ARIX Systems Corporation

Test specification for :

I/O Processor Module Diagnostics

Rev 2.0

by Mike Haley, 4-May-88

Revision record

1.00 20-Jan-88 Preliminary

Reference documents

Arete : System 3000 Hardware Specification, May 4 ,1987
Arete : IOPM Hardware Specification, ver 1.0, Nov 5, 1987

Document overview

This document is intended to provide a description of the bring-up, power-on, and manufacturing diagnostics that are to be provided for the IOPM board.

CONTENTS

1. Introduction

The I/O Processor Module (IOPM) is a high performance controller for System 3000. The controller is generic in the sense that it is a general purpose controller with different types of device boards connected via an interface bus.

2. Hardware features

The main sections of the IOPM are:
- A 20MHz 68030 processor
- Status LEDs
- 32 KByte EPROM
- 1 MByte of parity protected DRAM
- 32 bit local bus
- Control/Status registers
- I/O Map
- Interrupt control
- 3 timers
- CSS-bus interface
- Device board interface
- I/O interface for diag-bag.

3. Types of Service Module Diagnostics

3.1 IOPM engineering verification

These tests will allow the hardware engineer to verify that the hardware design works as expected. The engineering tests will consist of scope loops and board bring-up/debug diagnostics.

The scope loops will reside in EPROM.

The engineering verification tests will reside in EPROM or be downloadable to RAM.

Diag Bag will be supported.

ICE will also be used during engineering verification.

3.2 IOPM power-on confidence test

The IOPM power-on confidence tests will be a series of EPROM based tests executed automatically by the IOPM on power-up. These tests will begin by checking the CPU, checksum the EPROM, test RAM, and then the CSS bus interface. Once completed the appropriate LEDs will be set and will wait for further commands from the Service Module.

The power-on confidence tests will be developed after the engineering verification tests are completed and will be a subset of the engineering verification diagnostics.

3.3 IOPM manufacturing diagnostics

The IOPM manufacturing diagnostics will be a copy of the engineering verification tests. These diagnostics will provide as much detailed information as possible to help in the isolation of faults to the component level.

4. Description of tests

The following is a description of the diagnostics which will be provided for the IOPM board.

4.1 IOPM scope loop diagnostics

The scope loop diagnostics will reside in EPROM.

One test per EPROM.

The scope loops provided are as follows:

4.1.1 Reset to processor

Provide a 1Khz reset signal to the 68030.

4.1.2 LEDs

A loop that writes an incrementing pattern to all the software controlled LEDs at a visible rate.

4.1.3 Address Bus

Loop that exercises all address lines.

4.1.4 Data bus

Loop that writes to one location exercising all data lines.

4.1.5 EPROM

Loop on a checksum test of the EPROM

4.1.6 I/O map Registers

Write/Read to all fifteen map registers

4.1.7 Registers

Write/Read to all available registers on the board in a non-fatal manner.

4.1.8 Diag Bag I/O interface

- One PROM that will set up the serial interface and output a stream of data.
- One PROM that will set up the serial interface and loop reading characters and echoing them back.

4.2 Power-on confidence tests

When power is applied to a IOPM board, the IOPM board will be held in a reset state until it receives an ENABLE_MODULE cmd from the service module.
The tests executed by the power on confidence test are as follows:

4.2.1 EPROM

Checksum the on board EPROM.

4.2.2 CPU

Write/Read/Verify Registers.
Verify cache operates properly.
Verify TLB operates properly.

4.2.3 RAM

Fast address test.
Fast data test.
Misaligned read, write test.
Burst read test.

4.2.4 On board registers

Write, read all available register.

4.2.5 Interrupt logic

Verify all on board interrupts work properly.

4.2.6 Parity test

Verify parity generator generates proper parity.
Verify parity checker works properly.

4.2.7 CSS bus

Verify we can write data to ourselves.
Verify the the CSS interface can detect parity errors by sending a command to ourselves with bad parity.

4.3 Engineering verification diagnostics

These diagnostics will provide the primary bring up and debug tools for the IOPM board. This program will consist of a menu driven interface with a number of diagnostic tests designed to detect and identify faults.

The following tests will be part of this diagnostic package:

4.3.1 CPU test

- Verify the CPU registers operate properly.
- Verify cache operates properly.
- Verify MMU operates properly.

4.3.2 EPROM test

- Checksum the EPROM.
- Verify checksum is correct.

4.3.3 LEDs test

- Writes an incrementing pattern to all the software controlled LEDs at a visible rate.

4.3.4 Memory test

4.3.4.1 Address uniqueness test

- Initialize memory to 0's
- Write the address into each memory location.

- Read and verify data from the starting address to the ending address.

- Do the above going from the ending address to the starting address.

4.3.4.2 Memory data test

- Write 0's sequentially from start address to ending address.
- Read sequentially and verify data from start address to ending address.

- Write 1's sequentially from start address to ending address.
- Read sequentially and verify data from start address to ending address.

- Write 5's sequentially from start address to ending address.
- Read sequentially and verify data from start address to ending address.

- Write A's sequentially from start address to ending address.
- Read sequentially and verify data from start address to ending address.

4.3.5 Sliding 1's memory test

- Initialize memory to zero.
- Write a 1 to each memory location.
- Read and verify each memory location.
- Shift the data pattern left one position.
- Do the above until each data cell is tested (32x).

4.3.6 March memory test

- A background of 0x55 is written to memory.
- The data is read and verified starting at the first address.
- The complement is then written to this address.
- This two step read/verify/write procedure is done on each address until the end of memory is reached.
- Then starting at the end of memory each cell is tested and changed back to the original data until the starting address is reached.

4.3.7 Memory refresh test

- Set memory to a known pattern
- Execute wait loop from cache memory.
- Wait in cache loop for 500 ms.
- Verify data in memory has not changed.

4.3.8 Map I/O test

4.3.8.1 Test each map (byte wide) register as RAM

- Write/Read/Verify each register for a pattern of 0's F's, A's 5's.
- Verify we can correctly walk a 1 through each register.
- Address uniqueness test.

4.3.8.2 Map I/O address test

- Disable command sequencer.
- Set I/O map to point to ourselves.
- Write data to ourselves via CSS bus.
- Verify data and address is correct in input buffer.

4.3.9 Register test

- Verify each appropriate bit can be set and cleared in each appropriate register.

4.3.10 Interrupt logic test

- Read error status register and make sure there are no interrupts pending.
- Set bit 1 in the interrupt control register.
- Verify the appropriate interrupt was generated.
- Verify the bit can be cleared.

- Do the above for all 7 interrupts.

4.3.11 Error status register test

4.3.11.1 NACK_SENT, NO_ACK, TX_ACK, PERROR, bit test

- Set the I/O map to point to ourselves.
- Verify the error status register has no errors bits set.
- Set appropriate CSS_PARITY bit in control register (bad CSS parity).
- Try to write to ourselves via CSS bus.
- Verify a ERROR_INTERRUPT is generated.
- Verify the NACK_SENT bit is set in error status register.
- Verify the NO_ACK bit is set in the error status register.
- Verify the TX_NACK bit is set in the error status register.
- Verify the PERROR bit is set in the error status register.

4.3.11.2 68030 time out bit test

- Disable command sequencer.
- Execute a read to ourselves via CSS bus.
- Verify the 68K_TIMEOUT bit is set in the error status register.

4.3.12 Memory parity test

4.3.12.1 Parity checker

- Walk a 1 through data bits writing data to memory.
- Read memory
- Verify no parity ERROR INTERRUPTS are generated.

4.3.12.2 Parity generator

- Set the appropriate WRONG_RAM_PARITY bit in the control register.
- Write data to memory.
- Read data.
- Verify an ERROR INTERRUPT is generated, and the PERROR bit is set.

- Do above for all 4 bytes of data.

4.3.13 CSS bus interface test

4.3.13.1 Input/Output buffer

- Set up I/O map to point to ourselves.
- Disable command sequencer.
- Write a unique data pattern to ourselves via CSS bus. (logical addresses)
- Verify data is correct via pipeline diagnostic registers.

4.3.13.2 Command sequencer

- Set up I/O map to point to ourselves.
- Initialize memory to a unique data pattern (addresses).
- Read data to ourselves via CSS bus. (logical addresses)
- Verify data is correct.

4.3.14 Command sequencer count test

- Clear both the input and output pipelines.
- Disable the command sequencer.
- Issue a write command to ourselves via the CSS bus.
- Verify command count is correct.

- Do above until output buffer is full (3 write commands).
- Issue the FLUSH_PIPELINE command for both the input and output pipeline.
- Verify pipelines are cleared.

4.3.15 Burst read test

- Set up memory as cacheable data.
- Flush cache.
- Write a data to memory.
- Read the data via CSS bus.
- Verify data in registers is same as memory.
- Do above with and without parity errors.

4.3.16 CSS bus parity test

- Set to I/O map to point to ourselves.
- Set the appropriate CSS_PARITY bit in the control register.
- Send a command to ourselves.
- Verify an ERROR_INTERRUPT was generated.
- Verify the NACK_SENT bit is set in the error register.

- Verify we can do the above for each CSS_PARITY bit position.
- Do the above walking a 1 through data bits and address bits.
- Do the above with and without parity errors.

4.3.17 Display Status Register

- Read and display the IOPM status register.
- Read and display board slot ID

4.3.18 Misaligned local memory accesses

- Initialize memory block to known data pattern.
- Write complement of data to an even address.
- Verify data is correct.

- Reinitialize memory block.
- Write complement of data to an odd address.
- Verify data is correct.

- Do the above for byte, word, and long word data.

4.3.19 Misaligned memory accesses via CSS bus

- Initialize memory block to known data pattern.
- Write complement of data to an even address.
- Verify data is correct.
- Reinitialize memory block.
- Write complement of data to an odd address.
- Verify data is correct.

- Do the above for byte, word, and long word data.

4.3.20 Bus error exception

- Write bad parity into local memory.
- Read via CSS bus.
- Verify a bus error exception was generated.

4.3.21 Device board interface

- Read the device board ID
- If this device board has an EPROM installed:
    - Checksum the device board EPROM.
    - Download device board diagnostics from device board EPROM.
    - Execute device board diagnostics.

- If no EPROM is installed:
    - The IOPM will request the device board diagnostics from the service module.

5. Manufacturing Diagnostics

The manufacturing diagnostics will be a copy of the engineering verification diagnostics. These will include scope loops and the individual tests described in the Engineering verification section.

The diagnostics for manufacturing will also contain the scope loops diagnostics provided for the initial board bring-up.