BEHAVIORAL MODELING OF DRACO: A PERIPHERAL INTERFACE ASIC

by

Rajesh Gupta
Nikil D. Dutt

Technical Report 90-13

Information and Computer Science
University of California at Irvine
Irvine, CA 92717

Keywords

ASIC Design Modeling, Design Specification, VHDL, Reverse Engineering from Data Sheets.

Behavioral Modeling of DRACO: A Peripheral Interface ASIC

by

Rajesh Gupta
Nikil D. Dutt

Abstract

This paper describes the behavioral modeling of **DRACO**, a peripheral interface Application Specific Integrated Circuit (ASIC) developed by Rockwell International for numerical control applications. The behavioral model was generated from a data sheet of the fabricated chip, which primarily described the chip's input-output functionality, physical and operational characteristics, and a functional block diagram. The data sheet contained very little abstract behavioral information. This report describes the abstract behavioral model of the DRACO chip, and uses flowcharts and VHDL to capture the behavior. The behavioral model was developed through reverse engineering of the data sheet description, supplemented by further consultation with designers of the DRACO ASIC at Rockwell International. The report describes typical behavioral test sequences that were applied to the DRACO VHDL model to verify its correctness. The appendices contain the original DRACO datasheet and the VHDL code used to capture DRACO's behavior.

TABLE OF CONTENTS

CHAPTER

1. INTRODUCTION

A commercial chip design is typically described using logic schematics and data sheets which give a structural and functional view of the design from a logic designer's perspective. Unfortunately, such a description does not describe the abstract behavior of the design in a complete fashion. Although some of this behavioral information may be present in each chip's data sheets and schematics, the lack of complete, more rigorous behavioral descriptions of chip designs is a pressing problem faced by many system houses, chip manufacturers and end users. With technological changes occurring at such a rapid pace, chip designs can become obsolete quickly, requiring retargetting of the intial design specification to a new technology or library. Since there is no well documented behavioral description of the design, retargetting of the design is a tedious process involving the reverse engineering of the schematics and data sheets to understand the abstract behavior of the design. This means a longer time to design, and therefore a longer time to a finished product, in a market where chip designs get obsolete very quickly.

Furthermore, chip complexities are increasing at a tremendous pace; by the year 1994, we can expect to see a microprocessor-on-a-chip with 6 *million transistors* on a 750 *sq. mm. die*, delivering 200 mips of performance running on a 100 Mhz clock [Sumn89]. To cope with this explosion of design complexity, there is an urgent need for design tools that capture designs at higher levels of description, and which automate higher levels of the design process, so that design alternatives can be explored quickly and accurately. Moreover, several nationally recognized figures have indicated that the competitiveness of the U.S. semiconductor industry is dependent on our ability to integrate tools that permit rapid turnaround of chip designs, from concept, all the way to manufacturing [IEEE90].

All of these indicators underscore the need for better design specifications using more rigorous media such as behavioral hardware description languages. Such specifications can provide behavioral models for simulation, verification and synthesis of designs. They also document the design in a systematic, comprehensible fashion, removing the need for reverse engineering of lower level descriptions.

In this report, we attempt to describe the behavior of a commercial chip design (DRACO) developed by Rockwell International. Rockwell's design specification for DRACO consisted of a data sheet and a set of VHDL netlists (schematics) representing the final chip design. The functionality in the data sheet only described the block diagram of the chip and a description of its input-output characteristics. There was no comprehensive abstract behavioral description of the chip available at Rockwell International. As a result, there were no behavioral test cases available, nor were there any typical design scenarios for the chip.

Using this data sheet description, an abstract behavioral model of the chip was developed using simple flow charts. This required some reverse engineering in order to avoid references to specific hardware constructs. The flowcharts deliberately use pseudo-code instead of a particular hardware description language. This facilitated easier development of the behavioral model. Inconsistencies and clarifications were resolved by communication with designers at Rockwell International [Lars90] [Pase90]. Once the complete behavior of the chip had been described, a set of behavioral test scenarios were developed to test typical operational sequences of the chip.

At this point, a behavioral VHDL (VHSIC Hardware Description Language) model of the DRACO chip was developed. This model was subjected to test stimuli corresponding to

the typical operational scenarios developed previously, to verify its operational correctness.

This report begins with the functional description and structural view of the DRACO chip in sections 2 and 3. Section 4 describes the behavioral model of the DRACO chip using flowcharts and pseudocode. Section 5 describes how DRACO was modeled in behavioral VHDL. Section 6 gives five typical operational sequences that were used to test the behavioral VHDL model. Appendix I contains Rockwell's data sheet for DRACO[1], while appendix II lists the actual behavioral VHDL code used to model DRACO.

2. ROCKWELL DRACO CHIP

DRACO is a peripheral interface Application Specific Integrated Circuit (ASIC) developed by Rockwell International for numerical control applications. This section reviews the functional and structural characteristics of the DRACO chip as presented in the Rockwell DRACO data sheet. Appendix I contains the actual data sheet for DRACO, which has a more detailed description of DRACO's functionality.

2.1. Functional Description of DRACO

DRACO's basic function is to interface 16 I/O ports to a microprocessor's 8 bit multiplexed address/data bus and control signals. DRACO may be connected remotely through an 18 inch long ribbon cable. Such a configuration may introduce errors in the transmitted signal due to Electro Magnetic Interference (EMI). To minimize any danger that may be caused by the receipt of corrupt data, several security features have been built into

[1] Rockwell International has granted U.C. Irvine permission to duplicate the data sheets for educational purposes.

DRACO. These special features are:

(1) *Hardware Key*: DRACO has a key which must be unlocked prior to configuring the chip. This configuration protocol adds an extra level of security since an incorrect configuration of the chip could result in considerable operational havoc. Furthermore, configuration registers may be written into only when the configuration is unlocked and the I/O ports may be written into only when data is unlocked.

(2) *Address Parity Check*: DRACO may optionally be configured to perform parity checks on all received addresses from the host.

(3) *Data Parity Check*: DRACO may optionally be configured to perform data parity checks on data received from the host and may generate parity while loading data onto the address bus.

(4) *Checksum*: DRACO generates an inverted checksum from the data to be output, compares it with the the received checksum and updates the I/O ports only if the checksums are equivalent.

Such extensive error checking measures ensures nearly error-free operation in the presence of EMI.

DRACO's input-output configuration is presented in Figure 1. A description of some of DRACO's pins follows:

ADD_DATA_BUS: The bus transfers address and data from the host to DRACO and data from DRACO to the host.

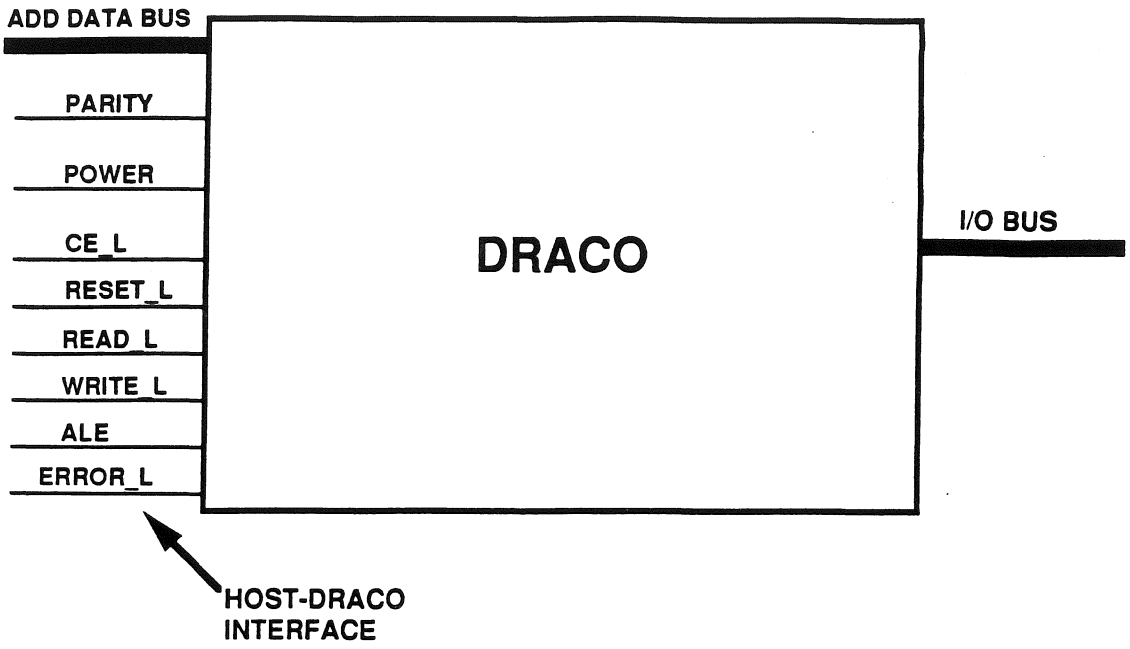


Figure 1. DRACO's I/O Configuration

PARITY: Carries address and data parity from the host to DRACO and data parity from DRACO to the host.

POWER: Indicates power on / power off status.

RESET_L: This input is used to initialize all internal registers and latches; it should be held low after power is applied.

READ_L: A low on this input causes internal read data to be placed on the address data bus and the parity to be placed on the parity pin when data is enabled.

WRITE_L: A low to high transition on this input causes external data on the address data bus and parity pins (when data parity is enabled) to be written into DRACO.

CE_L: This is a Chip enable input, which must be held low to execute a read or write cycle.

IO_BUS: These pins carry the output data and the data to be read in from DRACO.

ERROR_L: This is an active low output which is asserted whenever an error occurs in the data transmission between DRACO and the host microprocessor. This output will be latched low and must be reset by the user.

2.2. DRACO's Structural Model

The DRACO's structural model is shown in Figure 2. The structure consists of 6 registers and 4 D Flip Flops which store the data and DRACO's configuration.

3. BEHAVIORAL MODEL OF DRACO

The behavior of DRACO can be naturally modeled using a state transition diagram consisting of the following 8 primary states:

(1) Reset State

(2) Chip enabled

(3) Address Cycle

(4) Read Cycle

(5) Write Cycle

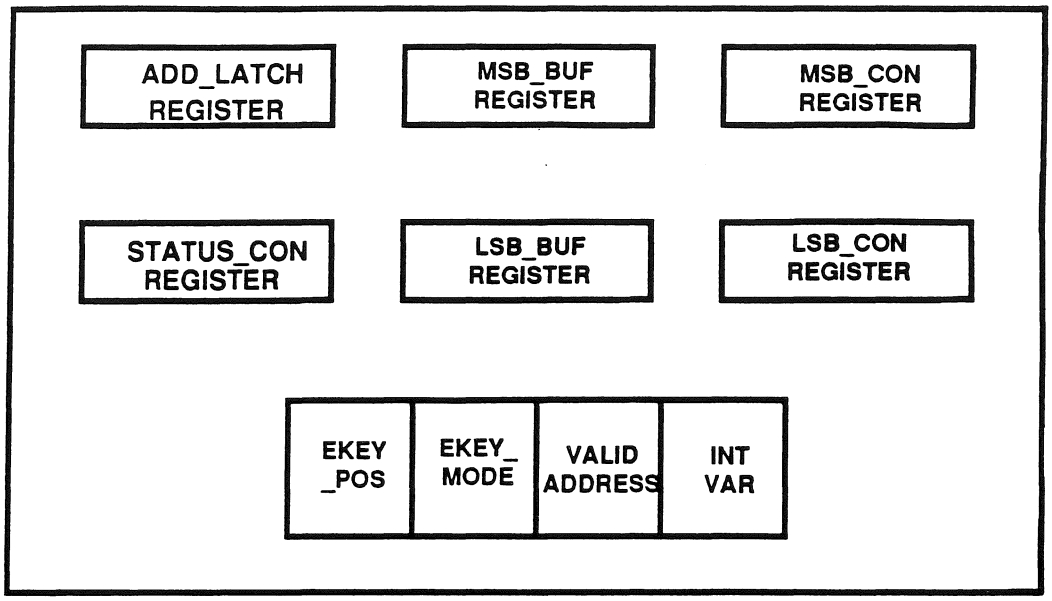


Figure 2. Behavioral Structure For DRACO

(6) Idle

(7) Chip Disabled

(8) Power Off

Figure 3 shows the state transition diagram using these eight states.

A typical initial sequence of operations for DRACO would involve turning the power on (Power-Up to Reset State), enabling the chip (Reset to Chip Enable State) and then resetting the chip (Chip Enable to Reset State) so as to configure DRACO using default settings (data/address parity off, ports set to be bidirectional, etc.). Subsequently, data can

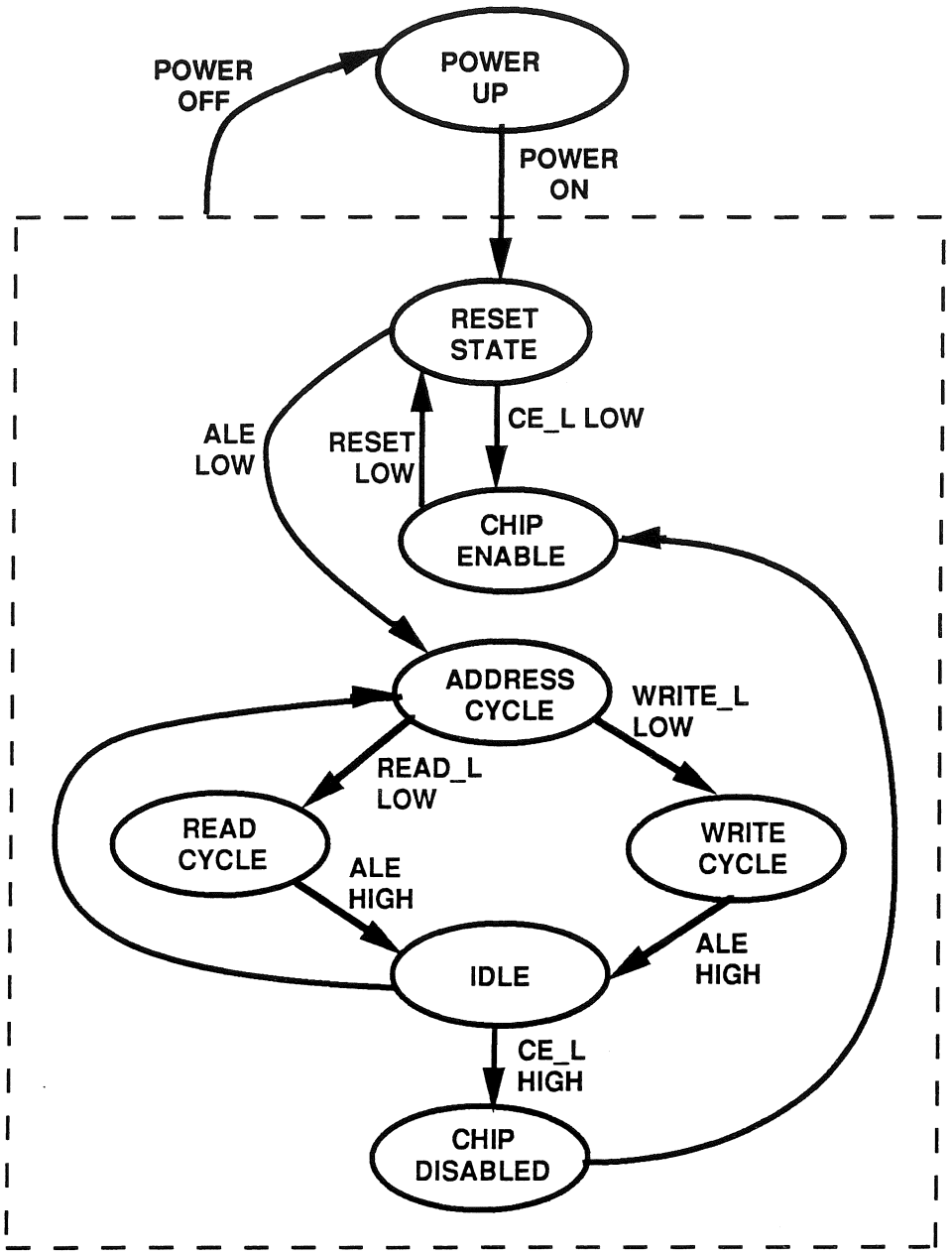


Figure 3. DRACO State Diagram

be written into or read from DRACO.

For a data access from DRACO, the chip passes through Address Cycle and the Read Cycle. The following sequence of events occurs: Address appears on the address/data bus, ALE goes low, READ_L goes low and finally data is placed by DRACO on the address/data bus. When ALE goes low, data from the bus is latched into DRACO if it is valid. DRACO places data on the bus a specified time after the READ_L signal goes low.

For writing to DRACO, the chip sequences through the Address Cycle and the Write Cycle. The following sequence of events occurs: Address appears on the address/data bus, ALE goes low, WRITE_L goes low, data appears on the address/data bus, and WRITE_L goes high. When ALE goes low, the address, if valid, is latched into DRACO. When WRITE_L goes high, data is written into DRACO.

DRACO is in the Idle State when power is on and the chip is enabled, but is not executing the Read, Write or Address Cycles. During this state ALE, READ_L and WRITE_L are all high. DRACO enters this state after the Read and Write cycles.

Whenever power is switched on, the chip immediately sequences to the RESET State; there is no "Power On" State, since this is effectively the Reset State.

Each of the the 4 states *Address Cycle*, *Write Cycle*, *Read Cycle* and *Reset Cycle* are described by secondary sequential state diagrams. The flowcharts and pseudo-code for the Address, Write, Read and Reset Cycles are given in Figures 4, 5, 6 and 7 respectively. These flowcharts should be fairly self-explanatory.

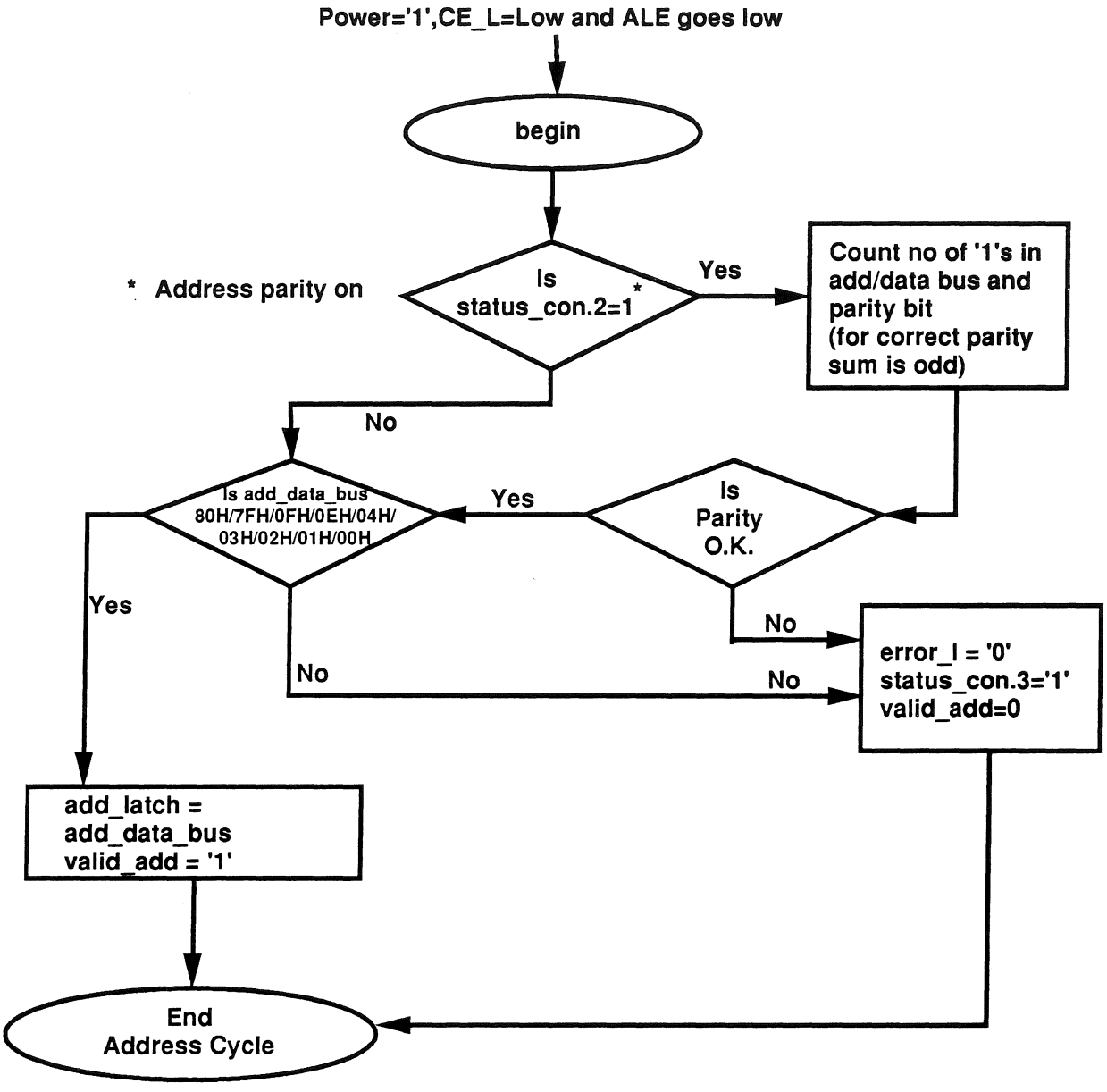


Figure 4. Secondary State Diagram for ADDRESS CYCLE

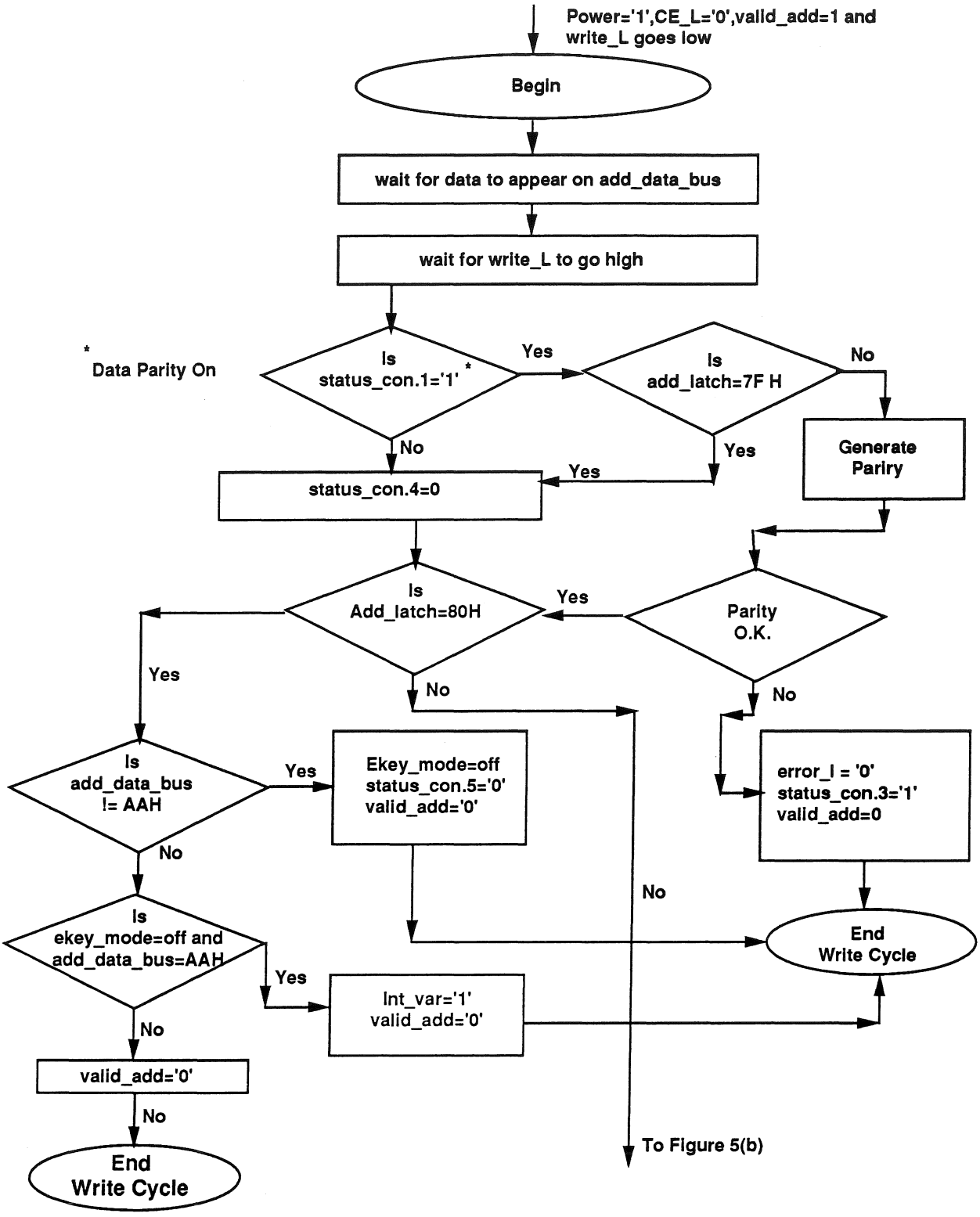


Figure 5(a). Secondary State Diagram of WRITE CYCLE

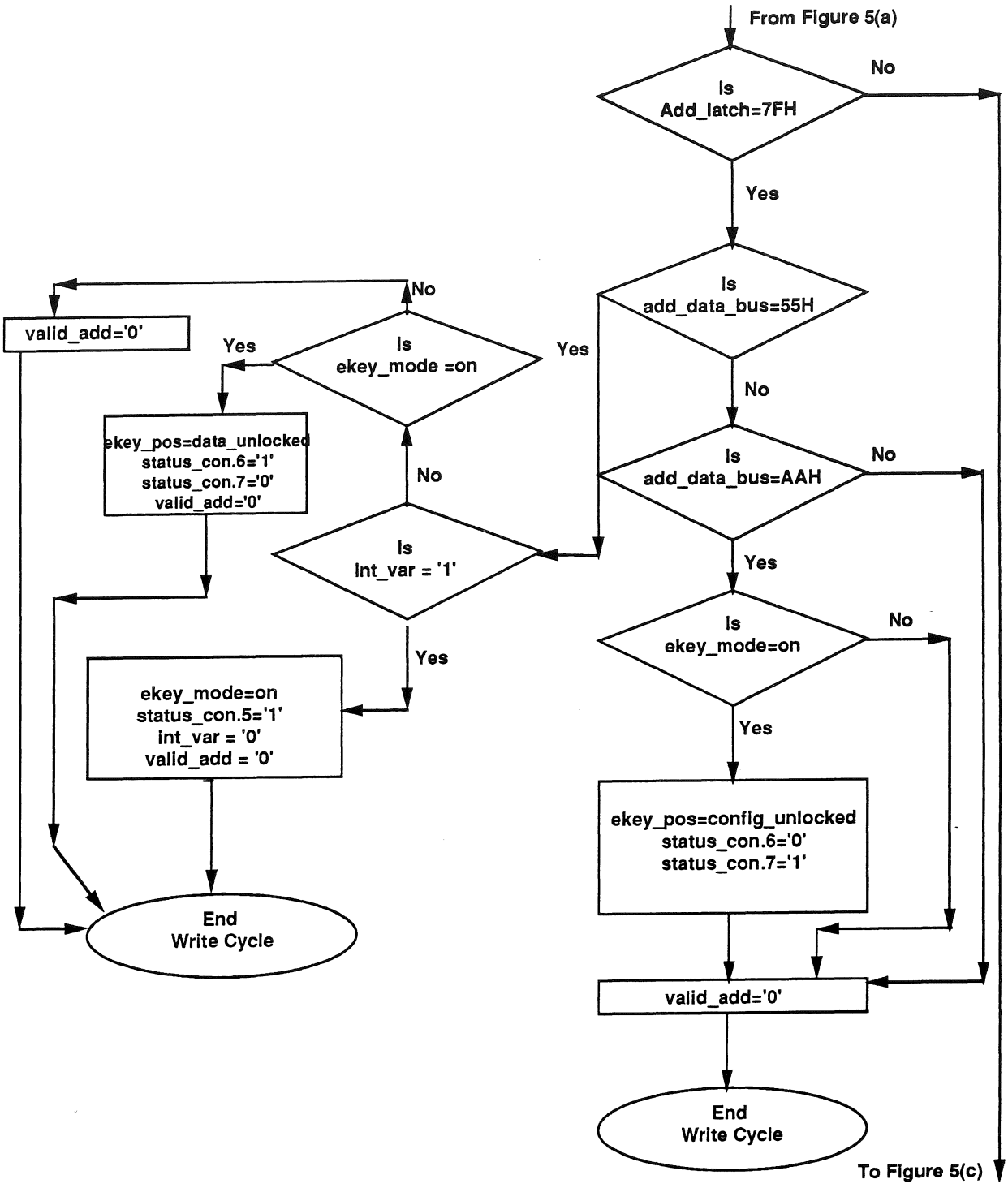


Figure 5(b). Secondary State Diagram of WRITE CYCLE (contd.)

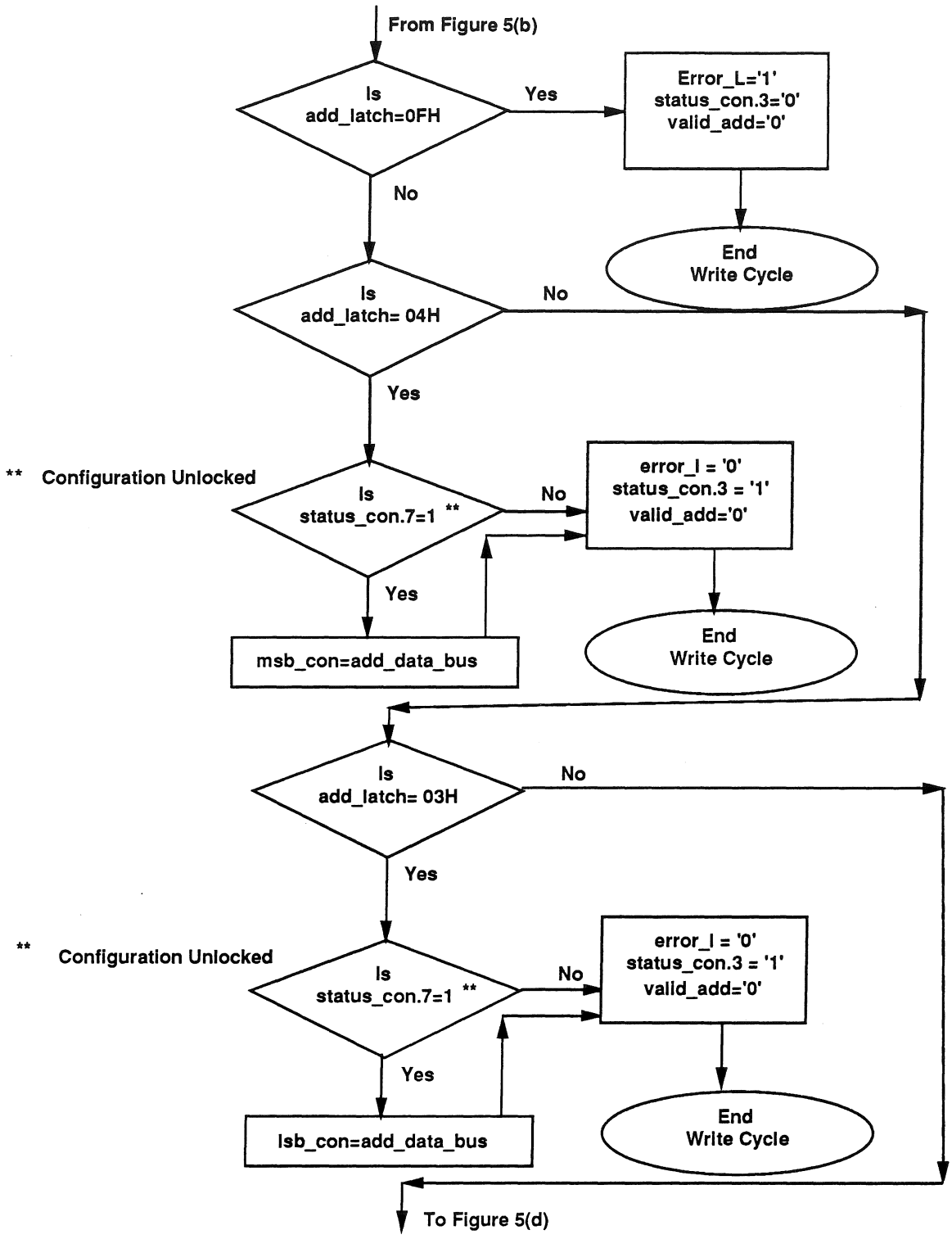


Figure 5(c). Secondary State Diagram of WRITE CYCLE (contd.)

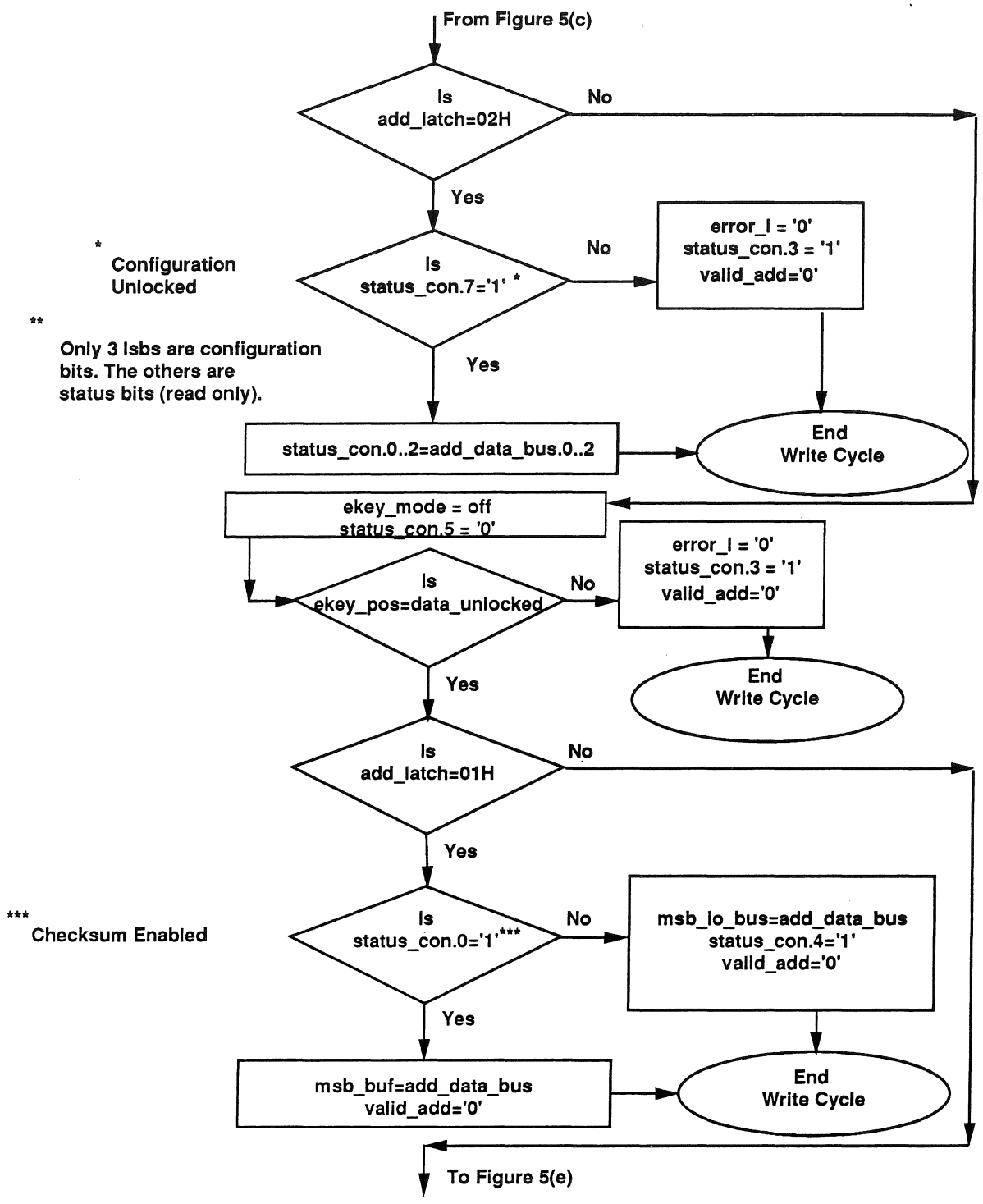


Figure 5(d). Secondary State Diagram of WRITE CYCLE (contd.)

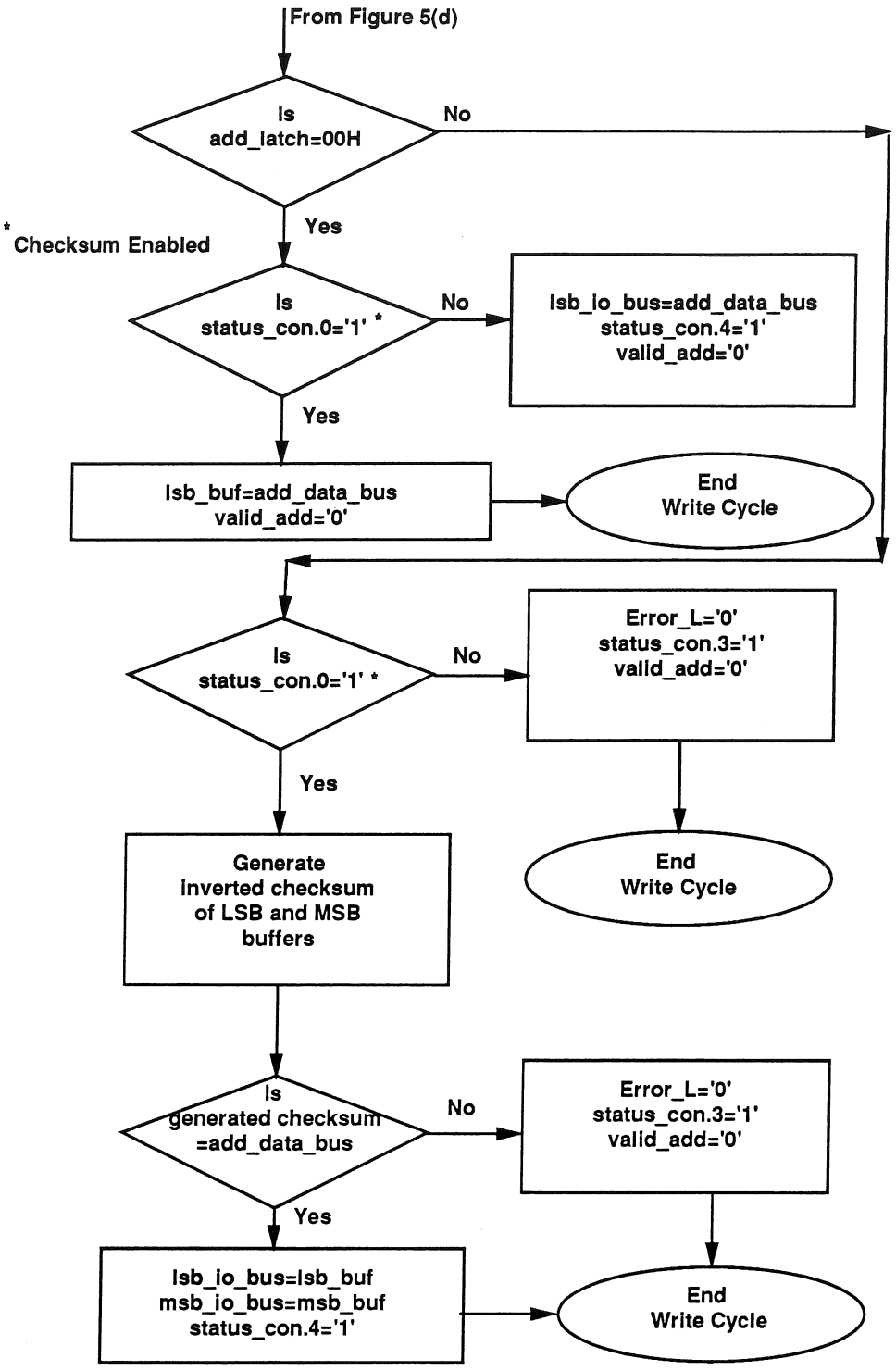


Figure 5(e). Secondary State Diagram of WRITE CYCLE (contd.)

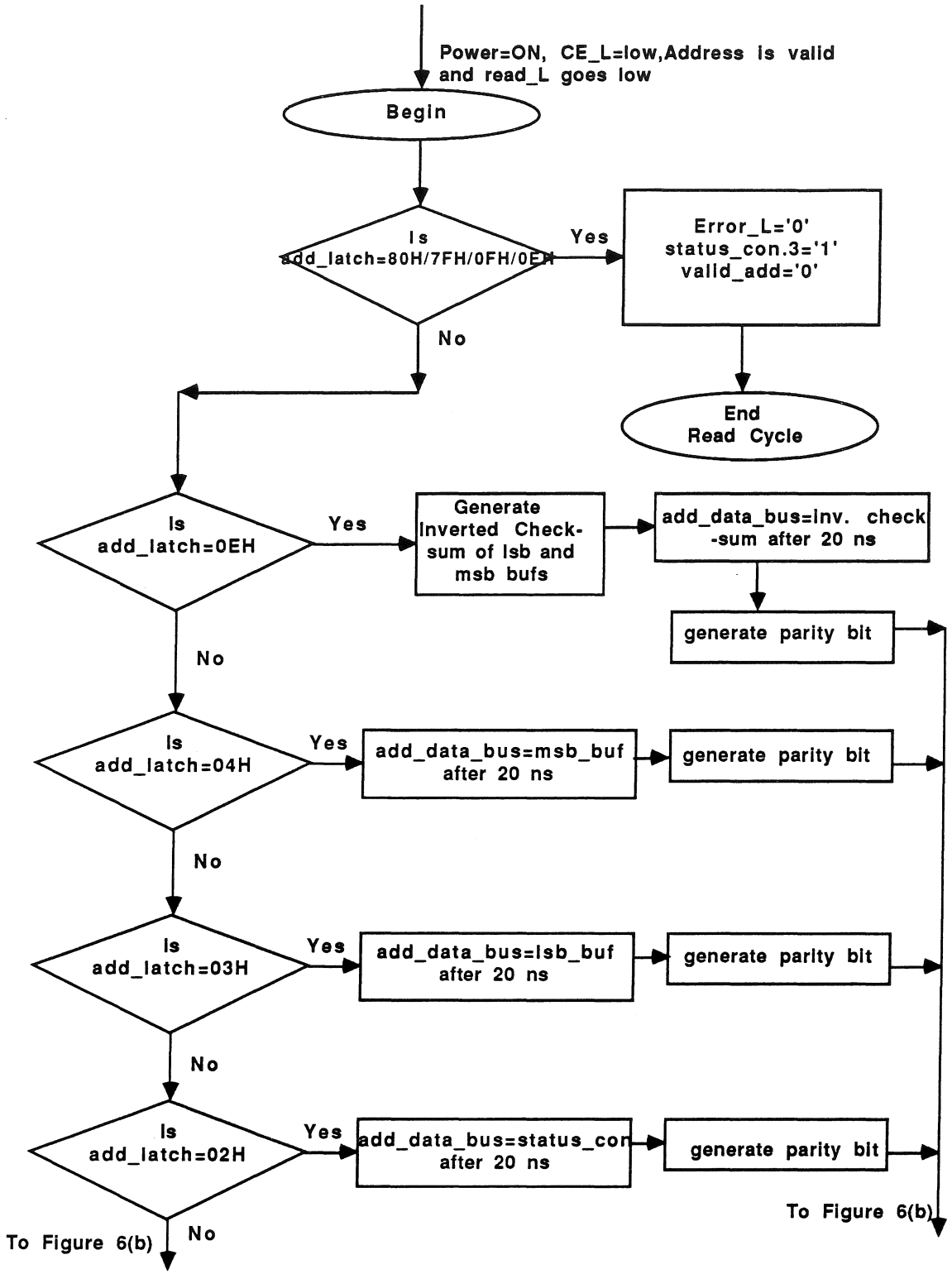
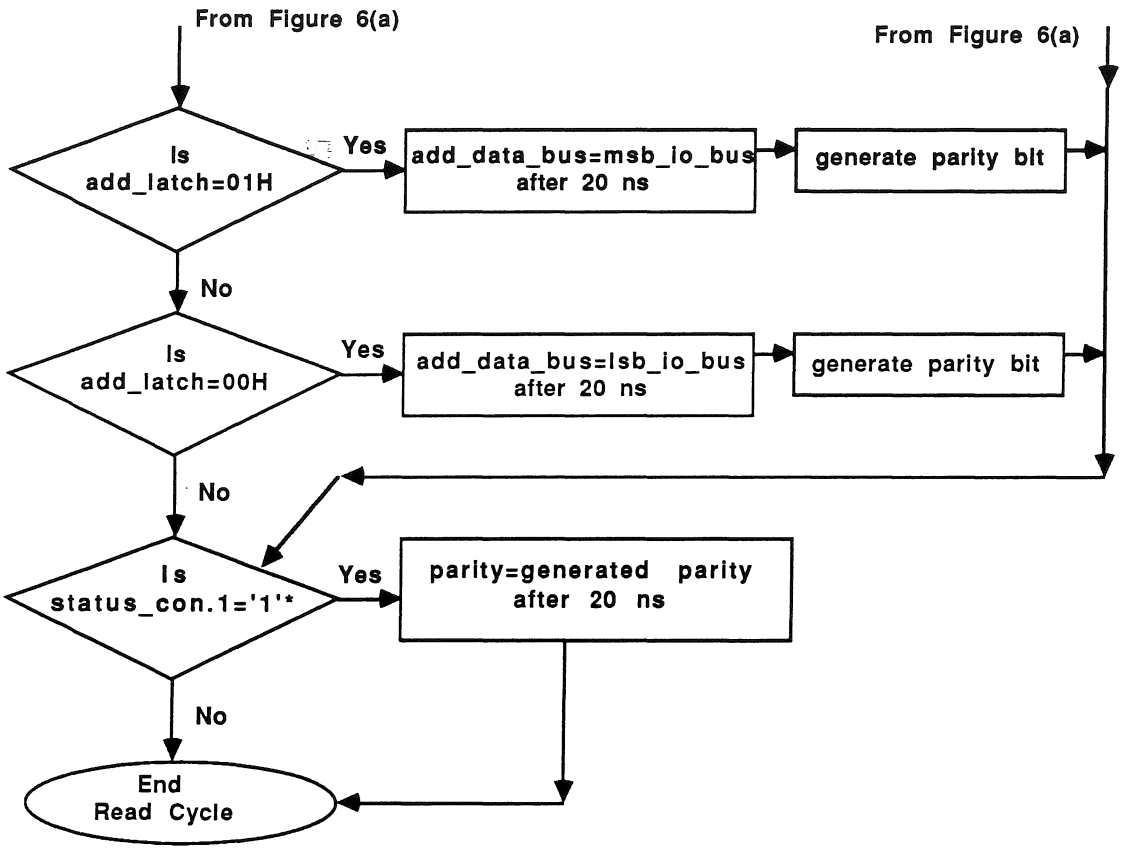


Figure 6(a). Secondary State Diagram of READ CYCLE



*
Data Parity
On

Figure 6(b). Secondary State Diagram of READ CYCLE (contd.)

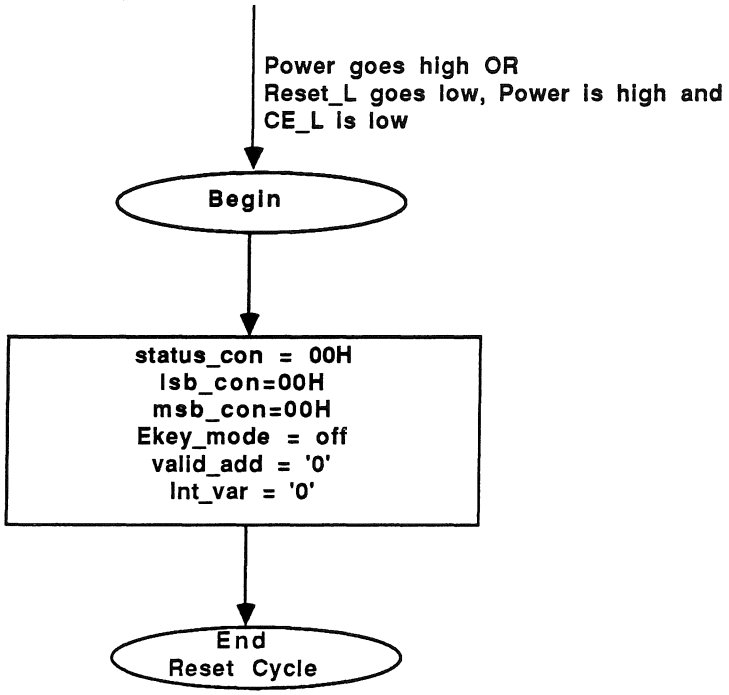


Figure 7. Secondary State Diagram of RESET CYCLE

4. VHDL DESCRIPTION OF DRACO

The behavior of the DRACO chip was described in VHDL using the flowcharts and pseudo-code as the preliminary design specification. The behavioral VHDL description used both block and process statements.

Each of DRACO's eight primary states was modeled using a VHDL block, in which a resolved signal of type "state" (described later in this section) is assigned the appropriate state value. The guard at entry to each block specifies the conditions under which a state is to be entered, while the body of each VHDL block includes a guarded signal assignment to the resolved signal. The Read, Write, Address and Reset Cycles were described using a process each for the actions performed in those states.

4.1. Treatment of Timing Constraints

The timing specifications in the DRACO data sheet represent the physical characteristics of the completed chip design. The data sheet did not have any behavioral timing specifications in it. As a result, the data sheet's timing specifications will be used as the timing constraints which must be met by the final design. We will look into how these timing specifications get transformed into timing constraints for internal structures (register, adder, etc.) of DRACO in forthcoming reports which will describe synthesis of the DRACO chip.

Assertions were added to the VHDL description to validate the correct sequence of critical signals received from the host. Error messages are reported when incorrect sequences are encountered. For example, an attempt to write to DRACO must be preceded by an address latch. The description, however, does not care about the minimum and

maximum timing constraints, but only checks and ensures sequentiality. Therefore, for error-free operation, the host only needs to make sure that it writes data into DRACO after latching a valid address; there is no minimum (or maximum) time constraint between these two events (latching of valid address and writing into the chip).

Moreover, the host should ensure that no two out of the four signals: *read_l*, *write_l*, *reset_l and ale* become active simultaneously (the behavior of the chip cannot be predicted if any pair of these signals is active simultaneously). However, power failures may occur at any time. If the power signal becomes active simultaneously with any other signal, the power signal is selected with priority while the other signal is ignored. Prioritized treatment of the power-off signal is built into the resolution function of the signal representing the chip's state.

From DRACO's state diagram (Figure 3), we can construct the following list which describes the eight states and the conditions under which they are entered:

(1) RESET STATE: POWER goes high or RESET_L signal falls low and POWER is high and chip is enabled.

(2) CHIP ENABLE: CE_L falls low and POWER is on.

(3) ADDRESS CYCLE: POWER is high, CE_L is low and ALE goes low.

(4) READ CYCLE: POWER is high, CE_L is low, VALID ADDRESS is true and READ_L goes low.

(5) WRITE_CYCLE: POWER is high, CE_L is low, VALID ADDRESS is true and WRITE_L goes low.

(6) IDLE: ALE goes high, power is on and chip is enabled.

(7) CHIP DISABLED: CE_L goes high and chip is enabled.

(8) POWER OFF: POWER goes low.

4.2. Type Declarations

A user defined type *state* has been defined which can assume the following values:

$$state = \{ reset, chip\_enabled, chip\_disabled, write,$$
$$read, address, idle, power\_off\}$$

The DRACO chip sequences through these states as indicated in Figure 3.

4.3. Resolution Functions

Since the VHDL description has several blocks that make assignments to this signal, a resolution function is declared to resolve the final value assigned to the state signal. This resolution function will give priority to the power_off state as discussed previously.

For status/configuration registers which have more than one source (for example, status bits of the status/configuration register are updated in all the 4 cycles: read, write, address and reset), we need to define resolution functions. Also, in some cases we need to address individual bits of these registers, both separately and in different processes. Thus each of the configuration registers is declared as an array of resolved bit type. Specifically, the exact VHDL statements are:

function bit_res_fun (Input bit_res) return bit;

subtype bitres is bit_res_fun bit;

status_con_reg: *array* (7 *downto* 0) *of bitres Register*;

The first statement declares a bit resolution function that returns a signal of type *bit*. The second statement declares a resolved signal subtype called *bitres* and the last statement declares a register (width 8) of type *bitres*.

4.4. Stimulus to the VHDL Description

In the current VHDL description, input stimuli to the chip are generated without the use of a stimulus/command file. Instead, the ports which carry signals to DRACO from the host are commented out from the entity declaration and declared as signals in the architecture body. These signals are then assigned waveforms in a process body within the architecture. The process "generate_signals" in the VHDL description performs the function of generating input stimuli for exercising the VHDL model.

4.5. Simulator Specifics: Vantage and Zycad

Simulations of DRACO's VHDL behavior were attempted on the Vantage [Vant89] and Zycad [Zyca89] simulators.

The Zycad simulator supported the resolved types described above. However, the Vantage simulator (version 1.203) could not simulate the description, since that version did not support bit-slicing. A description to be simulated on Vantage would therefore require each bit of the register status_con_reg to be separated as shown below:

status_con_reg: *bitres*; *for* $i = 0$ *to* 7

With such a description, a probe file (command file) has to assign values to a register by addressing each bit individually. This is an onerous task when dealing with even a few registers and buses.

5. EXAMPLES

This section describes five typical operational scenarios for the DRACO chip. Included with each scenario (labeled "Example") are the stimulus files and the simulation results generated by the Zycad simulator.

5.1. Example 1

The first example writes data (FFH) onto the low byte of the I/O ports. In this example the following signals are received by DRACO:

Power on at 25 *fs*;

Chip enabled at 50 *fs*;

Following the receipt of these two signals, the chip is reset and is set to execute further commands such as address latch and reset. In the reset state, the address parity, data parity and checksum are off, the electronic key is off and data and configurations are locked. To write data into the low byte of the I/O ports, the following states have to execute four times:

address cycle -----> *write cycle* ------> *idle*

In the first two cycles, the electronic key is switched on; data is unlocked in the third cycle, while data is written to the I/O ports in the fourth cycle.

Details of each cycle are given below:

cycle1: write data AAH at address 80H followed by

cycle2: write data 55H at address 7FH.

cycle3: write data 55H at address 7FH

cycle4: write data FFH at address 00H

These four cycles execute between 100 fs and 500 fs. Finally the chip is disabled at 1910 fs and the power is turned off at 1920 fs after completion of these four cycles.

The stimulus file for this example is shown in Figure 8.

Tabular results obtained from the simulation are shown in Figure 9.

5.2. Example 2

This example simulates the behavior of DRACO when the parity checks are enabled. DRACO needs to be configured to activate these checks. As with the previous example, the following actions are required after the electronic key is switched on:

1) Unlock the DRACO configuration

2) Set the 1st and 2nd bits of the configuration register to enable parities.

The first step requires writing AAH at address 7FH, while the second step requires writing 06H at address 02H.

STIMULUS FILE FOR EXAMPLE 1

```
-- 40 LINES
GENERATE_SIGNALS:
process
begin
  -- This example unlocks the key, unlocks the data and writes
  -- to the lsb I/O bus.

  -- POWER SIGNAL
  power <= '0',
        '1' after 25 fs,
        '0' after 1920 fs;

  -- CHIP ENABLE SIGNAL
  ce_l <= '1',
       '0' after 50 fs,
       '1' after 1910 fs;

   add_data_bus <= X"80" after 105 fs,
                X"AA" after 140 fs,
                X"7F" after 205 fs,
                X"55" after 240 fs,
                X"7F" after 305 fs,
                X"55" after 340 fs,
                X"00" after 405 fs,
                X"FF" after 440 fs;

   ale <= '1', '0' after 110 fs, '1' after 170 fs,
           '0' after 210 fs, '1' after 270 fs,
           '0' after 310 fs, '1' after 370 fs,
           '0' after 410 fs, '1' after 470 fs;

   write_l <= '1', '0' after 130 fs, '1' after 160 fs,
               '0' after 230 fs, '1' after 260 fs,
               '0' after 330 fs, '1' after 360 fs,
               '0' after 430 fs, '1' after 460 fs;

  read_l  <=  '1';
  wait;

end process GENERATE_SIGNALS;
```

Figure 8. Stimulus File for Example 1.

SIMULATION RESULTS FROM EXAMPLE 1

```
SMON9          CE ACTIVE /DRACO/ERROR_L
SMON8          CE ACTIVE /DRACO/MSB_BUF
SMON7          CE ACTIVE /DRACO/LSB_BUF
SMON6          CE ACTIVE /DRACO/DATA_BUS
SMON5          CE ACTIVE /DRACO/MSB_IO_BUS
SMON4          CE ACTIVE /DRACO/LSB_IO_BUS
SMON3          CE ACTIVE /DRACO/CYCLE
SMON2          CE ACTIVE /DRACO/EKEY_POS
SMON1          CE ACTIVE /DRACO/ADD_DATA_BUS
SMON          CE ACTIVE /DRACO/EKEY_MODE
25 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = RESET)
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = LOCKED)
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = LOCKED)
50 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = CHIP_ENABLED)
105 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"80")
110 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
130 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
140 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"AA")
170 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
205 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
210 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
230 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
240 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"55")
260 FS
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = ONN)
270 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
305 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
310 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
330 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
340 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"55")
360 FS
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = DATA_UNLOCKED)
370 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
405 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"00")
410 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
430 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
440 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"FF")
460 FS
```

```
  SMON4:   ACTIVE /DRACO/LSB_IO_BUS (value = X"FF")
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
470 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
1910 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = CHIP_DISABLE)
1920 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = POWER_OFF)
2000 FS
```

Figure 9. Tabular Results For Example 1.

Except for the last cycle, the parity of data and addresses received by DRACO from the host are correct. However, the address parity bit received in the last cycle is incorrect.

The tabular results obtained as a result of simulating this example are shown Figure 11. Note that a warning message is generated on receipt of incorrect parity (corrupted data). The subsequent write cycle is also aborted since there is no valid address to write the data.

Power is turned on at 25 fs, chip is enabled at 50 fs. In the end the chip is disabled at 1910 fs and power turned off at 1920fs.

The stimulus file for this example is shown in Figure 10.

5.3. Example 3

This example simulates the behavior for a test case which is identical to EXAMPLE 2 except that in the last cycle, the data parity (instead of the address parity) bit is received in error.

The stimulus file for this example is shown in Figure 12.

STIMULUS FILE FOR EXAMPLE 2

```
-- 59 LINES
GENERATE_SIGNALS:
process
begin
  -- THIS PROCESS CONFIGURES DRACO TO ENABLE DATA AND ADDRESS
  -- PARITIES. IT THEREAFTER UNLOCKS DATA AND WRITES FF TO THE
  -- LSB I/O PORT. SINCE ADDRESS PARITY IS FALSE IN THE
  -- LAST WRITE THE OPERATION IS UNSUCCESSFUL.

  -- POWER SIGNAL
  power <= '0',
         '1' after 25 fs,
         '0' after 1920 fs;

  -- CHIP ENABLE SIGNAL
  ce_l <= '1',
        '0' after 50 fs,
        '1' after 1910 fs;

   ale <= '1', '0' after 110 fs, '1' after 170 fs,
            '0' after 210 fs, '1' after 270 fs,
            '0' after 310 fs, '1' after 370 fs,
            '0' after 410 fs, '1' after 470 fs,
            '0' after 510 fs, '1' after 570 fs,
            '0' after 610 fs, '1' after 670 fs;


   add_data_bus <= X"80" after 105 fs,
                X"AA" after 140 fs,
                X"7F" after 205 fs,
                X"55" after 240 fs,
                X"7F" after 305 fs,
                X"AA" after 340 fs,
                X"02" after 405 fs,
                X"06" after 440 fs,
                X"7F" after 505 fs,
                X"55" after 540 fs,
                X"00" after 605 fs,
                X"FF" after 640 fs;


   write_l <= '1', '0' after 130 fs, '1' after 160 fs,
                '0' after 230 fs, '1' after 260 fs,
                '0' after 330 fs, '1' after 360 fs,
                '0' after 430 fs, '1' after 460 fs,
                '0' after 530 fs, '1' after 560 fs,
                '0' after 630 fs, '1' after 660 fs;

   read_l  <=  '1';

   -- PARITY SIGNAL
   parity <= '0' after 505 fs,
          '1' after 540 fs,
-- incorrect address parity correct data parity
          '0' after 605 fs,
          '1' after 640 fs;
  wait;

end process GENERATE_SIGNALS;
```

Figure 10. Stimulus File for Example 2.

SIMULATION RESULTS FROM EXAMPLE 2

```
SMON9          CE ACTIVE /DRACO/ERROR_L
SMON8          CE ACTIVE /DRACO/MSB_BUF
SMON7          CE ACTIVE /DRACO/LSB_BUF
SMON6          CE ACTIVE /DRACO/DATA_BUS
SMON5          CE ACTIVE /DRACO/MSB_IO_BUS
SMON4          CE ACTIVE /DRACO/LSB_IO_BUS
SMON3          CE ACTIVE /DRACO/CYCLE
SMON2          CE ACTIVE /DRACO/EKEY_POS
SMON1          CE ACTIVE /DRACO/ADD_DATA_BUS
SMON          CE ACTIVE /DRACO/EKEY_MODE
25 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = RESET)
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = LOCKED)
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = LOCKED)
50 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = CHIP_ENABLED)
105 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"80")
110 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
130 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
140 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"AA")
170 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
205 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
210 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
230 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
240 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"55")
260 FS
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = ONN)
270 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
305 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
310 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
330 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
340 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"AA")
360 FS
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = CONFIG_UNLOCKED)
370 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
405 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"02")
410 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
430 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
440 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"06")
470 FS
```

```
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
505 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
510 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
530 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
540 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"55")
560 FS
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = DATA_UNLOCKED)
570 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
605 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"00")
610 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
Assertion WARNING in BEHAVIOURAL: "ERROR 12"
  SMON9:   ACTIVE /DRACO/ERROR_L (value = '0')
630 FS
Assertion WARNING in BEHAVIOURAL: "ERROR 8"
640 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"FF")
670 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
1910 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = CHIP_DISABLE)
1920 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = POWER_OFF)
2000 FS
```

Figure 11. Tabular Results For Example 2.

The tabular results for this simulation are shown in Figure 13.

5.4. Example 4

In this example both the address and data parity bits are received correctly in the last cycle.

The stimulus file for this example is shown in Figure 14.

The results are shown in Figure 15.

Note that in this example the data does get written to the I/O bus of the DRACO chip correctly.

STIMULUS FILE FOR EXAMPLE 3

```
-- 54 LINES
GENERATE_SIGNALS:
process
  -- THIS PROCESS CONFIGURES DRACO TO ENABLE DATA AND ADDRESS
  -- PARITIES. IT THEREAFTER UNLOCKS DATA AND WRITES FF TO THE
  -- LSB I/O PORT. SINCE DATA PARITY IS FALSE IN THE
  -- LAST WRITE THE OPERATION IS UNSUCCESSFUL.
begin
  -- THIS PROGRAMS
  -- POWER SIGNAL
  power <= '0',
        '1' after 25 fs,
        '0' after 1920 fs;

  -- CHIP ENABLE SIGNAL
  ce_l <= '1',
       '0' after 50 fs,
       '1' after 1910 fs;

   ale <= '1', '0' after 110 fs, '1' after 170 fs,
            '0' after 210 fs, '1' after 270 fs,
            '0' after 310 fs, '1' after 370 fs,
            '0' after 410 fs, '1' after 470 fs,
            '0' after 510 fs, '1' after 570 fs,
            '0' after 610 fs, '1' after 670 fs;


   add_data_bus <= X"80" after 105 fs,
                X"AA" after 140 fs,
                X"7F" after 205 fs,
                X"55" after 240 fs,
                X"7F" after 305 fs,
                X"AA" after 340 fs,
                X"02" after 405 fs,
                X"06" after 440 fs,
                X"7F" after 505 fs,
                X"55" after 540 fs,
                X"00" after 605 fs,
                X"FF" after 640 fs;


   write_l <= '1', '0' after 130 fs, '1' after 160 fs,
                '0' after 230 fs, '1' after 260 fs,
                '0' after 330 fs, '1' after 360 fs,
                '0' after 430 fs, '1' after 460 fs,
                '0' after 530 fs, '1' after 560 fs,
                '0' after 630 fs, '1' after 660 fs;

   read_l  <=  '1';

   -- PARITY SIGNAL
   parity <= '0' after 505 fs,
          '1' after 540 fs,
-- correct address parity incorrect data parity
          '1' after 605 fs,
          '0' after 640 fs;
  wait;

end process GENERATE_SIGNALS;
```

Figure 12. Stimulus File for Example 3.

SIMULATION RESULTS FROM EXAMPLE 3

```
SMON9          CE ACTIVE /DRACO/ERROR_L
SMON8          CE ACTIVE /DRACO/MSB_BUF
SMON7          CE ACTIVE /DRACO/LSB_BUF
SMON6          CE ACTIVE /DRACO/DATA_BUS
SMON5          CE ACTIVE /DRACO/MSB_IO_BUS
SMON4          CE ACTIVE /DRACO/LSB_IO_BUS
SMON3          CE ACTIVE /DRACO/CYCLE
SMON2          CE ACTIVE /DRACO/EKEY_POS
SMON1          CE ACTIVE /DRACO/ADD_DATA_BUS
SMON          CE ACTIVE /DRACO/EKEY_MODE
25 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = RESET)
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = LOCKED)
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = LOCKED)
50 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = CHIP_ENABLED)
105 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"80")
110 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
130 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
140 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"AA")
170 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
205 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
210 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
230 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
240 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"55")
260 FS
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = ONN)
270 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
305 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
310 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
330 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
340 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"AA")
360 FS
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = CONFIG_UNLOCKED)
370 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
405 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"02")
410 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
430 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
440 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"06")
470 FS
```

```
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
505 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
510 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
530 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
540 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"55")
560 FS
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = DATA_UNLOCKED)
570 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
605 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"00")
610 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
630 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
640 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"FF")
660 FS
Assertion WARNING in BEHAVIORAL: "ERROR 14"
  SMON9:   ACTIVE /DRACO/ERROR_L (value = '0')
670 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
1910 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = CHIP_DISABLE)
1920 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = POWER_OFF)
2000 FS
```

Figure 13. Tabular Results For Example 3.

5.5. Example 5

In this example DRACO is configured so as to enable data and address parities as well as the checksum mode. Power is switched on, the chip is enabled and configured. Thereafter, the data is unlocked and data is written to the I/O ports. The data gets loaded into the buffers since checksum is enabled. A checksum byte is written, subsequently. A checksum of the msb and lsb bytes stored in the buffers is internally generated and compared with the checksum obtained from the host. A generated checksum which tallies with the checksum written by the host causes the data stored in the buffers to be transfferred to the I/O ports. Finally, the checksum of the low and high bytes of the data stored in the buffers is read from DRACO.

STIMULUS FILE FOR EXAMPLE 4

```
-- 58 LINES
GENERATE_SIGNALS:
process
begin
  -- THIS PROCESS CONFIGURES DRACO TO ENABLE DATA AND ADDRESS
  -- PARITIES. IT THEREAFTER UNLOCKS DATA AND WRITES FF TO THE
  -- LSB I/O PORT.

  -- POWER SIGNAL
  power <= '0',
        '1' after 25 fs,
        '0' after 1920 fs;

  -- CHIP ENABLE SIGNAL
  ce_l <= '1',
       '0' after 50 fs,
       '1' after 1910 fs;

   ale <= '1', '0' after 110 fs, '1' after 170 fs,
            '0' after 210 fs, '1' after 270 fs,
            '0' after 310 fs, '1' after 370 fs,
            '0' after 410 fs, '1' after 470 fs,
            '0' after 510 fs, '1' after 570 fs,
            '0' after 610 fs, '1' after 670 fs;


   add_data_bus <= X"80" after 105 fs,
                X"AA" after 140 fs,
                X"7F" after 205 fs,
                X"55" after 240 fs,
                X"7F" after 305 fs,
                X"AA" after 340 fs,
                X"02" after 405 fs,
                X"06" after 440 fs,
                X"7F" after 505 fs,
                X"55" after 540 fs,
                X"00" after 605 fs,
                X"FF" after 640 fs;


   write_l <= '1', '0' after 130 fs, '1' after 160 fs,
                '0' after 230 fs, '1' after 260 fs,
                '0' after 330 fs, '1' after 360 fs,
                '0' after 430 fs, '1' after 460 fs,
                '0' after 530 fs, '1' after 560 fs,
                '0' after 630 fs, '1' after 660 fs;

   read_l  <=  '1';

   -- PARITY SIGNAL
   parity <= '0' after 505 fs,
          '1' after 540 fs,
-- correct address parity correct data parity
          '1' after 605 fs,
          '1' after 640 fs;
  wait;

end process GENERATE_SIGNALS;
```

Figure 14. Stimulus File for Example 4.

SIMULATION RESULTS FROM EXAMPLE 4

```
SMON9          CE ACTIVE /DRACO/ERROR_L
SMON8          CE ACTIVE /DRACO/MSB_BUF
SMON7          CE ACTIVE /DRACO/LSB_BUF
SMON6          CE ACTIVE /DRACO/DATA_BUS
SMON5          CE ACTIVE /DRACO/MSB_IO_BUS
SMON4          CE ACTIVE /DRACO/LSB_IO_BUS
SMON3          CE ACTIVE /DRACO/CYCLE
SMON2          CE ACTIVE /DRACO/EKEY_POS
SMON1          CE ACTIVE /DRACO/ADD_DATA_BUS
SMON          CE ACTIVE /DRACO/EKEY_MODE
25 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = RESET)
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = LOCKED)
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = LOCKED)
50 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = CHIP_ENABLED)
105 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"80")
110 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
130 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
140 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"AA")
170 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
205 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
210 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
230 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
240 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"55")
260 FS
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = ONN)
270 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
305 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
310 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
330 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
340 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"AA")
360 FS
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = CONFIG_UNLOCKED)
370 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
405 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"02")
410 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
430 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
440 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"06")
470 FS
```

```
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
505 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
510 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
530 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
540 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"55")
560 FS
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = DATA_UNLOCKED)
570 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
605 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"00")
610 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
630 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
640 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"FF")
660 FS
  SMON4:   ACTIVE /DRACO/LSB_IO_BUS (value = X"FF")
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
670 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
1910 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = CHIP_DISABLE)
1920 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = POWER_OFF)
2000 FS
```

Figure 15. Tabular Results For Example 4.

The stimulus file for this example is shown in Figure 16.

The tabular results obtained as a result of simulation are shown in Figure 17.

SIMULATION RESULTS FROM EXAMPLE 5

```
SMON9          CE ACTIVE /DRACO/ERROR_L
SMON8          CE ACTIVE /DRACO/MSB_BUF
SMON7          CE ACTIVE /DRACO/LSB_BUF
SMON6          CE ACTIVE /DRACO/DATA_BUS
SMON5          CE ACTIVE /DRACO/MSB_IO_BUS
SMON4          CE ACTIVE /DRACO/LSB_IO_BUS
SMON3          CE ACTIVE /DRACO/CYCLE
SMON2          CE ACTIVE /DRACO/EKEY_POS
SMON1          CE ACTIVE /DRACO/ADD_DATA_BUS
SMON           CE ACTIVE /DRACO/EKEY_MODE
25 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = RESET)
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = LOCKED)
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = LOCKED)
```

STIMULUS FILE FOR EXAMPLE 5

```
-- 85 LINES
GENERATE_SIGNALS:
process
begin
  -- This example configures DRACO to enable data parity
  -- address parity and checksum. The key is in unlock config
  -- positions while configuring. Thereafter the data is
  -- unlocked and data is written to the lsb and msb I/O
  -- ports. Subsequently, checksum is written to and read from
  -- DRACO. A successful write of checksum writes data onto
  -- the I/O ports.

  -- POWER SIGNAL
  power <= '0',
         '1' after 25 fs,
         '0' after 1920 fs;

  -- CHIP ENABLE SIGNAL
  ce_l <= '1',
        '0' after 50 fs,
        '1' after 1910 fs;


  ale <= '1', '0' after 110 fs, '1' after 170 fs,
           '0' after 210 fs, '1' after 270 fs,
           '0' after 310 fs, '1' after 370 fs,
           '0' after 410 fs, '1' after 470 fs,
           '0' after 510 fs, '1' after 570 fs,
           '0' after 610 fs, '1' after 670 fs,
           '0' after 710 fs, '1' after 770 fs,
           '0' after 810 fs, '1' after 870 fs,
           '0' after 910 fs, '1' after 970 fs;



  add_data_bus <= X"80" after 105 fs,
                X"AA" after 140 fs,
                X"7F" after 205 fs,
                X"55" after 240 fs,
                X"7F" after 305 fs,
                X"AA" after 340 fs,
                X"02" after 405 fs,
                X"07" after 440 fs,
                -- unlock data
                X"7F" after 505 fs,
                X"55" after 540 fs,
                -- write into the lsb buf
                X"00" after 605 fs,
                X"08" after 640 fs,
                -- write into the msb_buf
                X"01" after 705 fs,
                X"04" after 740 fs,
                -- write inverted checksum
                X"0E" after 805 fs,
                X"F3" after 840 fs,
                      -- read checksum
                X"0E" after 905 fs;


  write_l <= '1', '0' after 130 fs, '1' after 160 fs,
                '0' after 230 fs, '1' after 260 fs,
```

```
                '0' after 330 fs, '1' after 360 fs,
                '0' after 430 fs, '1' after 460 fs,
                '0' after 530 fs, '1' after 560 fs,
                '0' after 630 fs, '1' after 660 fs,
                        '0' after 730 fs, '1' after 760 fs,
                        '0' after 830 fs, '1' after 860 fs;

    read_l <= '1',
              '0' after 930 fs;

    -- PARITY SIGNAL
    parity <= '0' after 505 fs,
                    '1' after 540 fs,
                    '1' after 605 fs,
                    '0' after 640 fs,
                    '0' after 705 fs,
                    '0' after 740 fs,
                    '0' after 805 fs,
                    '1' after 840 fs,
                    '0' after 905 fs;

    wait;
end process GENERATE_SIGNALS;
```

Figure 16. Stimulus File for Example 5.

```
50 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = CHIP_ENABLED)
105 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"80")
110 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
130 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
140 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"AA")
170 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
205 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
210 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
230 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
240 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"55")
260 FS
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = ONN)
270 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
305 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
310 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
330 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
340 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"AA")
360 FS
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = CONFIG_UNLOCKED)
370 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
405 FS
```

```
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"02")
410 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
430 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
440 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"07")
470 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
505 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"7F")
510 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
530 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
540 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"55")
560 FS
  SMON2:   ACTIVE /DRACO/EKEY_POS (value = DATA_UNLOCKED)
570 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
605 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"00")
610 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
630 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
640 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"08")
660 FS
  SMON7:   ACTIVE /DRACO/LSB_BUF (value = X"08")
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
670 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
705 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"01")
710 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
730 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
740 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"04")
760 FS
  SMON8:   ACTIVE /DRACO/MSB_BUF (value = X"04")
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
770 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
805 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"0E")
810 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
830 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = WRITE)
840 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"F3")
860 FS
  SMON5:   ACTIVE /DRACO/MSB_IO_BUS (value = X"04")
  SMON4:   ACTIVE /DRACO/LSB_IO_BUS (value = X"08")
   SMON:   ACTIVE /DRACO/EKEY_MODE (value = OFF)
870 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
905 FS
  SMON1:   ACTIVE /DRACO/ADD_DATA_BUS (value = X"0E")
```

```
910 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = ADDRESS)
930 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = READ)
950 FS
  SMON6:   ACTIVE /DRACO/DATA_BUS (value = X"F3")
970 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = IDLE)
1910 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = CHIP_DISABLE)
1920 FS
  SMON3:   ACTIVE /DRACO/CYCLE (value = POWER_OFF)
2000 FS
```

Figure 17. Tabular Results For Example 5.

6. Acknowledgements

Bob Larsen provided useful input and comments on an earlier draft of this paper. Sanjiv Narayan and Frank Vahid helped to refine an initial behavioral model of DRACO, and also suggested some stylistic improvements to the VHDL code used to model the DRACO chip. Dan Gajski and Bob Larsen acted as catalysts in this industry-university effort. The authors would like to thank all of these people.

7. Summary

This report described the behavioral model of a commercial chip design named DRACO from Rockwell International, which was initially documented with only a data sheet and associated logic schematics. The behavioral model was developed using flowcharts and pseudo-code. A set of five typical operational test cases was also developed. Subsequently, these flowcharts and test scenarios were described using behavioral VHDL and associated stimulus files. The VHDL code was tested on two commercial simulators (Vantage and Zycad) to verify the correctness of the behavior with respect to the operational test cases. The behavioral flowcharts, VHDL behavioral code, the stimulus files and

results of the simulation runs are all included in this report.

Future work will attempt to use this DRACO behavioral description as input to a suite of behavioral, logic and layout synthesis tools at U.C. Irvine.

8. References

[IEEE90] The IEEE Institute, "Save U.S. semiconductor industry now or lose technical edge, Bush told," Volume 14, Number 1, January 1990.

[Lars90] Robert P. Larsen, Rockwell International, *private communication*, April 1990.

[Pase90] Dave Pasela, Rockwell International, *private communication*, May 1990.

[Sumn89] Larry W. Sumney, "Workstations, Semiconductors, and Competitiveness," *Keynote Address at the First IEEE Workstations Symposium*,, Baltimore, MD, Oct. 1989.

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

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

[Zyca89] Zycad Corporation, Menlo Park, CA 1989.

APPENDIX A.

Rockwell DRACO Data Sheet

PREPARED BY	Rockwell International	NUMBER
Johnny Sitou		11495
APPROVALS		TYPE
Robert W. Polkinghorn	ROCKWELL INTERNATIONAL CORPORATION SEMICONDUCTOR PRODUCTS DIVISION	P01 Specification
	FSCM NO. 34576	DATE
		November 20, 1989
	PROPRIETARY INFORMATION OF SEMICONDUCTOR PRODUCTS DIVISION NO DISSEMINATION OR USE ALLOWED WITHOUT PRIOR WRITTEN PERMISSION	REV. LTR NC / PAGE 1 of 32
		TOTAL PAGES 32

TITLE

ENGINEERING REPORT

PRODUCT: Discrete I/O Backplane ASIC (DRACO)

PART NUMBER 11495

SYNOPSIS: This document describes the custom integrated circuit that interfaces 1781 single and quad discrete I/O modules to Allen Bradley 1781 communication adapters.

1. Overview
This document defines the operation, performance characteristics and quality assurance requirements for the 1781 discrete I/O backplane custom integrated circuit (941425-61). The ASIC's codename is DRACO and will be referenced as so throughout this document.

In a general sense, DRACO is a general purpose peripheral interface device. The function of DRACO is to interface 16 I/O ports to a microprocessor's 8-bit multiplexed address/data bus and control signals. DRACO has several optional features that assists data integrity in the presence of EMI. DRACO contains a hardware key that must be unlocked by the user prior to operation which adds a level of security to its applications. The versatility of the interface will allow future 1781 adapters to interface to DRACO.

In particular, DRACO will interface de facto standard single and quad point discrete I/O modules (1781-xx5S and 1781-xx5Q) to 1781-Jxx remote discrete I/O adapters. Initially DRACO will be used with the 1781-JAD remote adapter, however, due to the generic bus interface, DRACO will accommodate future 1781 discrete I/O adapters.

Features of DRACO:

* 16 bidirectional I/O ports with read/write byte integrity (16mA sink capacity)
* A checksum generator/comparator for output data.
* Odd parity generator/comparator for read and write addresses.
* Odd parity generator/comparator for write data.
* Odd parity generator for read data.
* Individual selection of parity error checking on address only or data only or both. Facilitates data parity generation in firmware.
* Checksum or Immediate modes for writing to output modules.
* Latched error output to indicate invalid address or parity or checksum error (reset by user).
* I/O direction register to set I/O ports as bidirectional or input-only.
* Electronic key for security and to prevent inadvertent writes to configuration and direction registers.
* Status register to read electronic key, error, and write_acknowledge status.
* Standard 8051 8-bit bus interface with optional parity.

2. Applications

2.1 1781 Backplanes
DRACO will be installed on all Allen Bradley 1781-ADxx discrete I/O backplanes. 1781 discrete I/O modules will coexist with the ASIC on the backplanes.

DRACO will be installed on the following 1781 Discrete I/O backplanes:

1781-AD4	4 single point backplane	1 ASIC per backplane
1781-AD8	8 single point backplane	1 ASIC per backplane
1781-AD16	16 single point backplane	1 ASIC per backplane
1781-AD4Q	4 quad point backplane(16pt)	1 ASIC per backplane
1781-AD8Q	8 quad point backplane(32pt)	2 ASICs per backplane

For additional information on these backplanes, refer to DS#PC 4235

2.2 1781 Adapter Interface
Initially DRACO will be used in conjunction with the 1781-JAD Discrete I/O adapter. The adapter to DRACO interface is a buffered 80C51 address/data 8-bit bus plus parity and may be connected remotely through an 18-inch long ribbon cable. Provisions will be made within DRACO for the propagation delays due to the buffer circuitry and ribbon cable.

2.3 I/O Modules
All Allen Bradley 1781-xx5S and 1781-xx5Q I/O modules will operate with DRACO.

Other manufacturers' single/quad modules are electrically compatible to operate with DRACO, however, the form factor of competitor's I/O modules will not allow non-1781 modules to reside in 1781-ADxx backplanes

3. Functional Overview
DRACO provides an interface from an 8-bit multiplexed address/data bus with accompanying parity and control signals to sixteen I/O ports. Each of the 16 I/O ports can be connected to an Allen Bradley 1781 input or output module (or equivalent). DRACO will accommodate a maximum of 16 input OR 16 output points OR a combination thereof.

DRACO will operate as an I/O mapped device to the user's microcontroller. All data communications between DRACO and the microcontroller is performed with read and write instructions to embedded address locations within DRACO. DRACO provides an interrupt signal (ERROR_L) to the host microcontroller to indicate an unsuccessful read or write instruction. The use of this interrupt pin is optional.

DRACO has two options for writing data to output modules; immediate and checksum. The immediate option only requires writing output data to either the high-byte or low-byte I/O address. Each output data byte immediately transfers to the I/O ports. The checksum option requires writing a checksum byte in addition to the output data before the I/O ports are updated. The data bytes are first buffered and validated with a checksum before updating the I/O ports. This option is controlled by the checksum enable bit in the configuration register.

Due to the bidirectional I/O module interface, the user has the ability to write to all and read from all of DRACO's 16 I/O ports. To prevent inadvertent writes to an input module, each port may optionally be configured as an input-only port. An I/O direction register can be loaded by the user to configure an I/O port as bidirectional or read-only.

3.1 EMI Precautions

To ensure data integrity over the interconnecting cable to and from DRACO in the presence of EMI, two forms of error checking are available; parity and checksum. Odd parity error checking can be performed on the address and/or data bytes to and from DRACO and checksum error checking can be performed only on the received output data from the host microcontroller. These error checking options are user selectable and are enabled by loading a configuration register.

A situation can exist where the address received by DRACO may be altered due to EMI and is transformed into another of DRACO's valid address without generating a parity error. This situation is most damaging when the configuration or direction register is altered inadvertently. To prevent this situation, an electronic key must be unlocked prior to any writes to the configuration or direction register. Once configured, the electronic key will lockout the configuration locations which will prevent these registers from being altered in the presence of EMI.

In addition to odd parity, checksum, and an electronic key, DRACO's embedded addresses were selected to eliminate the adverse effect of an undetected 2-bit error on the address and thus provide an additional level of immunity to EMI.

3.2 User Configuration

Four configuration options are user selectable within DRACO. All configuration options can only be altered if the electronic key is in the "unlock configuration" position. (See section 4.1.2) The options are as follows:

1) ADDRESS PARITY ENABLE (Default: Disabled)
With the address parity option enabled, all addresses received by DRACO must include valid parity in order to execute the cycle. Once enabled, receiving an address that generates invalid parity within DRACO will latch the ERROR_L pin low and inhibit the instruction.

2) DATA PARITY ENABLE (Default: Disabled)
With the data parity option enabled, data for any write cycle to DRACO must generate valid parity within DRACO in order to execute the write cycle. Once enabled, receiving data that generates invalid parity within DRACO will latch the ERROR_L pin low and inhibit the instruction.

3) CHECKSUM ENABLE for output data (Default: Disabled)
With the checksum option enabled, output data transfers to DRACO must include a checksum byte that equals the inverted sum of the OUTPUT[15:8] and OUTPUT[7:0] data. The three byte data transfer (output byte 1, output byte 2, checksum) adds an additional level of data integrity to the output data that is received by DRACO. When the user-supplied checksum equals the DRACO generated checksum, DRACO will transfer the output data to the output modules. Receiving or generating an invalid checksum at DRACO will latch the ERROR_L pin low and inhibit the outputs from being updated.

4) I/O PORTS; bidirectional or input only (Default: Bidirectional)
DRACO's 16 I/O ports can be individually configured as bidirectional or read-only. A user may desire to configure a port to be read-only to prevent an inadvertent write and consequently latching the state of an input module.

4. Functional Blocks

Refer to Figure 1 at the end of this document for a block diagram of DRACO.

DRACO is partitioned into three functional blocks: address decoding, checksum/parity/error, and I/O interface.

4.1 Address Decoding Block

This block includes latching the address byte and its associated parity bit, generating and comparing parity on the address, decoding the address to generate control signals, and implementing the electronic key.

The address byte and its associated parity bit is latched from the multiplexed addr/data bus with Address Latch Enable signal (ALE). Control signals are generated by decoding the address locations along with chip enable, read, and write signals, parity status, and the electronic key positions. The control signals generated in this block are used to read from or write to internal locations.

All address decoded control signals are logically ANDED with CHIP_ENABLE_L and READ_L or WRITE_L and thus gated by either the READ_L or WRITE_L signals. If the address parity is enabled and a parity error develops on the address of a read or write instruction, the address decoder is disabled and the instruction is inhibited to prevent erroneous operation. At power-up the address latch will be set to a default of FFH, an invalid address.

4.1.1 Invalid Address / Address Parity Error Indication
Attempting to read from an invalid address will provide two indications; the ERROR_L pin will be latched low and an invalid parity bit will be provided on the PARITY pin while the incorrect data is being read from the data bus. A parity error on the received address for a read instruction will provide the same results as an invalid address only if address parity checking is enabled.

NOTE: All that is required for DRACO to output data on its AD7-0 bus is a true READ_L AND CE_L signal. When reading an invalid address or a valid address with a parity error, DRACO will output its internal data bus. The value of the data is indeterminate and should not be assumed to be FFH.

Attempting to write to an invalid address will latch the ERROR_L pin low. A parity error on the received address for a write instruction will provide the same results as an invalid address only if address parity checking is enabled. In addition to writing to an invalid address or receiving an address with a parity error, writing to a "locked" valid address will latch the ERROR_L pin low and deem that operation as invalid.

4.1.2 Electronic Key
The function of the electronic key is twofold; to prevent unauthorized applications of DRACO and to ensure the integrity of the configuration and direction registers in the presence of EMI. The key may be in the on or off MODE and in a locked or unlocked POSITION. The position of the key affects address decoding within DRACO and the mode of the key allows the key's position to be changed. i.e. A position may be unlocked only if the key is first ON.

Following a reset of DRACO, the electronic key is off and in the locked position. DRACO functions as a read-only device in the locked position. The key must be unlocked to write any data to DRACO. There are two unlocked positions; unlock data and unlock configuration. These two unlocked positions are complements of one another; both cannot be unlocked simultaneously. The internal locations affected by the key in the two unlocked positions are summarized below.

CONFIGURATION LOCATIONS	DATA LOCATIONS
Configuration register (02H)	Low byte output data (00H)
Low byte I/O direction register (03H)	High byte output data (01H)
High byte I/O direction register (04H)	Inverted checksum (0EH)

The sequence of "key" events that a user would implement following a reset of DRACO are summarized below:

1) Turn the key to the ON position
2) Unlock Configuration locations and configure DRACO for optional address and/or data parity, checksum enable, and I/O direction selections. (Data locations within DRACO are locked at this time)
3) Unlock Data locations (Configuration locations become locked)
4) Turn key OFF

DRACO is now configured with the configuration locations locked and data locations unlocked to commence normal operations.

4.1.2.1 On/Off - Unlock/Lock Encoding
The key is activated by writing proper address/data combinations to locations within DRACO

To Turn the key ON : (proper sequence is required)
1) Write data value AAH to address 80H
2) Write data value 55H to address 7FH

Once ON, either of the two unlock positions may be chosen.
To unlock Configuration Locations (Locks Data Locations):
1) Write data value AAH to address 7FH

To unlock Data Locations (Locks Configuration Locations):
1) Write data value 55H to address 7FH

To Turn the key OFF:
- Write any data value not equal to AAH to address 80H OR
- Write to either High Byte, Low Byte, or Checksum address locations. (This features assures that the electronic key is periodically set to the off mode during normal operation) OR
- Invoke DRACO's reset pin.

4.2 Checksum / Parity / Error Block
This block consists of the following: an 8-bit full adder to generate an inverted checksum, a checksum comparator to compare internally generated checksum with the received checksum, an odd parity generator and comparator for incoming data, an odd parity generator for outgoing data, an error register to indicate address parity, data parity, or checksum errors, and a configuration register to select parity and checksum error-checking options.

4.2.1 Configuration Register / Status Register
A configuration / status register is included in this block to write and read the error-checking enabling options, and to read-only the error, write acknowledge, and electronic key status. Locations that can be writtten to are referred to as CONFIG.x and readable locations as STATUS.x. Following a reset of DRACO, all bits of this register is zero.

The three configuration locations can be summarized as follows:

CONFIG.2 Selects odd parity error checking on all addresses written to DRACO for read or write instructions.

CONFIG.1 Selects odd parity error checking on data that is written to DRACO.

CONFIG.0 Selects inverted checksum error checking on output data.

The status locations can be summarized as follows:

STATUS.7 Status of electronic key's configuration locations.

STATUS.6 Status of electronic key's data locations.

STATUS.5 Status of electronic key's mode.

STATUS.4 Status of a successful write cycle to output modules. This bit is set whenever output data is written to the I/O ports. This bit is cleared at the beginning of any write instruction to DRACO.

STATUS.3 Error status on address, data and checksum. See Section 4.2.4 for a detailed explanation. This bit must be cleared by the user or by a reset cycle.

STATUS.2 Address parity enable status.

STATUS.1 Data parity enable status

STATUS.0 Checksum enable status

The bit designation for the configuration/status register is as follows:

Bit #	7	6	5	4
	Ekey Config 1=Unl,0=Lock Read-only	Ekey Data 1=Unl,0=Lock Read-only	Key Enable 1=on,0=off Read-only	Write Ack 1=Ack 0=Nak Read-only

Bit #	3	2	1	0
	Error Status 1=error Read-only	Adr Par En 1=enable Read/Write	Data Par En 1=enable Read/Write	Checksum En 1=enable Read/Write

4.2.2 Checksum Generator / Comparator

When the user is writing output data to the I/O ports and the checksum enable bit is set (CONFIG.0=1), the checksum generator/comparator provides additional integrity to the output data written to the output modules. Data integrity is enhanced by requiring the user to write a checksum byte in addition to the two output data bytes before output modules are updated.

DRACO's checksum generator is a full adder and provides an inverted sum of the two bytes of output data that are stored in buffers. This inverted sum of the two output bytes is readable independent of the checksum enable bit. The checksum comparator compares the internally generated inverted sum to a checksum that is written to DRACO by the host microcontroller. If the checksums are equal, an internal control signal will transfer the buffered output data to the output ports. If the checksums do not compare, the ERROR_L line is latched low. The checksum comparator is only functional when the checksum enable bit is set in the configuration register.

4.2.3 Data Parity Generator / Comparator
An odd parity bit is generated on all data bytes received during a write cycle. If data parity error checking option is enabled (CONFIG.1=1), the DRACO generated parity bit will be compared with the user-supplied parity bit received on the PARITY pin. If a parity error is detected on any received data, the associated write instruction will be disabled and the ERROR_L signal will be latched low.

During all read cycles, DRACO's PARITY pin will provide an odd parity on all read data regardless of the state of CONFIG.1 bit. DRACO will also generate an invalid (inverted) parity bit, regardless of the state of CONFIG.1 bit, when an attempt is made to read from an invalid address within DRACO.

4.2.4 Error Register
The output pin ERROR_L will be latched low and bit 3 of the status register (STATUS.3) will be set if any of the following errors occur:

- Parity error on any read address (CONFIG.2=1).
- Parity error on any write address (CONFIG.2=1).
- Parity error on any write data (CONFIG.1=1).
- Write or read from an invalid address (CONFIG.x=X).
- Write to an address that is locked by the electronic key.
- Invalid checksum write when checksum mode is enabled.
- Write to the checksum address (0EH) when checksum mode is disabled.

With any one of the above errors, the error producing instruction will not be executed. In order to clear the latched ERROR_L signal, the user must write to DRACO's address location 0FH (Error Reset). The value of the data for this write is irrelevant.

NOTE: The next instruction to DRACO following the error producing instruction will be executed without clearing the previous error. The ERROR_L signal only provides an external error indication for the user. Consecutive instructions to DRACO are not dependent of the state of ERROR_L.

4.3 I/O Block

The I/O interface consists of 16 bidirectional ports, 16 output D_flip flops with their clock inputs serially connected through a delay element, two 8-bit transparent latches to buffer output data, two multiplexers to provide two modes of output updates (checksum and immediate), and a 16-bit I/O direction register.

Each I/O port can sink 16mA to drive an output module. Each I/O port can also read an input or output module with TTL levels.

4.3.1 Updating Outputs

With the checksum enable bit reset (CONFIG.0 =0) DRACO's checksum error-checking feature is disabled and updating output ports merely consists of writing a high or low output byte without the user generating a checksum byte. DRACO implements byte integrity in updating outputs with this option selected.

With the checksum enable bit set (CONFIG.0= 1), output data is first buffered in transparent latches until a checksum byte validates the integrity of the high and low byte address/data transfer. The output of the high and low byte latch is provided to DRACO's internal full adder to generate a checksum. The sequence of writing the high and low byte prior to writing the checksum byte is irrelevant. When a valid checksum is written to DRACO, a Checksum_OK signal will be directed through the multiplexers transferring the buffered output data to the output modules. DRACO implements word integrity in updating outputs with the checksum option selected.

NOTE: The checksum referred to in all cases in this document is the inverted sum of the two output bytes.

When the buffered output data is transferred to the output drivers, the 16 outputs will not be enabled simultaneously. They will be asynchronously staggered one output at a time. This staggering turn-on of outputs will prevent undesirable effects (ground bounce) within the internal circuitry of DRACO.

The address and data parity error-checking features (CONFIG.1 & CONFIG.2) as previously described in this document operate independently on the address and data with either the checksum or immediate options.

4.3.2 I/O Direction Register

The user has the option to load a 16-bit direction register within DRACO to configure a port as read-only or bidirectional. At power-up all bits in the register will be set to "0" which allows the state of the data bit to enable the output driver (bidirectional). A "1" in the direction register will disable the tristate output driver for that corresponding port providing read-only operations from that module. The electronic key must be in the "unlock configuration" position in order to write to this register.

5. User Interfaces

5.1 Bus Interface
The bus interface includes 8 bidirectional ADDRESS/DATA pins and 1 bidirectional PARITY pin, 5 control TTL Schmitt inputs: READ_L, WRITE_L, RESET_L, CHIP_ENABLE_L, and ALE, and 1 output: ERROR_L. All inputs respond to TTL levels. All bus interface drivers are 4mA.

5.1.1 Embedded Addresses
The following address listing refers to the memory locations within DRACO.

Addr	Location		Actions Invoked
80H	Electronic Key (1st Key & Key-Off Address)	Write	- Write AAH data to turn KEY ON - Write < >AAH data to turn KEY OFF Unaffected by Ekey.
		Read	- Not available. Reading from 80H is considered an invalid address and will latch ERROR_L pin low.
7FH	Electronic Key (2nd Key Address & Data/Config Unlock Address)	Write	- Write 55H data to turn KEY ON - Write AAH to unlock Configuration - Write 55H to unlock Data Unaffected by Ekey.
		Read	- Not available. Reading from 07F is considered an invalid address and will latch ERROR_L pin low.
0FH	Error Reset	Write	- A write to this location will clear the latched interrupt. (data value is irrelevant). Unaffected by Ekey.
		Read	- Not available. Reading from 0FH is considered an invalid address and will latch ERROR_L pin low.

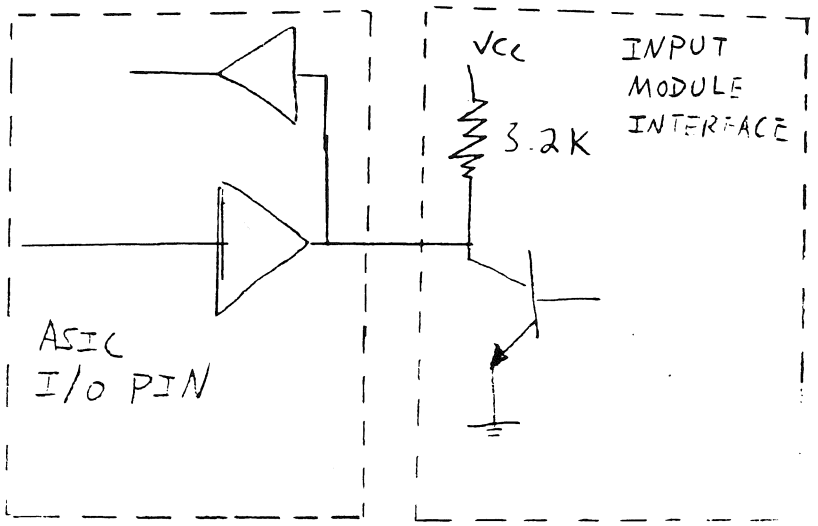
0EH	Inverted Checksum Byte	Write	- Compares host generated inverted checksum to internal inverted checksum and if equal, IO15-0 is updated with buffered data. CONFIG.1 must be set to activate this location. Unequal values will latch ERROR_L pin low. If CONFIG.1 = 0 and a write takes place to 0EH, the ERROR_L pin will also latch low. Ekey must be in the "Unlock Data" position for a valid write.
		Read	- Read the inverted sum of high and low bytes of output buffers.

04H	HB Direction IO15-8	Write	- Configures I/O ports 15-8 as either bidirectional(default) or input only Ekey must be in the "Unlock Configuration" position for a valid write.
		Read	- Read HB I/O direction register
03H	LB Direction IO7-0	Write	- Configures I/O ports 7-0 as either bidirectional(default) or input only Ekey must be in the "Unlock Configuration" position for a valid write.
		Read	- Read LB I/O direction register
02H	Configuration Register	Write	- Write to Configuration register to select checksum and parity options. Ekey must be in the "Unlock Configuration" position for a valid write.
		Read	- Read Status register
01H	High Byte IO15-8	Write	- Writes data to high byte buffer only if CONFIG.1=1. (checksum enabled) - Writes data to I/O Ports 15-8 immediately if CONFIG.1=0 (checksum disabled) Ekey must be in the "Unlock Data" position for a valid write.
		Read	- Read IO15-8
00H	Low Byte IO7-0	Write	- Writes data to low byte buffer only (CONFIG.1=1) - Writes data to I/O Ports 7-0 (CONFIG.1=0) Ekey must be in the "Unlock Data" position for a valid write.
		Read	- Read IO7-0

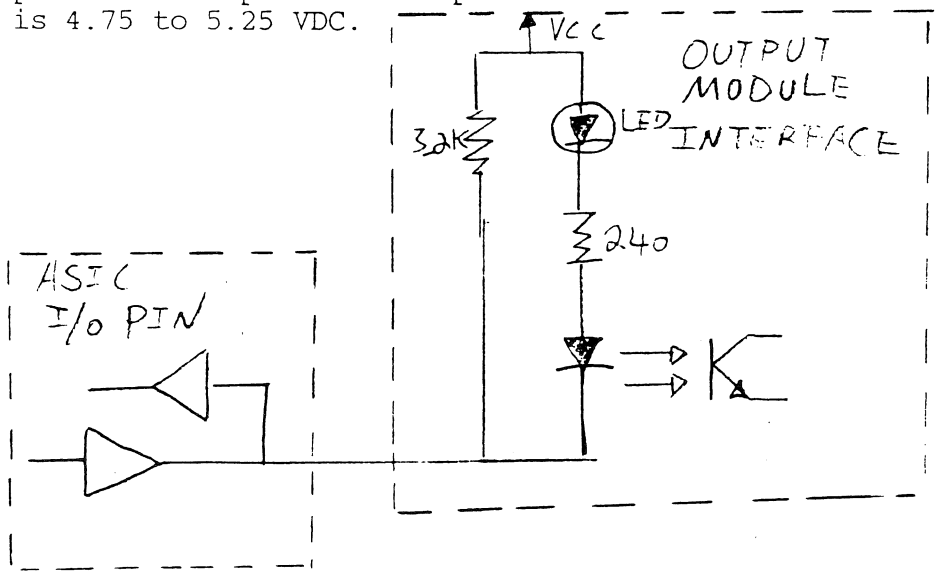
NOTE: An attempt to read/write from/to an unused address location will latch ERROR_L pin low regardless of any CONFIG.x setting.

5.2 I/O Module Interface
The circuit of an I/O module interfacing to DRACO is shown below.

5.2.1 Input Module Interface
The interface to an input module is an open collector transistor with a pull-up resistor. The Vcc range is 4.75 to 5.25VDC.



5.2.2 Output Module Interface
The interface to an output module is an opto-coupler in series with a LED and resistor. The 3.2K resistor is used for input modules but does add a 1.7mA current requirement to the output pad. A 16mA pulldown output driver will be used. The Vcc range is 4.75 to 5.25 VDC.



Pin Description

INPUT PINS (All inputs require TTL levels unless otherwise specified)

RESET_L	(3)	This input is used to initialize all internal registers and latches. It should be held low after power is applied to ensure the internal circuitry is initialized for proper operation. This is a schmitt trigger input.
ALE	(39)	This input latches the state of AD[7:0] and PARITY to internally latches. The address is latched on a high-to-low transition. This is a schmitt trigger input.
READ_L	(2)	A low on this input causes internal read data to be placed on AD[7:0] and PARITY pins. CE_L must be low simultaneously. This is a schmitt trigger input.
WRITE_L	(38)	A low-to-high transition on this input causes external data on the AD[7:0] and PARITY pins to be written into DRACO. CE_L must be low simultaneously. This is a schmitt trigger input.
CE_L	(36)	This is a Chip Enable input. This pin must be low to execute a read or write cycle. This is a schmitt trigger input.

OUTPUT PINS

ERROR_L	(37)	This active-low output will be invoked whenever an error occurs in the data transmission between DRACO and the host microprocesser. This output will latch low and must be reset by the user. This output is a low current open-drain output.

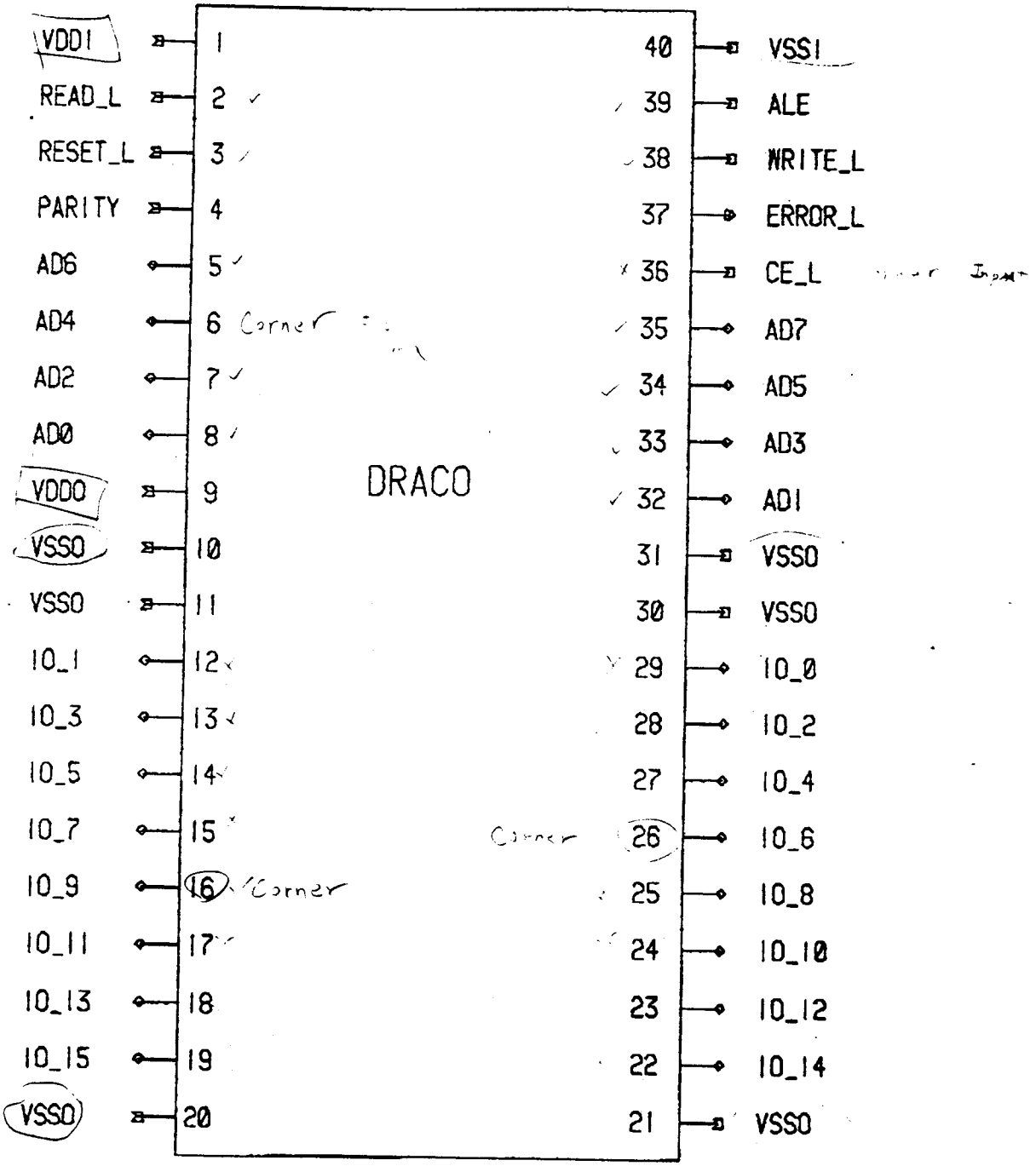
BIDIRECTIONAL PINS

PARITY	(4)	This pin receives or generates an odd parity bit for the AD[7:0] bus. It is an active-high and active-low output pin when CE_L and READ_L are low, otherwise is functions as an input.
AD[7-0]	(35,5,34,6, 33,7,32,8)	These pins make up the 8-bit address/data bus. They are active-high and active-low ouput pins when CE_L and READ_L are low. Otherwise, they function as inputs.
IO[15-0]	(19,22,18,23, 17,24,16,25, 15,26,14,27, 13,28,12,29)	These pins are high current active-low outputs that sink current from output modules connected to DRACO. These pins are always bidirectional and are not activated by READ_L or WRITE_L or CE_L pins. These pins have open-drain outputs.

Power Pins

VDD	(1,9)	These are the +5 Volt supply pins. Both must connected.
VSS	(10,11,20, 21,30,31,40)	These are the ground pins. All must be connected.

FSCM NO. 34576



FSCM NO. 34576

PIN LIST (PINS 1 - 25)

Device Name: DRACO
Device Number: 11495
Customer Part Number: 941425-61

Page 2 of 3
Date: 10-27-89
Design Engineer: Dave Pesetz

PIN NO.	PIN NAME	PIN TYPE	PIN CHAR	INPUT LEVELS	OUTPUT LEVELS	DRIVE TYPE	LOAD CAP	REQUIRED DRIVE DELAY	SSO GROUP	PAD DRIVER (T)	PAD DRIVER ((p)	DRIVER CHOSEN	COMMENTS
1.	V_{DDI}												
2.	READ_L	I		S(T)									
3.	RESET_L	I		S(T)									
4.	PARITY	I/O	OD	T	T	2X	100		1				
5.	AD6	I/O	OD	T	T	2X	100		1				
6.	AD4	I/O	OD	T	T	2X	100		1				
7.	AD2	I/O	OD	T	T	2X	100		1				
8.	AD0	I/O	OD	T	T	2X	100		1				
9.	V_{DDO}												
10.	V_{SSO}												
11.	V_{SSO}												
12.	IO1	I/O	OD	T	T	8X	75						
13.	IO3	I/O	OD	T	T	8X	75						
14.	IO5	I/O	OD	T	T	8X	75						
15.	IO7	I/O	OD	T	T	8X	75						
16.	IO9	I/O	OD	T	T	8X	75						
17.	IO11	I/O	OD	T	T	8X	75						
18.	IO13	I/O	OD	T	T	8X	75						
19.	IO15	I/O	OD	T	T	8X	75						
20.	V_{SSO}												
21.	V_{SSO}												
22.	IO14	I/O	OD	T	T	8X	75						
23.	IO12	I/O	OD	T	T	8X	75						
24.	IO10	I/O	OD	T	T	8X	75						
25.	IO8	I/O	OD	T	T	8X	75						

ROCKWELL INTERNATIONAL
PROPRIETARY INFORMATION

FSCM NO. 34576

PIN LIST (PINS 26 - 50)

Device Name: DRACO

Device Number: 11485

Customer Part Number: 941425-61

Page 3 of 3

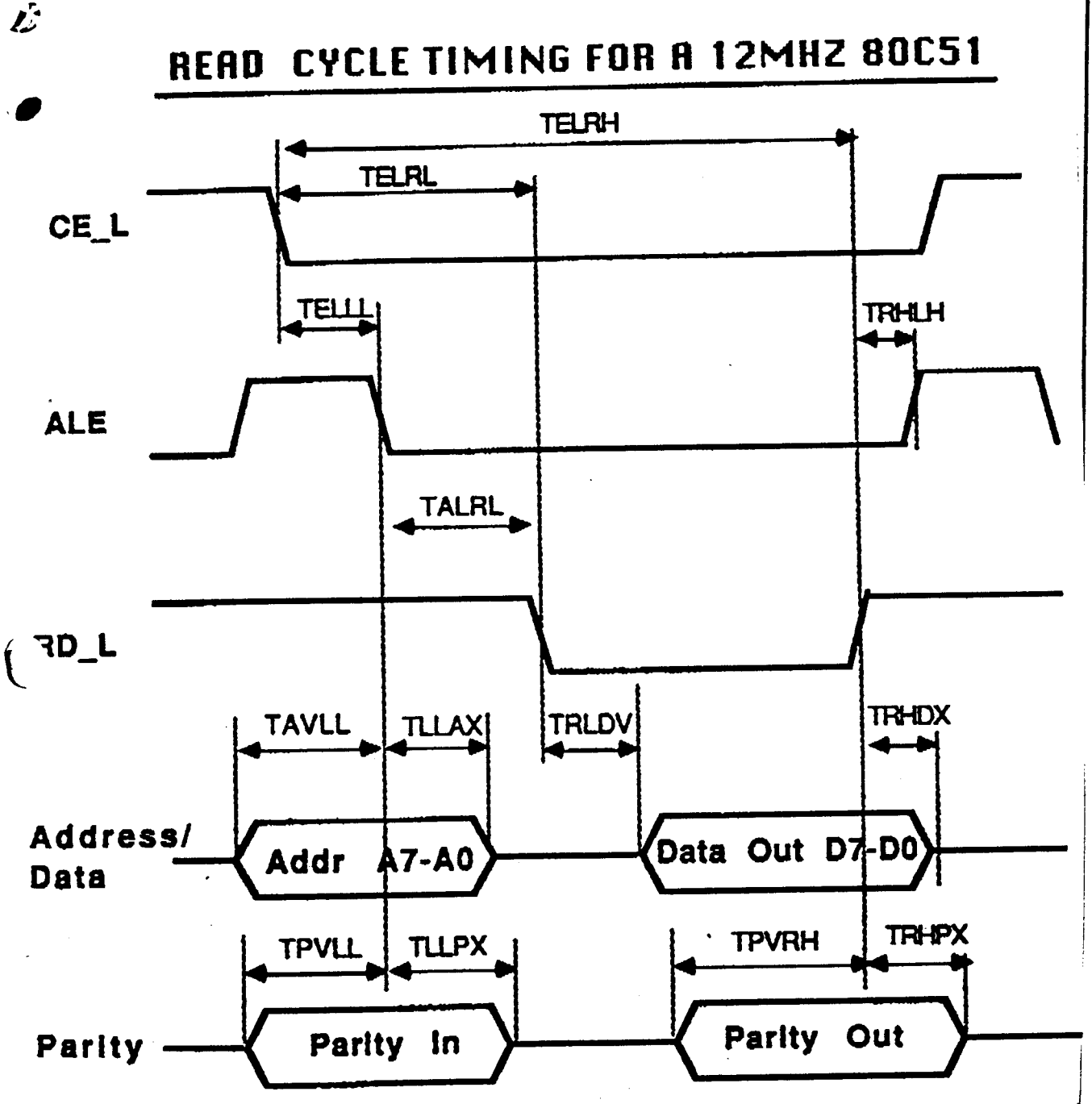
Date: 10-27-89

Design Engineer: Dave Pasela

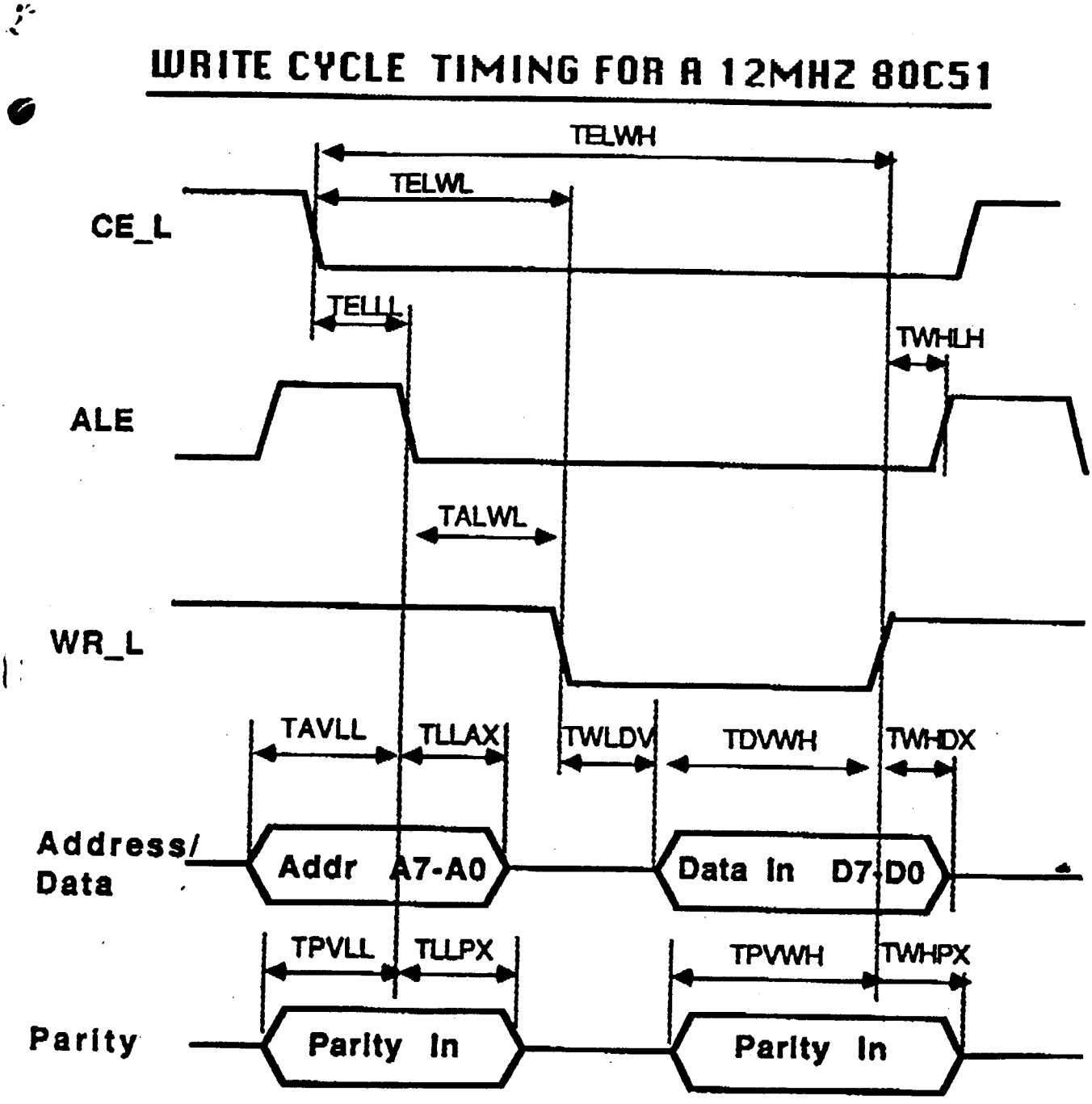
PIN NO.	PIN NAME	PIN TYPE	PIN CHAR	INPUT LEVELS	OUTPUT LEVELS	DRIVE TYPE	LOAD CAP	REQUIRED DRIVE DELAY	SSO GROUP	PAD DRIVER (Ω)	PAD DRIVER ((μ)	DRIVER CHOSEN	COMMENTS
26.	IO6	I/O	OD	T	T	8X	75						
27.	IO4	I/O	OD	T	T	8X	75						
28.	IO2	I/O	OD	T	T	8X	75						
29.	IO0	I/O	OD	T	T	8X	75						
30.	Vsso												
31.	Vsso												
32.	AD1	I/O	OD	T	T	2X	100		1				
33.	AD3	I/O	OD	T	T	2X	100		1				
34.	AD5	I/O	OD	T	T	2X	100		1				
35.	AD7	I/O	OD	T	T	2X	100		1				
36.	CEL	I		S(T)									
37.	ERROR-L	O	OD		T	2X	100						
38.	WRITE-L	I		S(T)									
39.	ALE	I		S(T)									
40.	Vssi												
41.													
42.													
43.													
44.													
45.													
46.													
47.													
48.													
49.													
50.													

5.4 Timing Diagrams

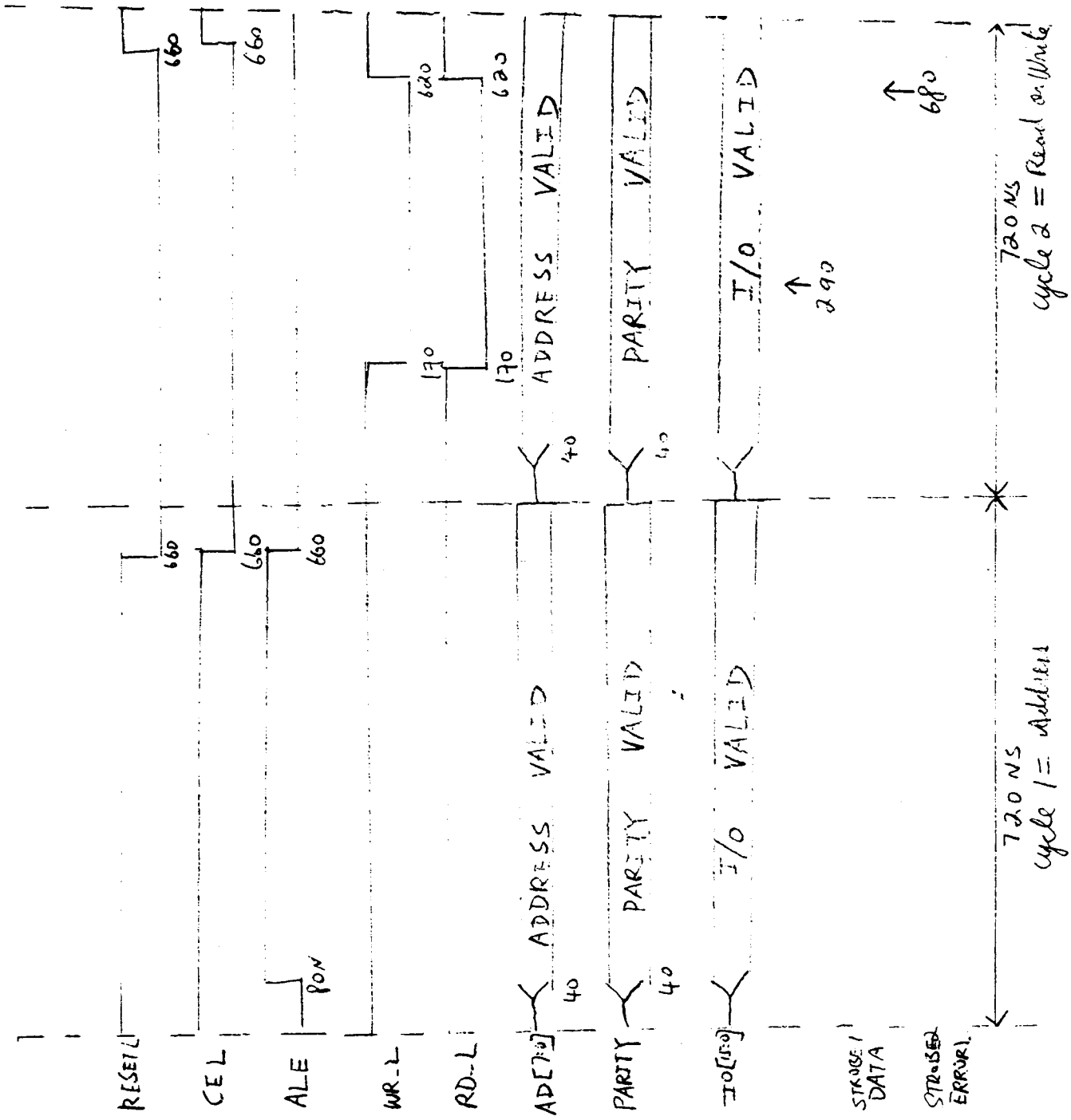
5.4.1 Read Cycle with a 12MHz 80C51



5.4.2 Write Cycle with a 12MHz 80C51



5.4.3 Read and Write Cycles Timing Values



READ CYCLE

		Min	Max	Units
TALRL	ALE low to Read_L low	200	300	ns
TAVLL	Address[7:0] valid to ALE low	8		ns
TELLL	CE_L low to ALE low	-4		ns
TELRH	CE_L low to Read_L high	571		ns
TELRL	CE_L low to Read_L low	171		ns
TLLAX	Address[7:0] valid after ALE low	41		ns
TRHDX	Data [7:0] valid after Read_L high	0		ns
TRHLH	ALE high after Read_L high	43		ns
TRLDV	Read_L low to Data[7:0] valid		227	ns
TPVRH	Parity Out valid to Read_L high	28		ns
TRHPX	Parity Out valid after Read_L high	0		ns
TPVLL	Parity In valid to ALE low	4		ns
TLLPX	Parity In valid after ALE low	45		ns

WRITE CYCLE

		Min	Max	Units
TALWL	ALE low to Write_L low	200	300	ns
TAVLL	Address[7:0] valid to ALE low	8		ns
TELLL	CE_L low to ALE low	-4		ns
TELWH	CE_L low to Write_L high	571		ns
TELWL	CE_L low to Write_L low	171		ns
TLLAX	Address[7:0] valid after ALE low	41		ns
TWHDX	Data [7:0] valid after Write_L high	26		ns
TWHLH	ALE high after Write_L high	43	123	ns
TDVWH	Data[7:0] valid to Write_L high	403		ns
TPVLL	Parity In valid to ALE low	4		ns
TPVWH	Parity In valid to Write_L high	399		ns
TLLPX	Parity In valid after ALE low	45		ns
TWHPX	Parity In valid after Write_L high	31		ns

5.4.4 Miscellaneous Timing Values

	Min	Max	Units
RESET_L pulse width (80C51 requirements)	2		usec
I/O_15 to I/O_0 Enable (staggered turn-on)		500	nsec

6. General Specifications

6.1 Packaging
The package will be a 40-pin plastic DIP.

6.2 Electrical Specifications

6.2.1 Absolute Maximum Specifications

Supply Voltage (Vdd).....-0.6 to +7.0 VDC (1)
Input Voltage ..... (Vss-0.6) to (Vdd +0.6) VDC
Output current (with no latch-up).....100 mA
Static Protection.....2000 V (2)
Operating free-air temperature range..... 0 to 850C
Storage temperature range..... -650 to 150 0C

All voltages measured with respect to device Vss pin
(1) Input voltage may be as low as -1.2V for not more than 30ns
(2) Peak voltage of the standard ESD pulse waveform.

6.2.2 Recommended Operating Conditions

		Min	Max	UNIT
Vdd	Supply voltage	4.5	5.5	V
Ta	Operating free-air temperature	0	85	C
Iol(h)	Low-level output current		16	mA
Ioh(h)	High-level output current		TBD	mA
Iol(l)	Low-level output current		4	mA
Ioh(l)	High-level output current		TBD	mA
Vih	High-level input voltage (TTL compatible)	2.0		V
Vil	Low-level input voltage (TTL compatible)		0.8	V
Vih(S)	High-level input voltage (Schmitt)	2.4		V
Vil(S)	Low-level input voltage (Schmitt)		TBD	V
Voh	High-level output voltage (at Ioh=max)	2.4		V
Vol	Low-level output voltage (at Iol=max)		0.4	V

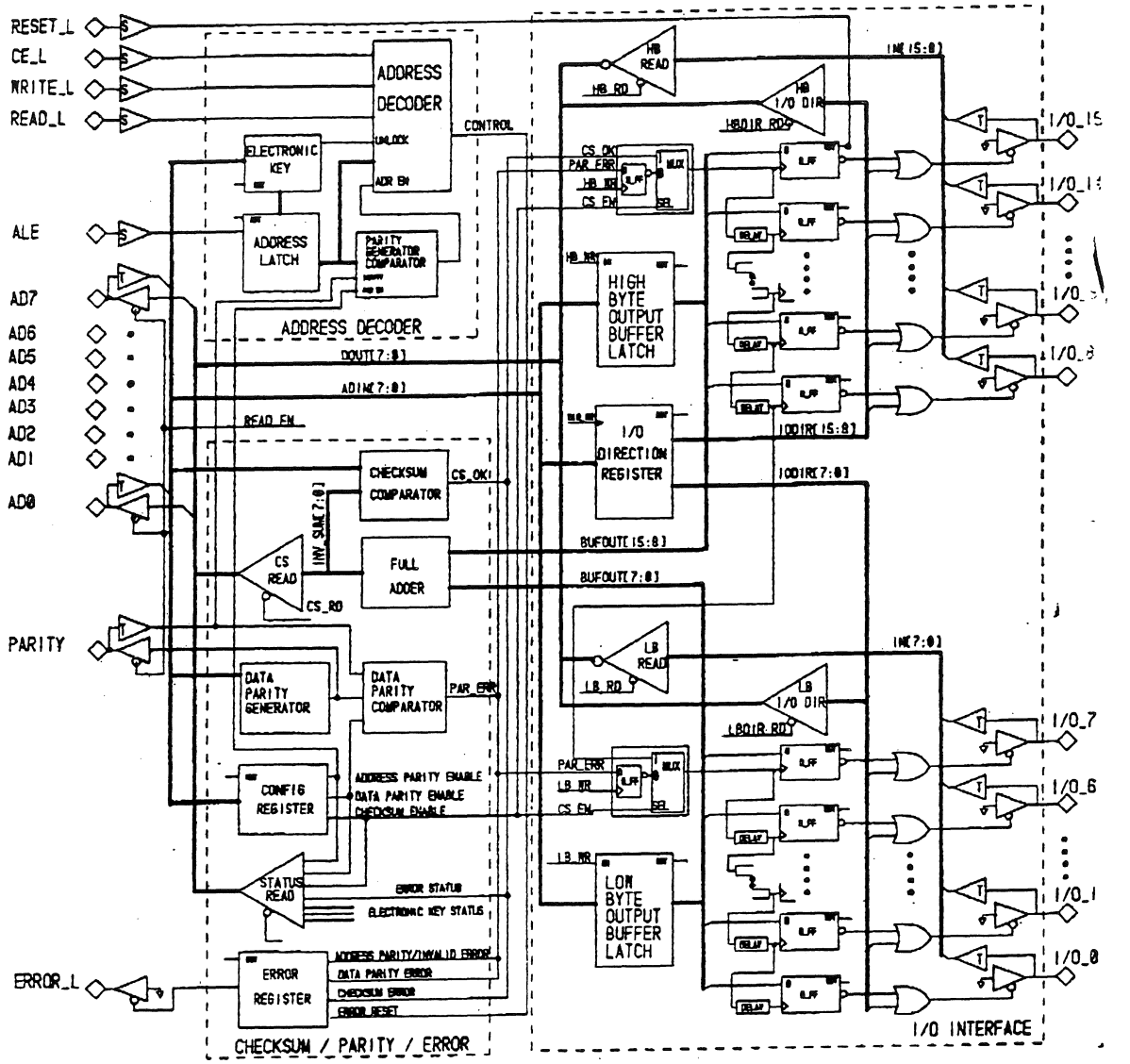
Remaining specifications will be included as they become available from the manufacturer.

Capacitive Loading

Cio_ports	75	pfd
Cdata_bus (18-inch 50-cond ribbon cable)	100	pfd

FIGURE 1 - DRACO BLOCK DIAGRAM

FIGURE 1 - DRACO BLOCK DIAGRAM



CODE IDENT. NO. 34576 | REVISION PAGE |

TDR #	REV	REASON FOR CHANGE	DATE	APPVD
	NC	Initial release.	11-20-89	

```
use work.res_funcs.all;
use work.binary_ops.all;

entity DRACO is
  port (--power: in bit;
        --CE_L: in bit;
--        RESET_L,
--        ALE,
--        WRITE_L,
--        READ_L : in bit;
        ERROR_L: out bitres ;
--        PARITY: inout bit;
--        ADD_DATA_BUS: inout Bit_vector (7 downto 0);
        MSB_IO_BUS : inout Bit_vector (15 downto 8);
        LSB_IO_BUS : inout Bit_vector (7 downto 0));
end DRACO;


architecture BEHAVIOURAL of DRACO is

-- signals from the entity
signal ALE, Reset_l, write_l,read_l: bit;
signal parity, power,ce_l: bit;
signal out_parity: bit;
signal add_data_bus, data_bus : bit_vector (7 downto 0);

signal cycle: stateres register := idle;
signal status_con_reg: bit_res_reg (7 downto 0) register;
signal add_latch: Bit_vector (7 downto 0);
signal msb_con_reg: Bit_res_reg (15 downto 8) register;
signal lsb_con_reg: Bit_res_reg (7 downto 0) register;
signal msb_buf: Bit_vector (15 downto 8);
signal lsb_buf: Bit_vector (7 downto 0);
signal ekey_mode : switchres register := off;
signal ekey_pos  : posres register := locked;
signal int_var: switch := off;
signal p_on : boolres register := false;
signal chip_en : boolres register := false;
signal valid_address: boolean;

function ODD_PARITY_fct ( vl :bit_vector) return bit is
      variable parity :  bit;
   begin
      if ((SUM(vl) mod 2) = 1) then
         return '0';
      else
         return '1';
      end if;
    end;

function ODD_PARITY_2 ( vl : bit_res_reg) return bit is
      variable parity :  bit;
   begin
      if ((SUM_2(vl) mod 2) = 1) then
         return '0';
      else
```



```
         return '1';
      end if;
    end;

begin


-- PLACE STIMULUS FILE HERE
-- 85 LINES
GENERATE_SIGNALS:
process
begin
   -- This example configures DRACO to enable data parity
   -- address parity and checksum. The key is in unlock config
   -- positions while configuring. Thereafter the data is
   -- unlocked and data is written to the lsb and msb I/O
   -- ports. Subsequently, checksum is written to and read from
   -- DRACO. A successful write of checksum writes data onto
   -- the I/O ports.

   -- POWER SIGNAL
   power <= '0',
            '1' after 25 fs,
            '0' after 1920 fs;

   -- CHIP ENABLE SIGNAL
   ce_l <= '1',
           '0' after 50 fs,
           '1' after 1910 fs;


    ale <= '1', '0' after 110 fs, '1' after 170 fs,
                '0' after 210 fs, '1' after 270 fs,
                '0' after 310 fs, '1' after 370 fs,
                '0' after 410 fs, '1' after 470 fs,
                '0' after 510 fs, '1' after 570 fs,
                '0' after 610 fs, '1' after 670 fs,
                '0' after 710 fs, '1' after 770 fs,
                '0' after 810 fs, '1' after 870 fs,
                '0' after 910 fs, '1' after 970 fs;



    add_data_bus <= X"80" after 105 fs,
                    X"AA" after 140 fs,
                    X"7F" after 205 fs,
                    X"55" after 240 fs,
                    X"7F" after 305 fs,
                    X"AA" after 340 fs,
                    X"02" after 405 fs,
                    X"07" after 440 fs,
                    -- unlock data
                    X"7F" after 505 fs,
                    X"55" after 540 fs,
                    -- write into the lsb buf
                    X"00" after 605 fs,
```

```
                    X"08" after 640 fs,
                    -- write into the msb_buf
                    X"01" after 705 fs,
                    X"04" after 740 fs,
                    -- write inverted checksum
                    X"0E" after 805 fs,
                    X"F3" after 840 fs,
                    -- read checksum
                    X"0E" after 905 fs;


    write_l <= '1', '0' after 130 fs, '1' after 160 fs,
                    '0' after 230 fs, '1' after 260 fs,
                    '0' after 330 fs, '1' after 360 fs,
                    '0' after 430 fs, '1' after 460 fs,
                    '0' after 530 fs, '1' after 560 fs,
                    '0' after 630 fs, '1' after 660 fs,
                    '0' after 730 fs, '1' after 760 fs,
                    '0' after 830 fs, '1' after 860 fs;

   read_l  <=  '1',
               '0' after 930 fs;

    -- PARITY SIGNAL
    parity <= '0' after 505 fs,
              '1' after 540 fs,
              '1' after 605 fs,
              '0' after 640 fs,
              '0' after 705 fs,
              '0' after 740 fs,
              '0' after 805 fs,
              '1' after 840 fs,
              '0' after 905 fs;

    wait;
end process GENERATE_SIGNALS;


rising_power_signal:
block (power = '1' and not power'stable)
begin
     cycle <= guarded reset;
     p_on <=  guarded true;
end block rising_power_signal;

falling_power_signal:
block (power = '0'and not power'stable)
begin
    cycle <= guarded power_off;
    p_on <=  guarded false;
end block falling_power_signal;

falling_chip_enable:
block (ce_l = '0' and not ce_l'stable and p_on)
begin
      cycle <= guarded chip_enabled;
```



```
      chip_en <= guarded true;
end block falling_chip_enable;

rising_chip_enable:
block (ce_l='1' and not ce_l'stable and chip_en)
begin
    cycle <= guarded chip_disable;
    chip_en <= guarded false;
end block rising_chip_enable;

falling_reset_signal:
block (reset_l = '0' and not reset_l'stable and p_on and chip_en)
begin
      cycle <= guarded reset;
end block falling_reset_signal;

falling_ale_signal:
block (ale = '0' and not ale'stable and p_on and chip_en)
begin
       cycle <= guarded address;
end block falling_ale_signal;

rising_ale:
block (ale = '1' and ale'active and p_on and chip_en)
begin
        cycle <= guarded idle;
end block rising_ale;

falling_write:
block(write_l='0'and(not write_l'stable) and p_on and chip_en and valid_address)
begin
       cycle <=  guarded write;
end block falling_write;

falling_read_signal:
block(read_l ='0' and (not read_l'stable) and p_on and chip_en and valid_address
begin
       cycle <= guarded read;
end block falling_read_signal;

-- assertions when CE_L goes low
   assert not(ce_l='0' and not ce_l'stable and not p_on)
   report "ERROR 1"
   severity warning;

-- assertions when RESET_L  goes low
   -- check to see that power is on
   assert not(reset_l = '0' and not reset_l'stable and not p_on)
   report "ERROR 2"
   severity warning;
   -- check to see that chip is enabled
   assert not(reset_l='0' and not reset_l'stable and not chip_en)
   report "ERROR 3"
   severity warning;

-- assertions when ALE goes low
```

```
    -- check if power is off
    assert not(ale = '0' and not ale'stable and not p_on)
    report "ERROR 4"
    severity warning;
    -- check if chip is enabled
    assert not(ale = '0' and not ale'stable and not chip_en)
    report"ERROR 5"
    severity warning;


-- assertions when WRITE_L goes low
    -- check if power is off
    assert not(write_l = '0' and not write_l'stable and not p_on)
    report "ERROR 6"
    severity warning;
    -- check if chip is enabled
    assert not(write_l = '0' and not write_l'stable and not chip_en)
    report "ERROR 7"
    severity warning;
    -- check if valid address is available
    assert not(write_l='0' and not write_l'stable and not valid_address)
    report "ERROR 8"
    severity warning;

-- assertions when READ_L goes low
    -- check if power is off
    assert not(read_l = '0' and not read_l'stable and not p_on)
    report "ERROR 9"
    severity warning;
    -- check if chip is enabled
    assert not(read_l = '0' and not read_l'stable and not chip_en)
    report"ERROR 10 "
    severity warning;
    -- check if valid address is available
    assert not(read_l='0' and not read_l'stable and not valid_address)
    report "ERROR 11"
    severity warning;


-- ADDRESS, WRITE, RESET AND READ CYCLES FOLLOW


ADDRESS_LATCH:
block
  signal a: boolean := true;
-- the address is read from the address data bus and is stored
-- in a latch. It is necessary to store the address in a latch so
-- that the write cycle can access it even after it is no longer
-- available at the address data bus. The process is sensitive to
-- the ALE signal
begin
process
variable count: integer;
variable valid_add : boolean;
begin
        wait until (not cycle'stable and cycle = address);
```



```
        a <= not a;
        valid_address <= true;
        valid_add:= true;
        -- check if address parity is  on
        -- if parity is on check for address parity
        -- if parity is incorrect latch error_l low
        -- and make valid address false
        if (status_con_reg(2) = '1') then
            if (odd_parity_fct(add_data_bus)/= parity) then
               error_l <= '0';
               status_con_reg(3) <= '1';
               valid_address <= false;
               valid_add := false;
               assert false
               report "ERROR 12"
               severity warning;
            end if;
        end if;

        if (valid_add = true)  then
         -- is address one of the 9 valid addresses
               if (not(VALID_ADDRESS_FUN(add_data_bus))) then
                 error_l <= '0';
                 status_con_reg(3) <= '1';
                 valid_address <= false;
                 valid_add := false;
                 assert false
                 report "ERROR 13"
                 severity warning;
               end if;

         -- the valid address is latched onto the add_latch signal
               if (valid_add = true) then
                  add_latch <= add_data_bus;
               end if;
        end if;

        wait on a;
        status_con_reg <= null;

end process;
end block ADDRESS_LATCH;


WRITE_CYCLE:
block
  signal a: boolean := true;
begin
process
  variable count: integer;
  variable valid_checksum,valid_data: boolean;
  variable checksum: bit_vector (7 downto 0);

begin

    wait  until (cycle = write and not add_data_bus'stable);
```

```
    wait until (write_l ='1' and not write_l'stable );
    a <= not a;
    valid_data := true;

    -- check if data parity is on
    if (status_con_reg(1) = '1') then
      if (add_latch /= X"0F") then
          -- do a parity check on the data
          if (odd_parity_fct(add_data_bus)/=parity) then
              -- error in data received
              assert false
              report "ERROR 14"
              severity warning;
              error_l <= '0';
              status_con_reg(3)<= '1';
              valid_data := false;
          end if;
       end if;
    end if;

    status_con_reg(4) <= '0';

    if (valid_data = true) then

       case add_latch is
       -- is address 80H
       when X"80" =>
          if (add_data_bus /= X"AA") then
             -- turn the key off
              ekey_mode <=  OFF;
              status_con_reg(5) <= '0';
          elsif (ekey_mode = OFF and add_data_bus = X"AA") then
             int_var <= ONN;
          elsif (ekey_mode = Onn and add_data_bus =X"AA") then
             assert false
             report "ERROR 15"
             severity warning;
          end if;

       -- is address 7FH
       when X"7F" =>
           if (add_data_bus = X"55") then
              if (int_var = ONN) then
                   ekey_mode <= onn;
                   status_con_reg(5) <= '1';
                   int_var <= off;
              else
                    if (ekey_mode = onn) then
                      ekey_pos <= data_unlocked;
                      status_con_reg(6) <=   '1';
                      status_con_reg(7) <= '0';
                    end if;
              end if;
           elsif (add_data_bus = X"AA") then
                    if (ekey_mode = onn) then
                      ekey_pos <= config_unlocked;
```



```
                      status_con_reg(6) <=   '0';
                      status_con_reg(7) <= '1';
                    end if;
          end if;

       -- is address 0FH (error reset)
       when X"0F" =>
             error_l <= '1';
             status_con_reg(3) <= '0';

       -- is address 04H (address of MSB IO ports configuration
       when X"04" =>
          -- config should be unlocked to configure io ports
          if (status_con_reg(7) = '1' ) then
          loop2:
          for I in 8 to 15 loop
                msb_con_reg(I)<= add_data_bus(I-8);
          end loop loop2;
          else assert false
               report "ERROR 16"
               severity warning;
               error_l <= '0';
               status_con_reg(3) <= '1';
          end if;

       -- is address 03H (address of LSB IO port config register)
       when X"03" =>
          -- change the config only if config is unlocked
          if (status_con_reg(7) = '1') then
          loop3:
          for I in 0 to 7 loop
                lsb_con_reg (I)<= add_data_bus(I);
          end loop loop3;
          else assert false
               report "ERROR 17"
               severity warning;
               error_l <= '0';
               status_con_reg(3) <= '1';
          end if;

       -- is address 02H (address of configuration register)
       when X"02" =>
          -- config should be unlocked to change draco config
          if (status_con_reg(7) = '1') then
             status_con_reg(0) <=  add_data_bus(0);
             status_con_reg(1) <=  add_data_bus(1);
             status_con_reg(2) <=  add_data_bus(2);
          else  assert false
                report "ERROR 18"
                severity warning;
                error_l <= '0';
                status_con_reg(3) <= '1';
          end if;

       -- for addresses 0EH, 01H, 00H check to see if the data is onn
       when others =>
```

```
ekey_mode <= off;
status_con_reg(5) <= '0';
if (ekey_pos  = data_unlocked) then
-- is address 01H (address of msbyte io ports)
case add_latch is
when X"01" =>
 -- if checksum is on then
 if (status_con_reg(0)='1' ) then
    -- store data in msb buffer
    for I in 8 to 15 loop
        msb_buf(I) <= add_data_bus(I-8);
    end loop;
 else -- update the msb io ports
    loop7:
    for I in 8 to 15 loop
        msb_io_bus(I) <= add_data_bus (I-8);
    end loop loop7;
    status_con_reg(4) <= '1';
 end if;

-- is address 00H ( address of lsb io ports)
when X"00" =>
-- if checksum is on then
 if (status_con_reg(0) ='1') then
   -- store data in the lsb buf
   loop4:
   for I in 0 to 7 loop
       lsb_buf(I) <= add_data_bus(I);
   end loop loop4;
 else -- update the lsbyte of the io ports
    loop5:
    for I in 0 to 7 loop
       lsb_io_bus(I) <= add_data_bus(I);
    end loop loop5;
    status_con_reg(4) <= '1';
 end if;

-- is address 0EH (address of the checksum)
when others  =>
    -- check to see that the checksum option is enabled
    if (status_con_reg(0) /= '1') then
        assert false
        report "ERROR 19"
        severity warning;
        error_l <= '0';
        status_con_reg(3) <= '1';
    else -- generate the checksum
         -- compare it with the data on the address data
         -- bus. if they tally then update the ports of
         -- draco with the contents of the buffer

        -- generate the sum
           checksum := "+"(lsb_buf,msb_buf);

        -- invert the generated sum
           checksum := ones_comp(checksum);
```



```
           -- compare the inverted checksum
           valid_checksum := compare(checksum,add_data_bus);

            if (not valid_checksum) then
               assert false
               report "ERROR 20"
               severity warning;
               error_l <= '0';
               status_con_reg(3) <= '1';
        else -- update the io ports with the contents of the buffers
               loop4e:
               for I in 0 to 7 loop
             if  (lsb_con_reg(I)='0')then-- port is  bidirectional
                       lsb_io_bus(I) <= lsb_buf(I);
             end if;
             if  (msb_con_reg(I+8) ='0') then
                     msb_io_bus(I+8) <= msb_buf(I+8);
             end if;
                   end loop loop4e;
              status_con_reg(4) <= '1';
              end if;
             end if;

         end case;
        else assert false
        report " ERROR 21"
        severity warning;
        error_l <= '0';
        status_con_reg(3) <= '1';

       end if;
    end case;

    end if;

    wait on a;
    status_con_reg <= null;
    lsb_con_reg <= null;
    msb_con_reg <= null;
    ekey_pos <= null;
    ekey_mode <= null;

end process;
end block WRITE_CYCLE;


RESET_CYCLE:
block
signal a: boolean := true;
begin
process
begin
    wait until (not cycle'stable and cycle = reset);
    a<= not a;
    -- disable checksum
```

```
ekey_mode <= off;
status_con_reg(5) <= '0';
if (ekey_pos  = data_unlocked) then
-- is address 01H (address of msbyte io ports)
case add_latch is
when X"01" =>
 -- if checksum is on then
 if (status_con_reg(0)='1' ) then
    -- store data in msb buffer
    for I in 8 to 15 loop
        msb_buf(I) <= add_data_bus(I-8);
    end loop;
 else -- update the msb io ports
    loop7:
    for I in 8 to 15 loop
        msb_io_bus(I) <= add_data_bus (I-8);
    end loop loop7;
    status_con_reg(4) <= '1';
 end if;

-- is address 00H ( address of lsb io ports)
when X"00" =>
-- if checksum is on then
 if (status_con_reg(0) ='1') then
   -- store data in the lsb buf
   loop4:
   for I in 0 to 7 loop
       lsb_buf(I) <= add_data_bus(I);
   end loop loop4;
 else -- update the lsbyte of the io ports
    loop5:
    for I in 0 to 7 loop
       lsb_io_bus(I) <= add_data_bus(I);
    end loop loop5;
    status_con_reg(4) <= '1';
 end if;

-- is address 0EH (address of the checksum)
when others  =>
    -- check to see that the checksum option is enabled
    if (status_con_reg(0) /= '1') then
        assert false
        report "ERROR 19"
        severity warning;
        error_l <= '0';
        status_con_reg(3) <= '1';
    else -- generate the checksum
         -- compare it with the data on the address data
         -- bus. if they tally then update the ports of
         -- draco with the contents of the buffer

        -- generate the sum
           checksum := "+"(lsb_buf,msb_buf);

        -- invert the generated sum
           checksum := ones_comp(checksum);
```



```
           -- compare the inverted checksum
           valid_checksum := compare(checksum,add_data_bus);

            if (not valid_checksum) then
               assert false
               report "ERROR 20"
               severity warning;
               error_l <= '0';
               status_con_reg(3) <= '1';
        else -- update the io ports with the contents of the buffers
               loop4e:
               for I in 0 to 7 loop
             if  (lsb_con_reg(I)='0')then-- port is  bidirectional
                       lsb_io_bus(I) <= lsb_buf(I);
             end if;
             if  (msb_con_reg(I+8) ='0') then
                      msb_io_bus(I+8) <= msb_buf(I+8);
             end if;
                   end loop loop4e;
              status_con_reg(4) <= '1';
              end if;
             end if;

         end case;
        else assert false
        report " ERROR 21"
        severity warning;
        error_l <= '0';
        status_con_reg(3) <= '1';

       end if;
    end case;

    end if;

    wait on a;
    status_con_reg <= null;
    lsb_con_reg <= null;
    msb_con_reg <= null;
    ekey_pos <= null;
    ekey_mode <= null;

end process;
end block WRITE_CYCLE;


RESET_CYCLE:
block
signal a: boolean := true;
begin
process
begin
    wait until (not cycle'stable and cycle = reset);
    a<= not a;
    -- disable checksum
```

```
    status_con_reg(0) <=  '0';
    -- disable data parity
    status_con_reg(1) <='0';
    -- disable address parity
    status_con_reg(2) <=  '0';
    -- error reset state
    status_con_reg(3) <= '0';
    -- write acknowledge
    status_con_reg(4)<=  '0';
    -- ekey is off
    ekey_mode <= off;
    status_con_reg(5) <= '0';
    -- ekey mode is locked
    ekey_pos <= locked;
    status_con_reg(6) <=  '0';
    status_con_reg(7) <=  '0';
    -- ports are configured as bidirectional
    msb_con_reg <= X"00";
    lsb_con_reg <= X"00";

    wait on a;

    status_con_reg <= null;
    ekey_mode <= null;
    ekey_pos  <= null;
    lsb_con_reg <= null;
    msb_con_reg <= null;

end process;
end block RESET_CYCLE;

READ_CYCLE:
block
   signal a: boolean := true;
begin
  process
   variable count : integer;
   variable checksum: bit_vector (7 downto 0);
  begin
       wait until (cycle = read and not cycle'stable);

       a <= not a;

       if (add_latch=X"80" or add_latch=X"7F" or add_latch=X"0F" )
       then error_l <= '0';
            status_con_reg(3) <= '1';
            assert false
            report "ERROR 22"
            severity warning;

       elsif (add_latch = X"0E") then
              -- generate the sum
                 checksum := "+"(lsb_buf,msb_buf);
              -- invert the generated sum
                 checksum := ones_comp(checksum);
                 data_bus <= checksum after 20 fs;
```



```
              if (status_con_reg(1)='1') then
                   out_parity <= odd_parity_fct(checksum) after 20 fs;
              end if;

       else
             case add_latch is
               when X"04" =>
                     for I in 0 to 7 loop
                        data_bus(I) <= msb_buf(I+8) after 20 fs;
                     end loop;
                     if (status_con_reg(1) = '1') then
                         out_parity <= odd_parity_fct(msb_buf) after 20 fs;
                     end if;
               when X"03" =>
                     data_bus <= lsb_buf after 20 fs;
                     if (status_con_reg(1) = '1') then
                          out_parity <= odd_parity_fct(lsb_buf) after 20 fs;
                     end if;
               when X"02" =>
                     for I in 0 to 7 loop
                       data_bus(I) <= status_con_reg(I) after 20 fs;
                     end loop;
                     if(status_con_reg(1) = '1') then
                       out_parity <= odd_parity_2(status_con_reg) after 20 fs;
                     end if;
               when X"01" =>
                     for I in 0 to 7 loop
                       data_bus (I) <= msb_io_bus(I+8) after 20 fs;
                     end loop;
                     if (status_con_reg(1) = '1') then
                      out_parity<= odd_parity_fct(msb_io_bus) after 20 fs;
                     end if;
               when X"00" =>
                     data_bus <= lsb_io_bus after 20 fs;
                     if (status_con_reg(1) = '1') then
                      out_parity<= odd_parity_fct(lsb_io_bus) after 20 fs;
                     end if;
               when others =>
                       count := 0 ;
             end case;
          end if;
          wait on a;
          status_con_reg(3)<= null;
    end process;
end block READ_CYCLE;



end;


configuration behavioural_con of draco is

   for behavioural
   end for;
```

```
end behavioural_con;
```



```
package res_funcs is

type bit_res is array (Natural Range <>) of bit;
type state is(write, read, address,reset,idle,chip_disable,power_off,chip_enable
type state_res is array (natural range <>) of state;
type switch is (off,onn);
type switch_res is array (natural range <>) of switch;
type bool_res is array (natural range <>) of boolean;
type pos is (data_unlocked, config_unlocked, locked);
type pos_res is array (natural range <>) of pos;


function Bit_res_fun (input : bit_res) return bit;
function state_res_fun (input: state_res ) return state;
function switch_res_fun(input: switch_res) return switch;
function bool_res_fun (input : bool_res) return boolean;
function pos_res_fun (input : pos_res) return pos;

subtype bitres is bit_res_fun bit ;
subtype stateres is state_res_fun state ;
subtype switchres is switch_res_fun switch ;
subtype boolres is bool_res_fun boolean;
subtype posres is pos_res_fun pos;

type bit_res_reg is array (integer range <>) of bitres;
end res_funcs;

package body res_funcs is

function state_res_fun (input: state_res ) return state is
variable a: boolean := false;
begin
   a:= false;
   loop3:
   for I in Input'range loop
      if (input(I)= power_off) then
          return Input(I);
          a:=true;
      end if;
   end loop loop3;
   if (a=false) then
      return Input(0);
   end if;
end state_res_fun;

function Bit_res_fun (input : bit_res) return bit is
begin
    return input(0);
end bit_res_fun;

function switch_res_fun (input: switch_res) return switch is
begin
    return input(0);
end switch_res_fun;

function bool_res_fun (input : bool_res) return boolean is
```

```
begin
    return input(0);
end bool_res_fun;

function pos_res_fun (input : pos_res) return pos is
begin
    return input(0);
end pos_res_fun;

end res_funcs;
```



```
use work.res_funcs.all;

package binary_ops is

   function ONES_COMP (v2: BIT_VECTOR) return BIT_VECTOR;
   function "+" (x1, x2: BIT_VECTOR) return BIT_VECTOR;
   function COMPARE (x1,x2 : BIT_VECTOR) return boolean;
   function EVEN_PARITY (x1: BIT_VECTOR) return bit;
   function VALID_ADDRESS_FUN(x1: BIT_VECTOR) return boolean;
   function SUM(v2 : BIT_VECTOR) return integer;
   function SUM_2(v2 :bit_res_reg)  return integer;

end binary_ops;

package body binary_ops is

--*************************************************************************
function ones_comp (v2: BIT_VECTOR) return BIT_VECTOR is

   variable v1 : BIT_VECTOR(v2'high downto v2'low);
    variable temp: BIT_VECTOR(v1'range);
    variable I: INTEGER;

   begin
      v1 := v2;
      for I in v1'range loop
         if v1(I) = '0' then
            temp(i) := '1';
         else
            temp(i) := '0';
         end if;
      end loop;

      return temp;
end ones_comp;
--*************************************************************************

--*************************************************************************
function "+"(x1, x2 :BIT_VECTOR) return BIT_VECTOR is

   variable v1 : BIT_VECTOR(x1'high downto x1'low);
   variable v2 : BIT_VECTOR(x2'high downto x2'low);
   variable CARRY: BIT := '0';
   variable S: BIT_VECTOR(1 to 3);
   variable NUM: INTEGER range 0 to 3 := 0;
   variable SUM: BIT_VECTOR(v1'range);
   variable I,K,L: INTEGER;

   begin
      v1 := x1;
      v2 := x2;
      assert v1'length = v2'length
      report "BIT_VECTOR +: operands of unequal lengths"
      severity FAILURE;

      for I in v1'low to v1'high loop
```

```
        L:= I + v2'low;
        S:= v1(I) & v2(L) & CARRY;
        NUM := 0;

        for K in 1 to 3 loop
           if S(K) = '1' then
              NUM := NUM + 1;
           end if;
        end loop;

        case NUM is
           when 0 => SUM(I) := '0'; CARRY := '0';
           when 1 => SUM(I) := '1'; CARRY := '0';
           when 2 => SUM(I) := '0'; CARRY := '1';
           when 3 => SUM(I) := '1'; CARRY := '1';
        end case;
     end loop;

     return SUM;
end "+";
--*************************************************************************

--*************************************************************************
function COMPARE (x1, x2: BIT_VECTOR) return boolean is

      variable v1 : Bit_VECTOR(x1'high downto x1'low);
      variable v2 : BIT_VECTOR(x2'high downto x2'low);
      variable temp : boolean;

   begin

      temp := true;
      v1 := x1;
      v2 := x2;
      assert v1'length = v2'length
      report "BIT_VECTOR COMPARE: operands of unequal lengths"
      severity FAILURE;

      for I in v1'low to v1'high loop
            if (((v1(I))xor(v2(I)))='1') then
                temp := false;
            end if;
      end loop;

      return temp;

end COMPARE;
--*************************************************************************

--*************************************************************************
function SUM(v2 : BIT_VECTOR) return integer is

   variable v1 : BIT_VECTOR(v2'high downto v2'low);
   variable count : integer := 0;

   begin
```



```
      v1 := v2;
      for I in v1'high downto v1'low loop
          if (v1(I) = '1') then
              count := count + 1;
          end if;
      end loop;

      return count;
end SUM;
--*************************************************************************

--*************************************************************************
function SUM_2(v2 : bit_res_reg) return integer is

   variable v1 : BIT_RES_REG(v2'high downto v2'low);
   variable count : integer := 0;

 · begin
      v1 := v2;
      for I in v1'high downto v1'low loop
          if (v1(I) = '1') then
              count := count + 1;
          end if;
      end loop;

      return count;
end SUM_2;
--*************************************************************************

--*************************************************************************
function EVEN_PARITY (x1: BIT_VECTOR) return bit is

      variable v1 : BIT_VECTOR(x1'high downto x1'low);
      variable temp: bit;

   begin

      temp := '0';
      v1:=x1;
      for I in v1'high to v1'low loop
           if v1(I) = '1' then temp:= not(temp);
           end if;
      end loop;

      return temp;

end EVEN_PARITY;
--*************************************************************************

--*************************************************************************
function VALID_ADDRESS_FUN (x1: BIT_VECTOR) return boolean is

      variable temp: boolean;
      variable v1: BIT_VECTOR (x1'high downto x1'low);

  begin
```

```
    v1:= x1;
    temp := v1=X"00" or v1=X"01"
            or v1=X"02" or v1=X"03"
            or v1=X"04" or v1=X"0E"
            or v1=X"0F" or v1=X"7F"
            or v1=X"80";

    return temp;

end VALID_ADDRESS_FUN;
--*****************************************************************

--*****************************************************************
end binary_ops;
```