSES
--
-- The following processes are used as input/output interface from the
-- internal registers holding content of port A to the external bus A.
-- The reason we have to use such interface because as a bus, when the
-- driver is deactivated, its value will be re-calculate.  But we want to
-- retain those value, hence, we have to use registers as buffers.
process (A0)
  begin
    PA0 <= A0;
  end process;
process (A1)
  begin
    PA1 <= A1;
  end process;
process (A2)
  begin
    PA2 <= A2;
  end process;
process (A3)
  begin
```

```
    PA3 <= A3;
  end process;
process (A4)
  begin
    PA4 <= A4;
  end process;
process (A5)
  begin
    PA5 <= A5;
  end process;
process (A6)
  begin
    PA6 <= A6;
  end process;
process (A7)
  begin
    PA7 <= A7;
  end process;
process (PA0)
  begin
    A0 <= PA0, null after 1 ns;
  end process;
process (PA1)
  begin
    A1 <= PA1, null after 1 ns;
  end process;
process (PA2)
  begin
    A2 <= PA2, null after 1 ns;
  end process;
process (PA3)
  begin
    A3 <= PA3, null after 1 ns;
  end process;
process (PA4)
  begin
    A4 <= PA4, null after 1 ns;
  end process;
process (PA5)
  begin
    A5 <= PA5, null after 1 ns;
  end process;
process (PA6)
  begin
    A6 <= PA6,null after 1 ns;
  end process;
process (PA7)
  begin
    A7 <= PA7,null after 1 ns;
  end process;
process (B0)
  begin
    PB0 <= B0;
  end process;
```



```
-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- PORT B-INTERFACE PROCESSES
--
-- The following processes are used as input/output interface from the
-- internal registers holding content of port A to the external bus A.
-- The reason we have to use such interface because as a bus, when the
-- driver is deactivated, its value will be re-calculate.  But we want to
-- retain those value, hence, we have to use registers as buffers.
process (B1)
  begin
    PB1 <= B1;
  end process;
process (B2)
  begin
    PB2 <= B2;
  end process;
process (B3)
  begin
    PB3 <= B3;
  end process;
process (B4)
  begin
    PB4 <= B4;
  end process;
process (B5)
  begin
    PB5 <= B5;
  end process;
process (B6)
  begin
    PB6 <= B6;
  end process;
process (B7)
  begin
    PB7 <= B7;
  end process;
process (PB0)
  begin
    B0 <= PB0, null after 1 ns;
  end process;
process (PB1)
  begin
    B1 <= PB1, null after 1 ns;
  end process;
process (PB2)
  begin
    B2 <= PB2, null after 1 ns;
  end process;
process (PB3)
  begin
    B3 <= PB3, null after 1 ns;
  end process;
process (PB4)
  begin
    B4 <= PB4, null after 1 ns;
  end process;
```

```
  process (PB5)
    begin
      B5 <= PB5, null after 1 ns;
    end process;
  process (PB6)
    begin
      B6 <= PB6,null after 1 ns;
    end process;
  process (PB7)
    begin
      B7 <= PB7,null after 1 ns;
    end process;


  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- PORT C-INTERFACE PROCESSES
  --
  -- The following processes are used as input/output interface from the
  -- internal registers holding content of port A to the external bus A.
  -- The reason we have to use such interface because as a bus, when the
  -- driver is deactivated, its value will be re-calculate.  But we want to
  -- retain those value, hence, we have to use registers as buffers.
  process (C0)
    begin
      PC0 <= C0;
    end process;
  process (C1)
    begin
      PC1 <= C1;
    end process;
  process (C2)
    begin
      PC2 <= C2;
    end process;
  process (C3)
    begin
      PC3 <= C3;
    end process;
  process (C4)
    begin
      PC4 <= C4;
    end process;
  process (C5)
    begin
      PC5 <= C5;
    end process;
  process (C6)
    begin
      PC6 <= C6;
    end process;
  process (C7)
    begin
      PC7 <= C7;
    end process;
  process (PC0)
    begin
```



```
      C0 <= PC0, null after 1 ns;
    end process;
  process (PC1)
    begin
      C1 <= PC1, null after 1 ns;
    end process;
  process (PC2)
    begin
      C2 <= PC2, null after 1 ns;
    end process;
  process (PC3)
    begin
      C3 <= PC3, null after 1 ns;
    end process;
  process (PC4)
    begin
      C4 <= PC4, null after 1 ns;
    end process;
  process (PC5)
    begin
      C5 <= PC5, null after 1 ns;
    end process;
  process (PC6)
    begin
      C6 <= PC6,null after 1 ns;
    end process;
  process (PC7)
    begin
      C7 <= PC7,null after 1 ns;
    end process;

  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- This process acts as a timming constraint checker for RD_BAR pulse width
    process (RD_BAR)
      variable readlastevent : time := 0 ns;
      begin
      if (RD_BAR'EVENT) and (RD_BAR = '1') then
        assert (NOW = 0 ns) or ((NOW - readlastevent) >= tRR)
          report "RD_BAR pulse width is less than tRR" severity warning;
      elsif (RD_BAR'EVENT) and (RD_BAR = '0') then
        readlastevent := NOW;
      end if;
      end process;

  -- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  -- This process acts as a timming constraint checker for WR_BAR pulse width
    process (WR_BAR)
      variable writelastevent : time := 0 ns;
      begin
      if (WR_BAR'EVENT) and (WR_BAR = '1') then
        assert (NOW = 0 ns) or ((NOW - writelastevent) >= tWW)
          report "WR_BAR pulse width is less than tWW" severity warning;
      elsif (WR_BAR'EVENT) and (WR_BAR = '0') then
        writelastevent := NOW;
      end if;
      end process;
```

```
-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- This process acts as a timming constraint checker for hold/setup time
-- for address after/before read pulse width

  process (RD_BAR, A0_MODE, A1_MODE, CS_BAR)
    variable readlastevent : time := 0 ns;
    variable addresslastevent : time := 0 ns;
    begin
    if (A0_MODE'EVENT) or (A1_MODE'EVENT) or (CS_BAR'EVENT) then
      assert (NOW = 0 ns) or ((NOW - readlastevent) >= tRA)
        report "Address stable not longer than tRA after read pulse"
          severity warning;
      addresslastevent := NOW;
    end if;

    if (RD_BAR'EVENT) and (RD_BAR = '0') then
      assert (NOW = 0 ns) or ((NOW - addresslastevent) >= tAR)
        report "Address stable not longer than tAR before read pulse"
          severity warning;
    elsif (RD_BAR'EVENT) and (RD_BAR = '1') then
      readlastevent := NOW;
    end if;
    end process;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- This process acts as a timming constraint checker for hold/setup time
-- for address after/before write pulse width

  process (WR_BAR, A0_MODE, A1_MODE, CS_BAR)
    variable writelastevent : time := 0 ns;
    variable addresslastevent : time := 0 ns;
    begin
    if (A0_MODE'EVENT) or (A1_MODE'EVENT) or (CS_BAR'EVENT) then
      assert (NOW = 0 ns) or ((NOW - writelastevent) >= tWA)
        report "Address stable not longer than tWA after write pulse"
          severity warning;
      addresslastevent := NOW;
    end if;

    if (WR_BAR'EVENT) and (WR_BAR = '0') then
      assert (NOW = 0 ns) or ((NOW - addresslastevent) >= tAW)
        report "Address stable not longer than tAW before write pulse"
          severity warning;
    elsif (WR_BAR'EVENT) and (WR_BAR = '1') then
      writelastevent := NOW;
    end if;
    end process;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- This process acts as a timming constraint checker for time between read
-- and write pulse

  process (RD_BAR, WR_BAR)
    variable readlastevent : time := 0 ns;
    variable writelastevent : time := 0 ns;
```



```
    begin
    if (RD_BAR'EVENT) and (RD_BAR = '0') then
      assert (NOW = 0 ns) or ((NOW - writelastevent) >= tRV) or
             (writelastevent = 0 ns)
        report "Time between read and write is smaller than tRV"
          severity warning;
        readlastevent := NOW;
    end if;

    if (WR_BAR'EVENT) and (WR_BAR = '0') then
      assert (NOW = 0 ns) or ((NOW - readlastevent) >= tRV) or
             (readlastevent = 0 ns)
        report "Time between read and write is smaller than tRV"
          severity warning;
        writelastevent := NOW;
    end if;

    end process;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- This process acts as a timming constraint checker for hold/setup time for
-- data after/before write pulse width
  process (WR_BAR,D0,D1,D2,D3,D4,D5,D6,D7)
    variable writelastevent : time := 0 ns;
    variable writelow : integer := 0;
    variable datalastevent : time := 0 ns;
    begin
    if ((D0'EVENT) or (D1'EVENT) or (D2'EVENT) or
       (D3'EVENT) or (D4'EVENT) or (D5'EVENT) or
       (D6'EVENT) or (D7'EVENT)) and (CS_BAR'EVENT)
       and (WR_BAR = '1')then
      assert (NOW = 0 ns) or ((NOW - writelastevent) >= tWD)
        report "Data valid after write is shorter than tWD"
          severity warning;
    elsif ((D0'EVENT) or (D1'EVENT) or (D2'EVENT) or
       (D3'EVENT) or (D4'EVENT) or (D5'EVENT) or
       (D6'EVENT) or (D7'EVENT)) and (CS_BAR'EVENT)
       and (WR_BAR = '0') and (writelow = 1)then
      datalastevent := NOW;
    end if;

    if (WR_BAR'EVENT) and (WR_BAR = '1') then
      assert (NOW = 0 ns) or ((NOW - datalastevent) >= tDW)
        report "Data valid before write is shorter than tDW"
          severity warning;
      writelastevent := NOW;
      writelow := 0;
    elsif (WR_BAR'EVENT) and (WR_BAR = '0') then
      writelow := 1;
    end if;
    end process;

end i8255a_a;
```

```
--- configuration declaration
configuration i8255a_c of i8255a_e is
  for i8255a_a
  end for;
end i8255a_c;
```