



**Nikki**  
**Series 300 Workstation I/O ASIC**  
**SCSI, High Speed HPIB**  
**RS-232, Low Speed HPIB**  
**and**  
**Bi-directional Parallel Printer Interface**  
**(Centronics)**

**External Reference Specification**  
Don Soltis

Hewlett-Packard Co.  
Fort Collins Systems Division  
Workstation Systems Lab  
4-12-90



## CONTENTS

1.	General Description . . . . .	7
1.1	Features . . . . .	7
1.2	Target Application . . . . .	9
1.3	External Support Logic Requirements . . . . .	11
1.3.1	Interfacing to Fujitsu 87030/33 (SCSI) . . . . .	11
1.3.2	Interfacing to Medusa (HS-HPIB) . . . . .	11
1.3.3	Interfacing to Texas Instruments TMS9914 (LS-HPIB) . . . . .	11
1.3.4	Interfacing to National Semiconductor's NS8250/NS16550 (RS-232) . . . . .	11
1.3.5	Interfacing to Centronics . . . . .	11
2.	Hardware ERS and Theory of Operation . . . . .	11
2.1	Architecture and Functional Descriptions . . . . .	12
2.1.1	Hierarchy map . . . . .	13
2.1.2	SCSI Hardware ERS . . . . .	15
2.1.2.1	DIO to FIFO machine (nkDMA) . . . . .	15
2.1.2.2	Fujitsu 87033 to FIFO machine (SCSI <sub>idma</sub> ) . . . . .	16
2.1.2.3	SCSI DIO Register Access Machine and Registers (SCSI <sub>idioSM</sub> ) . . . . .	17
2.1.3	HS-HPIB Hardware ERS . . . . .	17
2.1.3.1	HS-HPIB DIO to FIFO Machine (nkDMA) . . . . .	18
2.1.3.2	Medusa to FIFO State Machine (HPIB <sub>dma</sub> ) . . . . .	18
2.1.3.3	HS-HPIB Register Access State Machine (HPIB <sub>dioSM</sub> ) . . . . .	18
2.1.4	LS-HPIB Hardware ERS . . . . .	19
2.1.4.1	LS-HPIB Control State Machine (lsH <sub>pibSM</sub> ) . . . . .	19
2.1.5	RS-232 Hardware ERS . . . . .	20
2.1.5.1	RS-232 Register Access State Machine (rs232SM) . . . . .	20
2.1.6	Centronics Hardware ERS . . . . .	20
2.1.6.1	Centronics DIO to FIFO DMA State Machine (cnDMA <sub>sm</sub> ) . . . . .	21
2.1.6.2	Centronics Back End (FIFO to Centronics) State Machine (cnBE <sub>sm</sub> ) . . . . .	21
2.1.6.3	Centronics Register Access State Machine (rs232SM) . . . . .	22
2.1.7	Clock generation ERS . . . . .	22
2.1.8	Scan Chain Testing ERS . . . . .	22
2.2	Pin List and Signal Descriptions . . . . .	24
2.2.1	Signal Descriptions . . . . .	29
2.3	DC Electrical Characteristics . . . . .	40
2.4	AC Electrical Specifications . . . . .	45
2.5	Timing Diagrams . . . . .	46
2.6	Packaging . . . . .	69
2.7	Mechanical Characteristics . . . . .	69
2.8	Test Considerations . . . . .	69
3.	Software ERS and Theory of Operation . . . . .	70
3.1	Software Interface . . . . .	70
3.1.1	SCSI . . . . .	71
3.1.1.1	SCSI ID and Reset Register (1) . . . . .	71
3.1.1.2	SCSI Status and Control Register (3) . . . . .	72

3.1.1.3	SCSI Wraparound and Loopback Data (5)	72
3.1.1.4	SCSI Flush and Hardware Configuration register (7)	73
3.1.1.5	SCSI Packer and Misc Status Register (9)	73
3.1.1.6	SCSI Packer Byte 0 Register (B)	74
3.1.1.7	SCSI Packer Byte 1 Register (D)	74
3.1.1.8	SCSI Packer Byte 2 Register (F)	74
3.1.1.9	SCSI Byte DMA Register (0x10)	74
3.1.1.10	SCSI Hidden Register 1	75
3.1.1.11	SCSI Hidden Register 3	75
3.1.2	HS-HPIB	76
3.1.2.1	HS-HPIB ID and Reset register (1)	77
3.1.2.2	HS-HPIB Status and Control register (3)	77
3.1.2.3	HS-HPIB EOI Control (5)	77
3.1.2.4	HS-HPIB DMA status (7)	77
3.1.2.5	HS-HPIB Hidden Register 1	78
3.1.3	LS-HPIB	79
3.1.3.1	LS-HPIB status/DMA control (3)	79
3.1.3.2	LS-HPIB auxillary status (5)	79
3.1.3.3	LS-HPIB Parallel Poll Status and Control (7)	79
3.1.3.4	LS-HPIB Parallel Poll Mask (9)	80
3.1.3.5	LS-HPIB Hidden Register 1	80
3.1.4	RS-232	81
3.1.4.1	RS-232 ID/Reset Register (1)	81
3.1.4.2	RS-232 Status/Control Register (3)	81
3.1.5	Centronics Software ERS	83
3.1.5.1	Centronics ID/Reset Register (1)	83
3.1.5.2	Centronics Interface Control/Status Register (3)	83
3.1.5.3	Centronics Device Status Register (5)	84
3.1.5.4	Centronics Device Control Register (7)	84
3.1.5.5	Centronics Interrupt Control/Status Register (9)	84
3.1.5.6	Centronics FIFO Data Register (11)	85
3.2	Software Performance Specifications and Considerations	86
3.2.1	SCSI	86
3.2.2	HS-HPIB	86
3.2.3	LS-HPIB	86
3.2.4	RS-232	86
3.2.5	Centronics	86
4.	Errata	88
5.	State Machine Algorithms	89
5.1	SCSI/HS-HPIB DIO-FIFO DMA Machine	89
5.2	SCSI Register Access Machine	92
5.3	SCSI FIFO-87033 DMA Machine	98
5.4	HS-HPIB Register Access Machine	104
5.5	HS-HPIB FIFO-Medusa DMA Machine	107
5.6	LS-HPIB Register Access, DMA, and Parallel poll Machine	114
5.7	RS-232 Register Access Machine	121
5.8	Centronics Register Access Machine	129
5.9	Centronics FIFO-CENTRONICS (back end) Machine	134
5.10	Centronics DIO-FIFO DMA Machine	139

LIST OF FIGURES

Figure 1. Block Diagram . . . . .	10
Figure 2. Clock Timing . . . . .	46
Figure 3. DIO Read Cycle Timing . . . . .	47
Figure 4. DIO Write Cycle Timing . . . . .	48
Figure 5. DIO Inbound DMA Cycle Timing . . . . .	49
Figure 6. DIO Outbound DMA Cycle Timing . . . . .	50
Figure 7. SCSI Read Cycle Timing . . . . .	51
Figure 8. SCSI Write Cycle Timing . . . . .	52
Figure 9. SCSI-FIFO Inbound DMA Cycle Timing . . . . .	53
Figure 10. SCSI-FIFO Outbound DMA Cycle Timing . . . . .	54
Figure 11. HS-HPIB Read Cycle Timing . . . . .	55
Figure 12. HS-HPIB Write Timing . . . . .	56
Figure 13. HS-HPIB Inbound DMA Cycle Timing . . . . .	57
Figure 14. HS-HPIB Outbound DMA Cycle Timing . . . . .	58
Figure 15. LS-HPIB Read Cycle Timing . . . . .	59
Figure 16. LS-HPIB Write Cycle Timing . . . . .	60
Figure 17. LS-HPIB Inbound DMA Cycle Timing . . . . .	61
Figure 18. LS-HPIB Outbound DMA Cycle Timing . . . . .	62
Figure 19. LS-HPIB Parallel Poll Cycle Timing . . . . .	63
Figure 20. RS-232 Read Cycle Timing . . . . .	64
Figure 21. RS-232 Read Cycle Timing . . . . .	65
Figure 22. Centronics Write Register 7 Cycle Timing . . . . .	66
Figure 23. Centronics Back-end Inbound Cycle Timing . . . . .	67
Figure 24. Centronics Back-end Outbound Cycle Timing . . . . .	68

LIST OF TABLES

TABLE 1. Pin List . . . . .	25
TABLE 2. DIO signals . . . . .	29
TABLE 3. Clock signals . . . . .	31
TABLE 4. SCSI support signals . . . . .	32
TABLE 5. High Speed HPIB Support Signals . . . . .	34
TABLE 6. Low Speed HPIB Support Signals . . . . .	35
TABLE 7. Internal RS-232 support signals . . . . .	36
TABLE 8. Centronics Support Signals . . . . .	37
TABLE 9. Absolute Maximum Ratings . . . . .	39
TABLE 10. Input Protection . . . . .	39
TABLE 11. Electrical Characteristics Over Operating Range . . . . .	39
TABLE 12. Select code register map . . . . .	70
TABLE 13. SCSI Standard Register Set . . . . .	71
TABLE 14. SCSI Hidden Register Set . . . . .	75
TABLE 15. HS-HPIB Register Set (98625B compatible) . . . . .	76
TABLE 16. HS-HPIB Register Set (98625C) . . . . .	76
TABLE 17. HS-HPIB Hidden Register Set . . . . .	78
TABLE 18. LS-HPIB Standard Register Set . . . . .	79
TABLE 19. LS-HPIB Hidden Register Set . . . . .	80
TABLE 20. RS-232 Standard Register Set . . . . .	81
TABLE 21. RS-232 Hidden Register Set . . . . .	82
TABLE 22. Centronics Standard Register Set . . . . .	83
TABLE 23. Centronics Hidden Register Set . . . . .	85

## 1. General Description

This ASIC contains the circuitry necessary to implement a SCSI (Small Computer Systems Interface) and/or High Speed HPIB (HS-HPIB) disc interface, and Low Speed HPIB (LS-HPIB), RS-232 serial, and bi-directional parallel (Centronics) interfaces. These interfaces are software compatible with the following previous products: 98265A SCSI, 98625B HS-HPIB, 98562 System interface built in HPIB and RS-232 (98644). The bi-directional parallel interface has no predecessor, and is for interfacing to HP LaserJet and ScanJet type peripherals as well as most "Centronics" type printers.

In addition, several enhancements were made to the SCSI, HS-HPIB and RS-232 interfaces; and all of the interfaces utilize software "switches" for configuration.

This ASIC (Nikki) implements the glue and control logic to interface 4 VLSI I/O controller chips to a DIO-1 or DIO-II bus. These are the Fujitsu MB87030 or MB87033 SCSI Protocol Controller (SPC), for SCSI, the Medusa (1LT1-9001) HPIB controller for HS-HPIB, and the Texas Instruments TMS9914 GPIB controller for LS-HPIB. The bi-directional parallel printer interface (Centronics) is implemented almost entirely within Nikki.

Because of pin limitations and perceived product mix, the SCSI interface is mutually exclusive with the HS-HPIB interface. These two interfaces share several pins and an internal DMA data FIFO. Although both interfaces supporting circuitry can be present at one time, Nikki will only configure one or the other, and this configuration can only be changed by asserting a power-up reset signal.

### 1.1 Features

- SCSI
  - Software compatible with current SCSI interfaces (98265A)
  - Circuitry needed to interface to Fujitsu MB87030 or MB87033 SCSI controller
  - DIO Registers and DMA interface
  - † 32 byte FIFO for SCSI DMA transfers
  - † Byte DMA and packer registers for tape drive support
  - † Ability to switch between 8 and 10 MHz Fujitsu frequency "on-the-fly"
  - † Higher DMA performance than 98265A
- HS-HPIB
  - Software compatible with 98625B High Speed HPIB Disc interface
  - Circuitry needed to interface to Medusa (1LT1-9001) HPIB controller
  - DIO Registers and DMA interface
  - 32 byte FIFO for HPIB DMA transfers
  - † 32-bit HPIB DMA transfers in addition to 8- and 16-bit supported by 98625B
  - † Higher DMA performance than 98625B



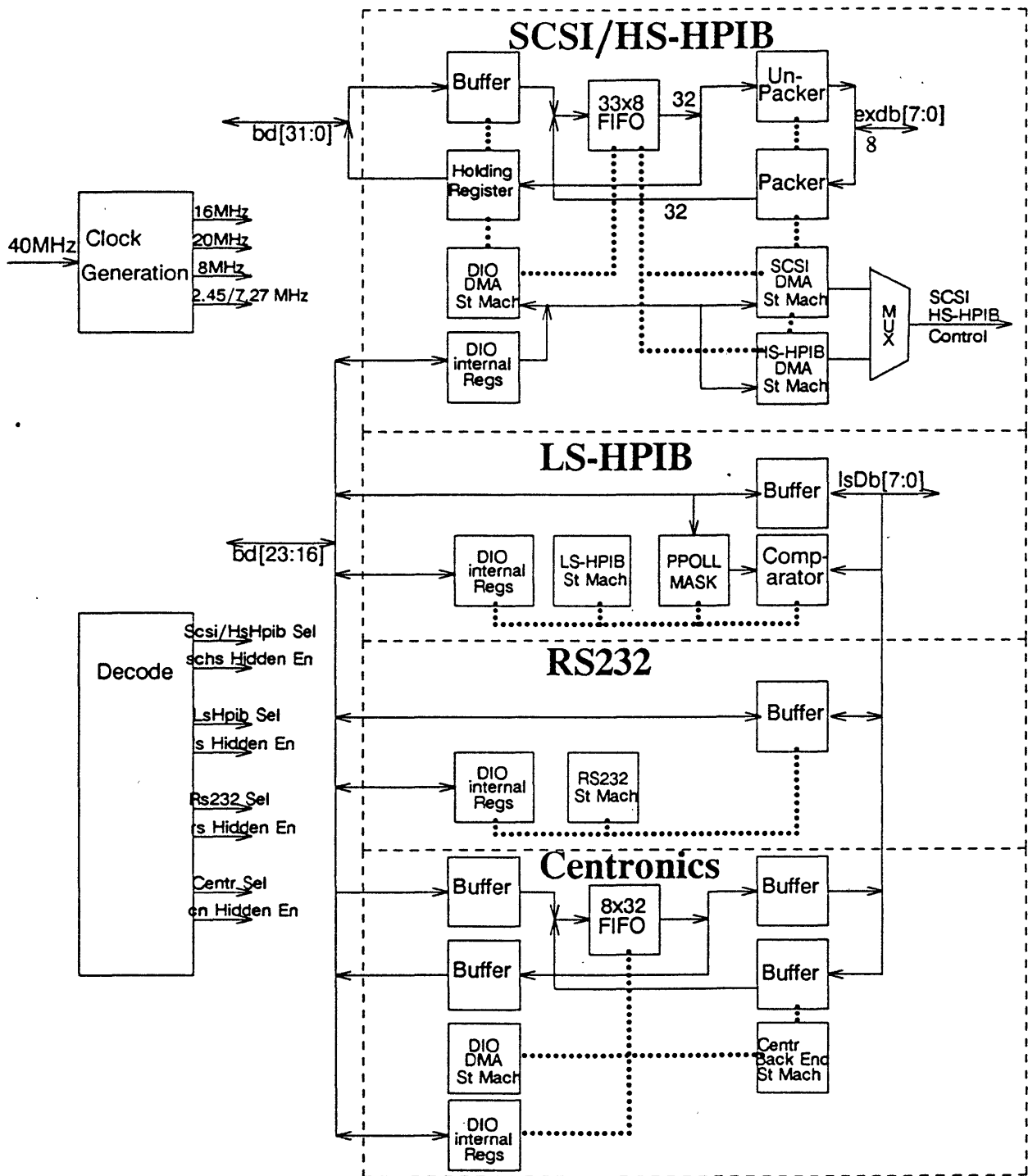
- **LS-HPIB**
  - Software compatible with 98625 System Interface Internal HPIB
  - Circuitry needed to interface to TMS9914 GPIB controller
  - DIO Registers and DMA interface
  - Hardware Parallel Poll circuitry
- **RS-232**
  - Software compatible with the 98625 System Interface internal RS-232 (98644)
  - Circuitry needed to interface to National 16550 or 8250 UART
  - DIO Registers
  - 2.4576 MHz baud rate generator
  - † 7.2727 MHz baud rate generator for high speed communications
- **Centronics**
  - LaserJet and ScanJet family compatible
  - 300kbyte per second outbound throughput
  - 700kbyte per second inbound throughput
  - Hardware controlled transfers
  - 32 byte FIFO for either DMA or programmed I/O
  - Byte wide DMA interface
  - Supports mixed DMA and programmed transfers
  - BUSY and NACK or BUSY-only hardware handshake
  - 7 maskable interrupting conditions
  - FIFO disable
- **General enhancements**
  - Software programmable select code.
  - Software programmable "switch" settings
  - Up to 4 Nikki's in one system
  - Individual module disables (Turn off I/O interface)

† *New features*

## *1.2 Target Application*

Nikki is an ASIC for use in HP9000 S300 IO PC boards to implement SCSI or HP-IB disc interfaces, Low Speed HP-IB, RS-232, and Centronics I/O interfaces. This chip is used to reduce system cost by reducing PC board area and components required to implement the desired functions. Power needs are also reduced.

Each interface can be independently disabled so the I/O subsection being implemented can be easily customized. One could use a Nikki to implement just a DIO-1 Centronics interface, or 4 Nikki's as part of the core I/O for a server with several disc/tape interfaces, 4 RS-232 interfaces, and 4 Centronics printer/scanner interfaces.



**Figure 1. Block Diagram**

### *1.3 External Support Logic Requirements*

All pads have n-channel output transistors; however, all but 4 pads have a p-type ESD protection diode. Only the 4 pads, *Ndtack16*, *Ndmrq[1]*, *Ndmrq[0]*, and *Ndmrdy* can be attached directly to a bus where there is the possibility that those bus signals may drive Nikki while power has not yet been applied.

The state machine timing is designed to meet DIO protocol when Nikki is completely buffered from the external bus. The buffers to be used in an actual circuit are assumed to be of the 74ALS or 74LS family.

#### *1.3.1 Interfacing to Fujitsu 87030/33 (SCSI)*

Reference Document A-98574-66510-9

#### *1.3.2 Interfacing to Medusa (HS-HPIB)*

Reference Document A-98574-66510-9

#### *1.3.3 Interfacing to Texas Instruments TMS9914 (LS-HPIB)*

Reference Document A-98574-66510-9

#### *1.3.4 Interfacing to National Semiconductor's NS8250/NS16550 (RS-232)*

Reference Document A-98574-66510-9

#### *1.3.5 Interfacing to Centronics*

Reference Document A-98574-66510-9

## **2. Hardware ERS and Theory of Operation**

This section lightly discusses the architecture and design of Nikki. Refer to the DCS schematics and PD (PAL Descriptor Language) source files in the Appendix (The schematics are not part of this document).

Nikki is implemented CMOS34 Standard Cell technology. It is packaged in a 160 pin plastic quad flat pack (PQFP). There are 114 DCS (Design Capture System) circuit pages that make up the schematic for the 18000+ gate Nikki design. The design utilizes a scan chain configuration for testing purposes. An automatic test generator (ATG) was run to generate a scan based test for "stuck at" fault coverage.

The design was completed using DCS v3.0 on a 370 running version 6.2 HP-UX. The entire design data base exists on two DC-300 tapes presumably in Manufacturing Specifications.

## *2.1 Architecture and Functional Descriptions*

The internal design was broken up into several portable modules to ease possible future use. A clock generation block generates internal the clocks and reset signals used by the rest of the chip. There is a scan control block, and a SCSI/HS-HPIB, a LS-HPIB, RS-232, and Centronics block. There is one module that performs powerup configuration, and module address decode. And several small miscellaneous blocks and gate to combine DIO and shared signals. Each of these is discussed below. Also a hierarchy map is shown to assist in following the schematics if necessary.

### 2.1.1 Hierarchy map

NikkiChip: Top level. PADS and Nikki

Nikki: Main Schematic.

Centron: Centronics interface top level

cnBEsm: Centronics back end state machine

spareG: Spare gates block

cnTimer: Centronics 1 microsecond timer

cnTimerEQN: Centronics 1 microsecond logic block

cnBEsmEQN: State machine logic block

cnDIO: Centronics DIO registers and state machine

cnDIOreg: Centronics DIO registers

strz8: Separate input tri-state 8 bit buffer

sssreg8: Separate output, synchronous, scannable 8 bit register

SSDC1F: synchronous, scannable STD34DC1F

SSDP1F: synchronous, scannable STD34DP1F

ltf8: 8 bit latch (STD34LTF)

cnInterrupt: Centronics interrupting condition register

cnEdgeDetect: Centronics Edge detect

cnDIOsm: Centronics DIO register access machine

spareG: Spare gates block

cnDIOsmEQN: Centronics DIO machine logic block

cnDMAsm: Centronics DMA state machine

spareG: Spare gates block

cnDMAsmEQN: Centronics DMA machine logic block

cnFifo: Centronics 32x8 FIFO

cnFifoCnt: Centronics FIFO pointer counter

cnFifoCntEQN: Centronics counter machine logic block

cnFullEmpty: Centronics FIFO full and empty generator

cnFullEmptyEQN: Centronics full/empty logic block

NAND32F: logic block that implements a 32 input NAND

cninvf32: 32 bit inverter (STD34INVF)

cnRegz32x8: 32 deep 8 bit wide tri-state register array

cnFifoRegz8: 8 bit wide tri-state register

cnnsbf8: 8 bit wide buffer (STD34SPBF8)

regz8: Tri-state 8 bit register

trz8: Tri-state 8 bit buffer

Clocks:

ResetCtl: Chip wide reset generation

clk7MHz: 7.272MHz clock generator

clk7MHzEQN: 7.272MHz generator logic block

clkBgen: 2.4576MHz clock generator

GclkBgen: 2.4576MHz generator logic block

ClkDiv2: 20MHz clock generator

a40to16MHz: 16MHz clock generator

Decode: Select code decode and select code registers

KeyBdEn: Keyboard NMI register enable

compare8: 8 bit comparator

SelDecEQN: Select code decode logic block

SelectReg: Select code register, hidden enable

compare8: 8 bit comparator

DecodeEQN: decode logic block

SelectEQN: hidden enable logic block

compare8: 8 bit comparator

iSelectReg: LS-HPIB select code register, hidden enable

compare8: 8 bit comparator

DecodeEQN: decode logic block

SelectEQN: hidden enable logic block

puCNsc: Centronics powerup select code generator



    mux4x6: 4 by 6bit MUX  
puLSsc: LS-HPIB powerupt select code generator  
puRSsc: RS-232 powerup select code generator  
    mux4x6: 4 by 6bit MUX  
puSCHSsc: SCSI/HS-HPIB powerup select code generator  
    mux4x6: 4 by 6bit MUX  
puSel: Power up configuration block  
    hsdcf: Flip flop (ATG hold type)  
    myDelay: delay gates  
    spareG: Spare gates block  
Keep8: 8 bit bus keeper (STD34KEEP)  
LSHPIB: LS-HPIB top level  
    lsHpiBIOreg: LS-HPIB internal DIO registers  
        SSDC1F: Synchronous, scannable flip flop (STD34DC1F)  
        SSDP1F: Synchronous, scannable flip flop (STD34DP1F)  
        strz8: Separate input tri-state 8 bit buffer  
    lsHpiBsm: LS-HPIB machine (REG,DMA,PPOLL)  
        fsdcf: Fast scannable flip-flop (STD34FDC)  
        lsHpiBsmEqn: LS-HPIB machine logic block  
            NAND12F: logic block that implements a 12 input NAND  
        myDelay: Delay gates  
        spareG: Spare gates  
    ltz8: 8 bit tri-state latch (STD34LTZF)  
    ppollmatch: Parallel poll match  
    trz8: tri-state buffer (STD34TRZF)  
PadControl: Pad drive enable/disable  
SCSIhpiB: SCSI/HS-HPIB top level  
    HPIBdio: HS-HPIB DIO registers/machine  
        HPIBdioReg: HS-HPIB internal DIO registers  
            SSDC1F: Synchronous, scannable flip-flop (STD34DC1F)  
            SSDP1F: Synchronous, scannable flip-flop (STD34DP1F)  
            strz8: separate input tri-state 8 bit buffer (STD34TRZF)  
        HPIBdioSM: HS-HPIB DIO register access machine  
            HPIBdioSMeqn: HS-HPIB DIO register access logic block  
            HPIBbrstSM: HS-HPIB 500ns reset machine  
                HPIBbrstSMeqn: 500ns reset logic block  
        spareG: Spare gates  
    ltz8: 8 bit latch (STD34LTZF)  
    trz8: 8 bit buffer (STD34TRZF)  
SCSIdio: SCSI DIO registers/machine  
    ltz8: 8 bit latch (STD34LTZF)  
    SCSIdioReg: SCSI internal DIO registers  
        SSDC1F: Synchronous, scannable flip-flop (STD34DC1F)  
        SSDP1F: Synchronous, scannable flip-flop (STD34DP1F)  
        strz8: Separate input tri-state 8 bit buffer (STD34TRZF)  
    SCSIdioSM: SCSI DIO register access machine  
        SCSIdioSMeqn: SCSI DIO register access logic block  
    myDelay: Delay gates  
    spareG: Spare gates  
    trz8: 8 bit tri-state buffer (STD34TRZF)  
SCSIparity: Parity generator  
myDelay: Delay gates  
nkDMA: SCSI/HS-HPIB FIFO, DMA machines  
    DMAdio: DIO-FIFO DMA machine  
        SSDC1F: Synchronous, scannable flip-flop (STD34DC1F)  
        fsdcf: Fast, scannable flip-flop  
        sDMAdioEQN: DIO-FIFO DMA logic block  
    spareG: Spare gates  
    Fifo: 8 deep 33 bit FIFO  
    fifocntr: FIFO pointer counter

fullempty: FIFO full and empty generator  
 reg8x32  
     nspb32: 32 bit buffer (STD34NSPBF)  
     reg32: 33 bit tri-state register  
 HPIBdma: FIFO-Medusa DMA machine  
     HPIBdmaEQN: FIFO-Medusa DMA logic block  
         NAND11F: logic block that implements a 11 input NAND gate  
         NAND12F: logic block that implements a 12 input NAND gate  
     SSDC1F: Synchronous, scannable flip-flop (STD34DC1F)  
     myDelay: Delay gates  
     spareG: Spare gates  
 SCSIdma: FIFO-87033 DMA machine  
     SCSIdmaEQN: FIFO-87033 DMA logic block  
         NAND11F: logic block that implements NAND11F  
     spareG: Spare gates  
 regz8: 8 bit tri-state register  
 sTimeoutCtr: 32 microsecond counter  
     sTimeoutEQN: 32 microsecond counter logic block  
 trz8: tri-state 8 bit buffer (STD34TRZF)  
 SChsMUX: SCSI/HS-HPIB shared signal muxing  
     SChsMUX8: SCSI/HS-HPIB 8 by 2 to 1 MUX  
 ScanControl: Scan Test control  
 irCombine: Interrupt request combine  
 nkArbiter: lsDb[7:0] data bus arbiter  
     arbiterEQN: lsDb arbiter logic block  
     spareG: Spare gates  
 nkRS232: RS-232 top level logic  
     ltz8: 8 bit tri-state latch (STD34LTZF)  
     nkStRes: 26 microsecond reset counter  
         nkSresEqn: 26 microsecond counter logic block  
     rs232SM: RS-232 DIO register access machine  
         rs232SMeqn: RS-232 DIO register access logic block  
     rs232reg: RS-232 internal DIO registers  
         SSDC1F: Synchronous, scannable flip-flop (STD34DC1F)  
         SSDP1F: Synchronous, scannable flip-flop (STD34DP1F)  
     strz8: Separate input 8 bit tri-state buffer (STD34TRZF)  
     spareG: Spare gates  
     trz8: 8 bit tri-state buffer (STD34TRZF)  
 ocCombine:

## 2.1.2 SCSI Hardware ERS

The SCSI interface is broken up into three state machines, a FIFO, the data path circuitry and the internal registers and hidden register sections. One machine controls accesses to the DIO and hidden registers, one performs DMA transfers between the Fujitsu and the FIFO, and the last performs DMA transfers between the DIO bus and the FIFO. One should refer to the schematic, timing diagram, and PD state machine source during the following discussion.

### 2.1.2.1 DIO to FIFO machine (nkDMA)

The nkDMA machine is common to the SCSI interface and the HS-HPIB interface. This machine transfers data to and from the DIO bus and the FIFO. It does NOT master the DIO bus, but uses the 1TQ4 DMA chip or equivalent to control the transfer and supply main memory addresses. The DMA transfer can be byte-wide (8bits), word-wide (16bits) or longword-wide (32bits). For SCSI only word and longword widths are used. The DMA width specifies the amount of data transferred on each cycle. This machine feeds the



32 bit wide FIFO using only 8 bits if in byte mode, 16 bits if in word mode or the entire 32 bits when in longword mode.

During inbound DMA (writing to main memory) the nkDMA machine is enabled with one of the DE1, DE0 bits in register 3 being set. Once set, it waits for the FIFO to become full or a timer to go off. This machine waits for a FIFO full condition so that a minimum of 8 words/longwords will be transferred for each bus arbitration (request, grant, acknowledge, release) cycle. This increases the available DIO bus bandwidth. However, at the end of a transfer the FIFO may be partially full and will never become full. This is why a timer is used. The timer is enabled when one of the DMA enable bits are set and the machine is currently not requesting a transfer. It cyclically times out at 30-32 microseconds intervals and at worst will effect the transfer speed by less than 10% (8kbytes/4Mbytes/sec = 2ms; 32us/2ms = 1.6%). Since the Fujitsu usually interrupts when it has transferred the last byte into Nikki, this will also cause a timeout.

Once the FIFO is full or a timeout has occurred, the FIFO data is latched into a holding register and *Ndmrq[0 or 1]* (DMA request channel 0 or 1) is asserted. When *Ndmack[0 or 1]* (DMA acknowledge channel 0 or 1) is asserted, the holding register data is driven onto the DIO bus and after some hold time, *Ndmrdy* is asserted indicating that data is valid. If the FIFO is not empty *Ndmrq* is held asserted so that the DMA controller does not relinquish the DIO bus; otherwise, *Ndmrq* is deasserted and the machine waits for FIFO full or timeout and the process repeats. Refer to figure titled *DIO Inbound DMA Cycle Timing*

On outbound DMA (read from main memory) the machine asserts *Ndmrq* whenever the FIFO is empty and continues to assert it until the FIFO will become full for the same bandwidth maximization concerns as before. Once *Ndmrq* has been asserted, both *Ndmack* and *Ndtkall* becoming asserted indicate valid data on the DIO bus. This data is loaded into the FIFO. If *Ndmrq* was deasserted after *Ndmack* was asserted because the FIFO was almost full, then the machine, after completing the current transfer, will again wait for the FIFO to become empty before starting another cycle. Refer to figure titled *DIO Outbound DMA Cycle Timing*

#### 2.1.2.2 Fujitsu 87033 to FIFO machine (SCSI<sub>idma</sub>)

The SCSI<sub>idma</sub> block performs byte wide DMA transfers with the Fujitsu 87033 SCSI controller. During inbound DMA (write to main memory) this state machine loads the data bytes from the Fujitsu into the packer until the DIO DMA width is obtained (this is 4 bytes or 2 bytes for longword or word mode). When the FIFO is not full these bytes are loaded into the FIFO and the cycle repeats. The state machine uses the packer clocks to indicate how full the packer is and where the next byte will go. So, after a *scDmrq* assertion, *scDmrsp* is asserted and when data is valid it is clocked into the proper packer byte. Once 2 or 4 bytes have been accumulated the packer contents are loaded into the FIFO. If the data bytes received from the 87033 are numbered 0, 1, 2, and 3 where 0 is the first byte received, then during longword DMA byte 0 will appear on BD[31:24], byte 1 on BD[23:16], byte 2 on BD[15:8] and byte four on BD[7:0]. During word DMA, byte 0 and 1 appear on the same data lanes, and byte 2 and 3 will be transferred on the next DMA cycle on BD[31:24] and BD[23:16] respectively. Refer to figure titled *SCSI-FIFO Inbound DMA Cycle Timing*.

During outbound DMA (read from main memory) the SCSI<sub>idma</sub> machine unloads a FIFO entry into the unpacker. And enables one byte at a time out to the Fujitsu controller. The byte that was received from the DIO dma cycle on data lane BD[31:24] is sent first and then the byte from BD[23:16]; if this is a longword transfer then the next two bytes will come from data lane BD[15:8] and BD[7:0] respectively. If the transfer is being performed in word mode, the next two bytes will come from the next entry in the FIFO on data lanes BD[31:24] and BD[23:16] respectively. Refer to figure titled *SCSI-FIFO Outbound DMA Cycle Timing*.

The SCSIIdma machine has priority over the DIO register access machine (SCSIIdioSM) for the use of the shared exDb data bus. Therefore, it relies on the fact that the SCSIIdioSM machine will assert *NIOActive* and then wait for *NDMAActive* to deassert before any DIO accesses are performed. If the SCSIIdma machine sees that *NIOActive* is asserted, it will deassert *NDMAActive* between cycles and wait for *NIOActive* to deassert before continuing.

### 2.1.2.3 SCSI DIO Register Access Machine and Registers (SCSIIdioSM)

The SCSIIdioSM machine controls accesses between DIO read/writes and internal/external/hidden registers. The internal and hidden registers are FLIP-FLOPS and buffers located in SCSIIdioReg. There are three types of external registers. Those in the Fujitsu, the wraparound and loopback test registers, and the byte DMA access "register".

The machine starts an access with a "chip select" from the decode block. It asserts *NIOActive* for a minimum of two states before checking *NDMAActive*. This is how it establishes ownership of the *exDb* data bus. Once *NDMAActive* is deasserted, a cycle is performed.

For an internal register read a buffer enable signal is asserted, the  $BD[23:16]$  pad output enables are asserted. This state is held for several clocks to allow for setup times and then *Ndtack16* is asserted. Then the machine waits for the "chip select" to be deasserted ( corresponds to *Nbas24*), and deasserts *Ndtack16*. *Ndtack16* is also asynchronously gated with *Nbas24* at the top level. The machine deasserts *NIOActive* and returns to the idle state.

For an internal register write, the *NIOActive/NDMAActive* handshake is performed and then after some data setup time, a write enable signal is asserted. This signal gates a MUX to a synchronous FLIP-FLOP in SCSIIdioReg to accept data off of the input DIO data bus. At the same time *Ndtack16* is asserted. The same algorithm as in the internal read access ensues until *Nbas24* deassertion. The only time that data is written to the flip flop is the rising clock edge directly preceding the assertion of *Ndtack16*.

An external register read occurs very similarly to that of an internal register read, except *sread* is asserted which asserts *NspcCs* on a falling edge of the clock, then asserts *NscRead* on the next rising edge to ensure *NspcCs* to *NscRead* setup time. Then waits three states with a latch in the transparent mode feeding *exDb[7:0]* to  $BD[23:16]$ , and *sread* asserted. Once enough data settling and setup time has passed, *sread* is deasserted and that *exDb[7:0]* data is latched internally. The deassertion of *sread* deasserts *NscRead* on a falling clock edge, and deasserts *NspcCs* on the next rising clock edge. This guarantees *NspcCs* hold time after *NscRead* deassertion. Refer to figure titled *SCSI Read Cycle Timing*.

An external register write uses the same mechanism via *swrite* to generate a *NspcCs* assertion followed by a *NscWrite* assertion one half clock later. *exDb[7:0]* is driven at the start of the cycle. The machine holds for three clocks and releases *swrite* which deasserts *NscWrite* on the falling clock edge followed by *NspcCs* deassertion on the next rising edge. Data (*exDb*) is valid until the end of the cycle (*Nbas24* or *Nlds*) Refer to figure titled *SCSI Write Cycle Timing*.

### 2.1.3 HS-HPIB Hardware ERS

The HS-HPIB interface is broken up into three state machines, a FIFO, the data path circuitry and the internal registers and hidden register sections. One machine controls accesses to the DIO and hidden registers, one performs DMA transfers between the Medusa and the FIFO, and the last performs DMA transfers between the DIO bus and the FIFO. One should refer to the schematic, timing diagram, and PD

state machine source during the following discussion.

### 2.1.3.1 HS-HPIB DIO to FIFO Machine (*nkDMA*)

This is shared with SCSI, see above section.

### 2.1.3.2 Medusa to FIFO State Machine (*HPIBdma*)

The Medusa to FIFO machine works very similarly to the SCSIdma. The HPIBdma block performs byte wide DMA transfers with the Medusa HPIB controller. During inbound DMA (write to main memory) this state machine loads the data bytes from Medusa into the packer until the DIO DMA width is obtained (this is 1 byte, 2 bytes, or 4 bytes for byte, word, or longword mode). When the FIFO is not full these bytes are loaded into the FIFO and the cycle repeats. The state machine uses the packer clocks to indicate how full the packer is and where the next byte will go. If *hsB[0]* is high during a transfer it indicates that this is the last transfer the Medusa will perform. The machine will set an interrupt (EOI) and, if the packer is not full, the odd bits and flush the packer into the FIFO so that the data will get transferred via DMA.

So, after a *NhsDmrq* assertion, a Medusa register 2 read cycle is performed. When data is valid on the *exDb[7:0]* bus it is clocked into the proper packer byte. Once 1, 2 or 4 bytes have been accumulated the packer contents are loaded into the FIFO. If the data bytes received from the Medusa are numbered 0, 1, 2, and 3 where 0 is the first byte received, then during byte DMA the bytes will appear on BD[23:16]. During word DMA, byte 0 and 1 appear on the BD[31:24] and BD[23:16] respectively, and byte 2 and 3 will be transferred on the next DMA cycle on the same data lanes. During longword DMA byte 0 will appear on BD[31:24], byte 1 on BD[23:16], byte 2 on BD[15:8] and byte four on BD[7:0]. Refer to figure titled *Medusa-FIFO Inbound DMA Cycle Timing*.

During outbound DMA (read from main memory) the HPIBdma machine unloads a FIFO entry into the unpacker. And enables one byte at a time out to the Medusa HPIB controller. The byte that was received from the DIO dma cycle on data lane BD[23:16] is sent during byte DMA. During word DMA the byte on data lane BD[31:24] is sent first and then the byte from BD[23:16]; if this is a longword transfer then the next two bytes will come from data lane BD[15:8] and BD[7:0] respectively. Refer to figure titled *Medusa-FIFO Outbound DMA Cycle Timing*.

The HPIBdma machine has priority over the DIO access machine (HPIBdioSM) for the use of the shared *exDb* data bus. Therefore, it relies on the fact that the HPIBdioSM machine will assert *NIOActive* and then wait two cycles for *NDMAActive* to deassert before any DIO accesses are performed. If the HPIBdma machine sees that *NIOActive* is asserted, it will deassert *NDMAActive* between cycles and wait for *NIOActive* to deassert before continuing.

### 2.1.3.3 HS-HPIB Register Access State Machine (*HPIBdioSM*)

The HPIBdioSM machine controls accesses between DIO read/writes and internal/external/hidden registers. The internal and hidden registers are FLIP-FLOPS and buffers located in HPIBdioReg. The external registers are those in the Medusa HPIB controller.

The machine starts an access with a "chip select" from the decode block. It asserts *NIOActive* for a minimum of two states before checking *NDMAActive*. This is how it establishes ownership of the *exDb* data bus. Once *NDMAActive* is deasserted, a cycle is performed.

For an internal register read a buffer enable signal is asserted, the BD[23:16] pad output enables are asserted. This state is held for several clocks to allow for setup times and then *Ndtack16* is asserted. Then the machine waits for the "chip select" to be deasserted (corresponds to *Nbas24*), and deasserts *Ndtack16*. *Ndtack16* is also asynchronously gated with *Nbas24* at the top level. The machine deasserts *NIOactive* and returns to the idle state.

For an internal register write, the *NIOactive/NDMAactive* handshake is performed and then after some data setup time, a write enable signal is asserted. This signal gates a MUX to a synchronous FLIP-FLOP in *HPIBdioReg* to accept data off of the input DIO data bus. At the same time *Ndtack16* is asserted. The same algorithm as in the internal read access ensues until *Nbas24* deassertion. The only time that data is written to the flip flop is the rising clock edge directly preceding the assertion of *Ndtack16*.

A write to register 1 will assert *hsPon* for 500 nanoseconds.

An external register read occurs very similarly to that of an internal register read, except the Medusa control signals are asserted and when data is valid on the *exDb[7:0]* data bus, it is latched into an internal latch and *Ndtack16* is asserted. The Medusa cycle is completed by the falling edge of *Ndtack16*. Data is driven on the *BD[23:16]* bus from the internal latch until the deassertion of the internal "chip select" signal. Refer to figure titled *HS-HPIB Read Cycle Timing*.

To perform an external register write the machine asserts the Medusa control signals and data, and when the Medusa accepts data off of the *exDb* bus, *Ndtack16* is asserted and the Medusa cycle is completed. The machine then waits for the deassertion of the internal "chip select" before returning to the idle state. Refer to figure titled *HS-HPIB Write Cycle Timing*.

#### 2.1.4 LS-HPIB Hardware ERS

The LS-HPIB machine controls DMA cycles; internal, hidden, and external register read and writes; and parallel poll operations. A parallel poll operation is basically an extended TMS9914 register 6 read where the data bits are compared to a mask (internal register 9) and when a match occurs the parallel poll is terminated and an interrupt is set. Since a parallel poll requires the use of the share *lsDb* data bus, a parallel poll is canceled any time Centronics, RS-232 needs the bus or a LS-hpib register access occurs. Once the access is completed the parallel poll resumes until a match occurs or the poll is disabled. One should refer to the schematic, timing diagram, and PD state machine source during the following discussion.

##### 2.1.4.1 LS-HPIB Control State Machine (*lsHpibSM*)

A DMA cycle is performed when the DMA enable bit is set, the TMS9914 asserts *NlsAccRq*, and *Ndmack[0]* is asserted. If *Nwrite* is asserted, an inbound cycle is performed; otherwise, an outbound cycle occurs. During an inbound cycle, the TMS9914 control signals are asserted according to a DMA access and when the data is valid it is latched internally to Nikki. The TMS9914 cycle is completed and *Ndmrdy* is asserted to indicate that *BD[23:16]* is valid. Then the machine waits for the deassertion of *Ndmack[0]* before returning to the idle state. Refer to figure titled *LS-HPIB Inbound DMA Cycle Timing*.

An outbound DMA cycle is initiated in the same manner as the inbound case. During an outbound DMA cycle an outbound DMA TMS9914 cycle is started and when *Ndtkall* is asserted, indicating valid *BD[23:16]* data, the write is completed and *Ndmrdy* is asserted indicating the data has been accepted. Again, the machine waits for the deassertion of *Ndmack[0]* before returning to the idle state. Refer to figure titled *LS-*

### *HPIB Outbound DMA Cycle Timing.*

An internal register read basically gates data on the  $BD[23:16]$  data bus allows some data setup time and then asserts  $Ndtack16$  to indicate valid data. The machine waits for  $Nbas24$  to deassert before the machine returns to the idle state. During an external register read, a TMS9914 read cycle is performed, and when data is valid it is latched internally. The TMS9914 cycle is completed and  $Ndtack16$  is asserted. Refer to figure titled *LS-HPIB Read Cycle Timing*.

An internal register write enables a set of synchronous FLIP-FLOPS to accept data off of the  $BD[23:16]$  bus.  $Ndtack16$  is then asserted to indicate that data was accepted and the machine waits for the deassertion of  $Nbas24$  to end the cycle. For an external register write, a TMS9914 write cycle is performed and the data on  $BD[23:16]$  is redirected to  $lsDb[7:0]$  and when the TMS9914 cycle is completed,  $Ndtack16$  is asserted. Refer to figure titled *LS-HPIB Write Cycle Timing*.

When parallel polling is enabled, a TMS9914 read cycle is performed. The read is held mid-cycle and the data on  $lsDb[7:0]$  is repeatedly sampled and compared to a mask value until a match occurs, or someone else (Centronics, RS-232, LS-HPIB) wants to use the  $lsDb[7:0]$  data bus. If a match occurs, the machine sets an interrupt flag, completes the TMS9914 read cycle and disables further parallel polls. If access to the  $lsDb$  bus is requested by another device, the TMS9914 cycle is completed and the  $lsDb$  bus is given up. When the device releases the resource, the parallel poll operation is once again started. Refer to figure titled *LS-HPIB Parallel Poll Cycle Timing*.

## *2.1.5 RS-232 Hardware ERS*

The RS-232 hardware performs only internal, hidden, and external register reads and writes. One should refer to the schematic, timing diagram, and PD state machine source during the following discussion.

### *2.1.5.1 RS-232 Register Access State Machine (rs232SM)*

An internal register read basically gates data on the  $BD[23:16]$  data bus allows some data setup time and then asserts  $Ndtack16$  to indicate valid data. The machine waits for  $Nbas24$  to deassert before the machine returns to the idle state. During an external register read, a NS16550 (NS8250) read cycle is performed, and when data is valid it is latched internally. The NS16550 (NS8250) cycle is completed and  $Ndtack16$  is asserted. Refer to figure titled *RS-232 Read Cycle Timing*.

An internal register write enables a set of synchronous FLIP-FLOPS to accept data off of the  $BD[23:16]$  bus.  $Ndtack16$  is then asserted to indicate that data was accepted and the machine waits for the deassertion of  $Nbas24$  to end the cycle. For an external register write, a NS16550 (NS8250) write cycle is performed and the data on  $BD[23:16]$  is redirected to  $lsDb[7:0]$  and when the NS16550 (NS8250) cycle is completed,  $Ndtack16$  is asserted. Refer to figure titled *RS-232 Write Cycle Timing*.

A write to register 1 will assert  $rsReset$  for 26microseconds.

## *2.1.6 Centronics Hardware ERS*

The Centronics hardware consists of a DIO-DMA machine for transferring data via DMA into the 32 byte deep FIFO; a DIO register access machine that reads and writes to the FIFO, reads Centronics input signal levels, sets Centronics output signals, and reads and writes various internal control registers; and a Back end

machine which interfaces the Centronics bus to the FIFO. One should refer to the schematic, timing diagram, and PD state machine source during the following discussion.

#### *2.1.6.1 Centronics DIO to FIFO DMA State Machine (cnDMA<sub>sm</sub>)*

This machine transfers data to and from the DIO bus and the Centronics FIFO. The DMA transfer can only be byte-wide (8bit).

During inbound DMA (writing to main memory) the cnDMA<sub>sm</sub> machine is enabled with one of the DE1, DE0 bits in register 3 being set. Once set, it waits for the FIFO to become not empty. The FIFO data is then latched into a holding register and *Ndmrq[0 or 1]* is asserted. When *Ndmack[0 or 1]* is asserted, the holding register data is driven onto the DIO bus and after some hold time, *Ndmrdy* is asserted indicating that data is valid. *Ndmrdy* and data are held until *Ndmack* is deasserted and then the process repeats. Refer to figure titled *DIO Inbound DMA Cycle Timing*

On outbound DMA (read from main memory) the machine asserts *Ndmrq* whenever the FIFO is not full. Once *Ndmrq* has been asserted, both *Ndmack* and *Natkall* becoming asserted indicate valid data on the DIO bus. This data is loaded into the FIFO. The machine asserts *Ndmrdy* to indicate that the data was taken, and waits for the deassertion of *Ndmack* before returning to the idle state and repeating the cycle. Refer to figure titled *DIO Outbound DMA Cycle Timing*

#### *2.1.6.2 Centronics Back End (FIFO to Centronics) State Machine (cnBE<sub>sm</sub>)*

This machine basically transfers data out of the FIFO to a Centronics peripheral or from the peripheral into the FIFO.

During inbound transfers, *NcnOut* is asserted by setting the direction bit in register 3, *NcnStrobe* becomes an input and *cnBusy* becomes an output. If the FIFO is not empty *cnBusy* is deasserted and the machine waits for *NcnStrobe* to be asserted by the peripheral indicating that data is valid. A bus request is then asserted to gain control over the *lsDb* bus and *cnDclk* is asserted to indicate that the Centronics data bus should be driven on the *lsDb* data bus. This data is clocked into the FIFO and *cnBusy* is asserted to indicate that data was accepted. The machine then waits for *NcnStrobe* to be deasserted indicating the end of the cycle so that *cnBusy* can be deasserted for the next transfer. Refer to figure titled *Centronics Inbound Back End Cycle Timing*

During outbound transfers *NcnStrobe* is an output and *cnBusy* is an input. The machine waits for the FIFO to become not full and *cnBusy* and *cnAck* to not be asserted. It then arbitrates for *lsDb* and drives the *lsDb* data bus with the FIFO data, waits for 50ns and then asserts *cnDclk* which should latch this data into an external latch that drives the Centronics bus. The machine waits for 1 microsecond for data settling time and asserts *NcnStrobe* indicating to the peripheral that data is valid. It then waits for the peripheral to assert *NcnBusy* and that *NcnStrobe* has been asserted for at least 1 microsecond before deasserting *NcnStrobe*. If the I/O modifier (IOM) bit is not set the machine will then wait for *cnAck* to be asserted to indicate the acceptance of that cycle. It then returns to the idle state to repeat the process. If the I/O modifier bit is set then the machine immediately returns to the idle state without waiting for the assertion of *cnAck*. Refer to figure titled *Centronics Outbound Back End Cycle Timing*

### 2.1.6.3 Centronics Register Access State Machine (rs232SM)

An internal register read basically gates data on the *BD[23:16]* data bus, allows some data setup time and then asserts *Ndtack16* to indicate valid data. The machine waits for *Nbas24* to deassert before the machine returns to the idle state. When reading from register 5 all of the signals are latched so that the data will not change during the DIO read cycle. When the FIFO is read, its data is latched and driven onto the *BD[23:16]* data bus, and the FIFO un-load clock is pulsed. The data for register 7 is stored internal to the part and is not read from the external latch. Refer to figure titled *Centronics Read Cycle Timing*.

An internal register write enables a set of synchronous FLIP-FLOPS to accept data off of the *BD[23:16]* bus. *Ndtack16* is then asserted to indicate that data was accepted and the machine waits for the deassertion of *Nbas24* to end the cycle. When writing to the FIFO, the FIFO's load clock is asserted and *Ndtack16* is asserted regardless of FIFO full or empty status, software beware. To write to register 7, the machine arbitrates for the *lsDb* data bus, drives *lsDb[7:0]*, waits 50ns and asserts *cnCclk* whose rising edge should latch *lsDb[2:0]* to drive the Centronics control signals NINIT (inverted), NSLCTIN (non-inverted), and AUTOFEEDXT (non-inverted) respectively. Refer to figure titled *Centronics Write Cycle Timing*.

### 2.1.7 Clock generation ERS

The clock generation block generates a 16 or 20MHz clock, a 20MHz, 8MHz, and 2.4576MHz or 7.2727MHz clock. It also generates several reset signals. If *NTest* is asserted, all of the clock outputs are equivalent to *clk40M*.

This block generates 4 reset signals. There are two types: a power-up only reset, and a power-up and chip reset. Both of the reset signals are synchronized and deglitched. *NPuReset* must be valid for at least 150ns to be recognized, but a value of greater than 300ns is recommended. *NReset* must be valid for at least 500ns. The power-up resets are *sNPuReset* and *rNPuReset*. The first is asserted and then held internally for 300ns. The second deasserts within 40ns of the deassertion of *NPuReset* and is used for latching in configuration pin information. The other 2 resets are *sNReset* and *sNReset16*. These are synchronized versions of *NReset*, the first at 20MHz and the second at 16MHz. These are also asserted any time *NPuReset* is asserted.

### 2.1.8 Scan Chain Testing ERS

For improving static "stuck-at" fault coverage, scan chain test circuitry exists within the ASIC. The improvement comes from internal controllable and observable nodes implemented as scannable FLIP-FLOPS. An automatic test generation program (ATG) will generate a test vector suite for use on the SENTRY testers that will detect as many stuck-at faults as possible. The basic concept behind scan-chain testing is to set or clear controllability points, let those levels propagate through asynchronous logic to observability points. The observability points are then compared to precomputed values and any discrepancies are flagged as errors.

Since it is very difficult, and thus expensive, to probe within the pad ring of an ASIC, scannable FLIP-FLOPS are used to obtain internal controllability and observability points. The pins are used as external controllability and observability points. To set the internal controllability points the scannable FLIP-FLOPS are tied into a serial shift register. Once all of the controllability points are valid, the shift register mode is disabled and the FLIP-FLOPS drive their typical circuit paths. Then, all of the FLIP-FLOPS are clocked. The data held in each FLIP-FLOP is some combinatorial logic function of all of the controllability points. Thus the controllability FLIP-FLOPS serve as observability points also. The data in the FLIP-FLOPS is then shifted out and compared with the expected pattern. There are then three operating modes of Nikki:

normal, serial scan, and parallel scan. Serial scan mode is used to shift data in and out of the scan chain. Parallel scan mode arranges the inputs/outputs of the FLIP-FLOPS to the same as in normal mode, except there is one common clock and a few other modifications. It is in parallel scan mode that the FLIP-FLOPS are clocked and become observability points.

There are 4 pins used for scan testing. There is the ScanIn pin (*BD[31]*), a ScanOut pin (*cnError*), a ScanMask pin (*BD[30]*), and the ScanMode pin (*NcnPE*). These pins are configured for scan testing when *NTest* and *NcnSelect* are asserted (low). The ScanIn pin is the data in of the shift register, the ScanOut is the data out of the shift register, and the ScanMask pin when high forces the ScanOut pin to a high. ScanMask is used to mask "unknowns (X)" in the scan chain; the tester checks for a high, rather than checking for an unknown.

Serial mode is selected when the ScanMode pin is low. When high, parallel mode is selected. During serial mode the *clk40M*, is tied directly to all scan flops, all data buses are inputs and all other I/O pins are inputs. Each scan flop is tied to the next in a 344 element shift register. Internally all tri-state bus drivers are disabled. In parallel mode there are some exceptions, but the circuit is basically the same as it is during normal operation. The main exceptions are that all flops are clocked by *clk40M* directly (no divide by two or anything else), and the *NReset* and *NPuReset* are completely asynchronous.

The test vectors generated by ATG should be available through NID.



## 2.2 Pin List and Signal Descriptions

The pin names follow a convention which identifies active levels, interface group and special operation. Signal names that begin with an upper case N are active low. For easy identification, interface signals (those other than DIO) begin, or follow the N, with two letters depending on the interface: *sc* for SCSI, *hs* for HS-HPIB, *ls* for LS-HPIB, *rs* for RS-232, *cn* for Centronics, and *kb* for keyboard. The name following the two character id should correspond to the signal name on the interface chip; i.e. 87033, TMS9914, Medusa, NS16550, etc. In addition, two names separated by a slash indicate shared SCSI/HS-HPIB pins; and names within parenthesis "()" identify the configuration purpose of the pin during the powerup phase. The pin number starts in the "upper left" corner and is numbered counter clockwise. A pull-up column of "Y" indicates that this pad has its HIGHBIAS (pull-up transistor) enabled. Die side is just that and has no relation to the pin numbering. PLOC is a number which indicates where a pad is on the die. The x axis is numbered from left to right, and the y axis is from bottom to top.

**TABLE 1. Pin List**

Nikki Pinout						
Pin	Name	Dir	Pull Up	Die Side	PLOC	Pad Type
1	hsSctrl (scE0)	Output(I)	Y	BOTTOM	10	BUSPAD10
2	scDbP/NhsFast	Tri-state/Open Drain	N	BOTTOM	20	BUSPAD10
3	NscLoopOe/hsB[0]	Output/Tri-state	N	BOTTOM	30	BUSPAD10
4	scWrapLc/hsB[1]	Output/Tri-state	N	BOTTOM	40	BUSPAD10
5	Dirty GND			BOTTOM	50	DGNDPAD
6	NscReset/hsPon	Output	N	BOTTOM	60	BUSPAD10
7	NhsIoend	Input	Y	BOTTOM	70	BUSPAD10
8	NhsIntr	Input	Y	BOTTOM	80	BUSPAD10
9	NhsDmrq	Input	Y	BOTTOM	90	BUSPAD10
10	NSCSIsel/hsA[1]	Input/Output	N	BOTTOM	100	BUSPAD10
11	hsA[2] (scE1)	Output (I)	N	BOTTOM	110	BUSPAD10
12	NMedusaCs (hsEn)	Output (I)	Y	BOTTOM	120	BUSPAD10
13	scHin	Output	N	BOTTOM	130	BUSPAD10
14	Dirty VDD			BOTTOM	140	DVDDPAD
15	NscWrite/NhsIogo	Output	N	BOTTOM	150	pad7
16	Dirty GND			BOTTOM	160	DGNDPAD
17	NscRead/hsWrite	Output	N	BOTTOM	170	pad7
18	NspcCs (scEn)	Output (I)	Y	BOTTOM	180	pad7
	Break VDD			BOTTOM	184	BVDDPAD
	Break GND			BOTTOM	187	BGNDPAD
19	scDmrsp/hsA[0]	Output	N	BOTTOM	190	pad7
20	rsIntr	Input	Y	BOTTOM	200	BUSPAD10
21	rsA[0] (nkSel0)	Output (I)	Y	BOTTOM	210	BUSPAD10
22	rsA[1] (nkSel1)	Output (I)	Y	BOTTOM	220	BUSPAD10
23	rsA[2]	Output	Y	BOTTOM	230	BUSPAD10
24	Dirty GND			BOTTOM	240	DGNDPAD
25	rsReset	Output	N	BOTTOM	250	BUSPAD10
26	NrsRemEn	Output	N	BOTTOM	260	BUSPAD10
27	NrsIoW	Output	N	BOTTOM	270	BUSPAD10
28	NrsIoR	Output	N	BOTTOM	280	BUSPAD10
29	Dirty VDD			BOTTOM	290	DVDDPAD
30	N8250cs (rsEn)	Output (I)	Y	BOTTOM	300	BUSPAD10
31	NcnStrobe	Input/Output	N	BOTTOM	310	BUSPAD10
32	cnDclk	Output	N	BOTTOM	320	BUSPAD10
33	Dirty GND			BOTTOM	330	DGNDPAD
34	cnCclk	Output	N	BOTTOM	340	BUSPAD10
35	cnError/ScanOut	Input/Output	N	BOTTOM	350	BUSPAD10
36	cnAck	Input	N	BOTTOM	360	BUSPAD10
37	cnBusy	Input/Output	N	BOTTOM	370	BUSPAD10
38	Clean GND			BOTTOM	380	CGNDPAD
39	NcnPE	Input	N	BOTTOM	390	BUSPAD10
40	NcnSelect	Input	Y	BOTTOM	400	BUSPAD10



Nikki Pinout						
Pin	Name	Dir	Pull Up	Die Side	PLOC	Pad Type
41	NcnOut (cnEn)	Output (I)	Y	RIGHT	10	BUSPAD10
42	oe	Input	Y	RIGHT	20	BUSPAD10
43	NTest	Input	Y	RIGHT	30	BUSPAD10
44	lsDb[0]	Tri-state	N	RIGHT	40	BUSPAD10
45	lsDb[1]	Tri-state	N	RIGHT	50	BUSPAD10
46	Dirty GND			RIGHT	60	DGNDDPAD
47	lsDb[2]	Tri-state	N	RIGHT	70	BUSPAD10
48	lsDb[3]	Tri-state	N	RIGHT	80	BUSPAD10
49	lsDb[4]	Tri-state	N	RIGHT	90	BUSPAD10
50	lsDb[5]	Tri-state	N	RIGHT	100	BUSPAD10
51	Dirty VDD			RIGHT	110	DVDDPAD
52	lsDb[6]	Tri-state	N	RIGHT	120	BUSPAD10
53	lsDb[7]	Tri-state	N	RIGHT	130	BUSPAD10
54	NkbNmi	Input	Y	RIGHT	140	BUSPAD10
55	NlsCont	Input	N	RIGHT	150	BUSPAD10
56	NlsAccRq	Input	Y	RIGHT	160	BUSPAD10
57	NlsIntr	Input	Y	RIGHT	170	BUSPAD10
58	N9914cs (lsEn)	Output (I)	Y	RIGHT	180	BUSPAD10
59	Dirty GND			RIGHT	190	DGNDDPAD
60	NlsAccGr	Output	N	RIGHT	200	BUSPAD10
61	VDD (core)			RIGHT	210	VDDPAD
62	lsDbIn	Output	N	RIGHT	220	BUSPAD10
63	NlsWe	Output	N	RIGHT	230	BUSPAD10
64	lsSctrl	Output	N	RIGHT	240	BUSPAD10
65	ba[1]	Input	N	RIGHT	250	BUSPAD10
66	ba[2]	Input	N	RIGHT	260	BUSPAD10
67	ba[3]	Input	N	RIGHT	270	BUSPAD10
68	ba[4]	Input	N	RIGHT	280	BUSPAD10
69	ba[5]	Input	N	RIGHT	290	BUSPAD10
70	ba[15]	Input	N	RIGHT	300	BUSPAD10
71	ba[16]	Input	N	RIGHT	310	BUSPAD10
72	ba[17]	Input	N	RIGHT	320	BUSPAD10
73	ba[18]	Input	N	RIGHT	330	BUSPAD10
74	ba[19]	Input	N	RIGHT	340	BUSPAD10
75	ba[20]	Input	N	RIGHT	350	BUSPAD10
76	ba[21]	Input	N	RIGHT	360	BUSPAD10
77	ba[22]	Input	N	RIGHT	370	BUSPAD10
78	ba[23]	Input	N	RIGHT	380	BUSPAD10
	Break GND			RIGHT	384	BGNDDPAD
	Break VDD			RIGHT	387	BVDDPAD
79	bd[16]	Tri-State	N	RIGHT	390	BUSPAD7
80	bd[17]	Tri-State	N	RIGHT	400	BUSPAD7



Nikki Pinout						
Pin	Name	Dir	Pull Up	Die Side	PLOC	Pad Type
81	bd[18]	Tri-State	N	TOP	400	BUSPAD7
82	Dirty GND			TOP	390	DGNDPAD
83	bd[19]	Tri-State	N	TOP	380	BUSPAD7
84	bd[20]	Tri-State	N	TOP	370	BUSPAD7
85	Dirty VDD			TOP	360	DVDDPAD
86	bd[21]	Tri-State	N	TOP	350	BUSPAD7
87	bd[22]	Tri-State	N	TOP	340	BUSPAD7
88	bd[23]	Tri-State	N	TOP	330	BUSPAD7
89	bd[24]	Tri-State	N	TOP	320	BUSPAD7
90	Dirty GND			TOP	310	DGNDPAD
91	bd[25]	Tri-State	N	TOP	300	BUSPAD7
92	bd[26]	Tri-State	N	TOP	290	BUSPAD7
93	bd[27]	Tri-State	N	TOP	280	BUSPAD7
94	bd[28]	Tri-State	N	TOP	270	BUSPAD7
95	bd[29]	Tri-State	N	TOP	260	BUSPAD7
96	bd[30]	Tri-State	N	TOP	250	BUSPAD7
97	Dirty VDD			TOP	240	DVDDPAD
98	bd[31]	Tri-State	N	TOP	230	BUSPAD7
99	Dirty GND			TOP	220	DGNDPAD
100	bd[0]	Tri-State	N	TOP	210	BUSPAD10
101	bd[1]	Tri-State	N	TOP	200	BUSPAD10
102	bd[2]	Tri-State	N	TOP	190	BUSPAD10
103	bd[3]	Tri-State	N	TOP	180	BUSPAD10
104	bd[4]	Tri-State	N	TOP	170	BUSPAD10
105	bd[5]	Tri-State	N	TOP	160	BUSPAD10
106	Dirty GND			TOP	150	DGNDPAD
107	bd[6]	Tri-State	N	TOP	140	BUSPAD10
108	bd[7]	Tri-State	N	TOP	130	BUSPAD10
109	bd[8]	Tri-State	N	TOP	120	BUSPAD10
110	bd[9]	Tri-State	N	TOP	110	BUSPAD10
111	bd[10]	Tri-State	N	TOP	100	BUSPAD10
112	Dirty VDD			TOP	90	DVDDPAD
113	bd[11]	Tri-State	N	TOP	80	BUSPAD10
114	bd[12]	Tri-State	N	TOP	70	BUSPAD10
115	Dirty GND			TOP	60	DGNDPAD
116	bd[13]	Tri-State	N	TOP	50	BUSPAD10
117	bd[14]	Tri-State	N	TOP	40	BUSPAD10
118	Clean VDD			TOP	30	CVDDPAD
119	bd[15]	Tri-State	N	TOP	20	BUSPAD10
	Break VDD			TOP	17	BVDDPAD
	Break GND			TOP	14	BGNDPAD
120	NPureSet	Input	Y	TOP	10	BUSPAD10



Nikki Pinout						
Pin	Name	Dir	Pull Up	Die Side	PLOC	Pad Type
121	NReset	Input	Y	LEFT	400	BUSPAD10
122	Ndone	Input	N	LEFT	390	BUSPAD10
123	Ndmack[0]	Input	N	LEFT	380	BUSPAD10
124	Ndmack[1]	Input	N	LEFT	370	BUSPAD10
125	Nlds	Input	N	LEFT	360	BUSPAD10
126	Nbas24	Input	N	LEFT	350	BUSPAD10
127	Ndtkall	Input	N	LEFT	340	BUSPAD10
128	Ndmrdy	Open Drain	N	LEFT	330	BUSPAD7
129	Ndmrq[0]	Open Drain	N	LEFT	320	BUSPAD7
130	Dirty GND			LEFT	310	DGNDDPAD
131	Ndmrq[1]	Open Drain	N	LEFT	300	BUSPAD7
132	NdmaActive	Open Drain	N	LEFT	290	BUSPAD10
133	Nwrite	Input	N	LEFT	280	BUSPAD10
134	Dirty VDD			LEFT	270	DVDDPAD
135	Ndtack16	Open Drain	N	LEFT	260	BUSPAD7
136	Nir[3]	Open Drain	Y	LEFT	250	BUSPAD10
137	Nir[4]	Open Drain	Y	LEFT	240	BUSPAD10
138	Nir[5]	Open Drain	Y	LEFT	230	BUSPAD10
139	GND (core)			LEFT	220	GNDPAD
140	Dirty GND			LEFT	210	DGNDDPAD
141	Nir[6]	Open Drain	Y	LEFT	200	BUSPAD10
142	Nima	Open Drain	N	LEFT	190	BUSPAD7
143	clk40M	Input	N	LEFT	180	BUSPAD10
	Break GND			LEFT	177	BGNDDPAD
	Break VDD			LEFT	174	BVDDPAD
144	clk8M	Output	N	LEFT	170	BUSPAD7
145	Dirty VDD			LEFT	160	DVDDPAD
146	Dirty GND			LEFT	150	DGNDDPAD
147	clk24576/7.27	Output	N	LEFT	140	BUSPAD7
	Break GND			LEFT	137	BGNDDPAD
	Break VDD			LEFT	134	BVDDPAD
148	scDmrq	Input	Y	LEFT	130	BUSPAD10
149	scIntr	Input	Y	LEFT	120	BUSPAD10
150	scTrmPwr	Input	N	LEFT	110	BUSPAD10
151	exDb[0]	Tri-state	N	LEFT	100	BUSPAD10
152	exDb[1]	Tri-state	N	LEFT	90	BUSPAD10
153	exDb[2]	Tri-state	N	LEFT	80	BUSPAD10
154	exDb[3]	Tri-state	N	LEFT	70	BUSPAD10
155	Dirty GND			LEFT	60	DGNDDPAD
156	exDb[4]	Tri-state	N	LEFT	50	BUSPAD10
157	Dirty VDD			LEFT	40	DVDDPAD
158	exDb[5]	Tri-state	N	LEFT	30	BUSPAD10
159	exDb[6]	Tri-state	N	LEFT	20	BUSPAD10
160	exDb[7]	Tri-state	N	LEFT	10	BUSPAD10

### 2.2.1 Signal Descriptions

TABLE 2. DIO signals

Pin Name	Type	Description (DIO Signal Name)
oe	I	Output enable, when low, all outputs are tristated, and the pad pullup transistors are disabled on signals <i>NTest</i> , <i>hsSctrl</i> , and <i>hsA2</i> .
NTest	I	Test signal, when low chip is in special test mode. When asserted, <i>NcnSelect</i> and <i>NcnPE</i> select Serial scan mode (both low) or Parallel scan mode ( <i>NcnSelect</i> low and <i>NcnPE</i> high); also, <i>bd[31]</i> becomes the scan in pin, <i>cnError/ScanOut</i> becomes the scan out pin, and <i>bd[30]</i> becomes the scan mask pin. See section on test for further details.
NPuReset	I	Power-up/Hard Reset (PURESET*). When asserted (low) the entire chip is reset, including hidden registers, to its default state and on the rising edge of <i>NPuReset</i> the configuration input pins latch their current values.
NReset	I	DIO Bus Reset (RESET*). When asserted most of the chip is reset. The current configuration and hidden register values are preserved. For each module the assertion of <i>NReset</i> is analogous to writing to the soft reset register.
Ndtkall	I	Logical OR of all DIO BUS DTACKs. This signal is used during outbound DMA. The DMA machines wait for the assertion of this signal to indicate that valid data exists on the <i>bd</i> bus.
Ndtack16	Open Drain	16 bit active low data transfer acknowledge signal (DTACK16*). This signal is asserted when Nikki has either received the data or presented valid data on the <i>bd[23:16]</i> bus. This signal can drive an expandable DIO bus directly.
Nwrite	I	Active low write, active high read (BR/W*). When asserted during NON-DMA cycles data is being written into Nikki, else it is being read. When asserted during LS-HPIB DMA cycles, it indicates that memory is being written, Nikki is driving the <i>bd[23:16]</i> bus, otherwise main memory is supplying the data, Nikki is receiving data.
Nbas24	I	Active low 24 bit address strobe (BAS24*). When asserted this signal indicates that the address, <i>ba</i> , and the control, <i>Nwrite</i> , signals are valid. This signal is not used for DMA cycles.
Nlds	I	Active low lower data strobe (BLDS*). When asserted, this signal indicates either that data is valid on the bus or that the bus is ready for Nikki to drive the data bus during DIO register accesses. This signal is not used for DMA cycles.

ba[23:15,5:1]	I	Bus address (BA23..BA1). These lines are decoded to select the various modules/registers within Nikki. The upper 8 bits <i>ba[23:16]</i> indicate a particular select code within which exist registers selected by the lower bits.
bd[31:0]	I/O	Bus data (BD15..BD0,XD15..XD0). All non-DMA accesses will use data lane 1 which comprise <i>bd[23:16]</i> . Longword DMA uses all lanes, <i>bd[31:0]</i> , word DMA uses lanes 0 and 1, <i>bd[31:16]</i> , and byte DMA uses lane 1, <i>bd[23:16]</i> .
Nir[6:3]	Open Drain	Active low interrupting level output (IR6*, IR5*, IR4*, IR3*). When asserted, some component of Nikki is indicating an interrupt on one of the four levels. Each module's interrupting level can be modified through its first hidden register.
Nima	O	Active low output "I am active" (IMA*). When asserted, some component of Nikki is going to perform a read or write cycle starting when <i>Nlds</i> is asserted. This signal will be asserted as long as the address and <i>Nbas24</i> are asserted. This signal should be used in conjunction with <i>Nwrite</i> to control the data buffer direction between Nikki and the DIO data bus.
Ndmrq[1:0]	Open Drain	Active low DMA request for channel 1 and 0 (DMRQ1*, DMRQ0*). When one of the modules within Nikki wants to perform a DMA cycle, one of these signals will be asserted.
Ndmack[1:0]	I	Active low DMA acknowledge for channel 1 and 0 (DMACK1*, DMACK0*). These signals are used analogously to <i>Nbas24</i> but for DMA cycles. When asserted, they indicate that the DMA controller is performing a cycle.
Ndmrdy	Open Drain	Active low DMA ready (DMARDY*). When asserted, Nikki has either presented valid data on, or has accepted data off of the <i>bd</i> data bus during a DMA cycle. This signal is analogous to <i>Ndtack16</i> but for DMA cycles.
Ndone	I	Active low DMA done (DONE*). This signal is asserted by the DMA controller during the last DMA cycle. It is only used by the HS-HPiB module within Nikki.
NdmaActive	O	Active low "I am DMAing" output. This signal is asserted in response to a <i>Ndmack</i> assertion to indicate that some module within Nikki will perform a DMA cycle. It is asserted the entire time <i>Ndmack</i> is asserted (less asynchronous delays, of course). It should be used in conjunction with <i>Nwrite</i> to control the direction of the data buffers between Nikki and the DIO data bus.

**TABLE 3. Clock signals**

Pin Name	Type	Description
clk40M	I	Input 40MHz clock. All internal clocks are generated from this input.
clk8M	O	Output 8MHz or 10MHz clock for the MB37030/3. This frequency can be changed between 8MHz and 10MHz at any time through a bit in the SCSI hidden register set.
clk24576	O	Output 2.4567MHz or 7.2727MHz clock for Baud rate generation by the NS16550. This frequency can be changed between 2.4567MHz and 7.2727MHz at any time through a bit in the RS232 hidden register set.



**TABLE 4. SCSI support signals**

Pin Name	Type	Description (87033 pin name)
scDmrsp	O	SPC (87030/33 SCSI Protocol Controller) DMA Response (DRESP). Active high output to indicate that Nikki is ready to transfer DMA data to/from the SPC (87030/33).
scDmrq	I	SPC DMA Request (DREQ). Active high input when asserted indicates that the SPC is requesting a DMA transfer.
scHin	O	SPC DMA data bus direction signal (HIN). When asserted indicates that DMA transfers will be inbound.
exDb[7:0]	I/O	SPC Data Bus bits 7-0 (D[7:0] and HDB[7:0]).
scDbP	I/O	SPC Data Bus parity (DBP, HDBP).
NspcCs	O(I)	SPC Chip Select Output ( $\overline{CS}$ ). This active low signal is asserted when SPC registers are accessed. If asserted, as an input, during <i>NPuReset</i> the SCSI module will be disabled completely.
NscWrite	O	SPC WRITE ( $\overline{WT}$ ). This active low output is asserted during SPC register writes.
NscRead	O	SPC READ ( $\overline{RD}$ and $\overline{RDG}$ ). This active low output is asserted during SPC register reads.
scIntr	I	SPC Interrupt input (INTR). This input is used to propagate SPC interrupts to <i>Nir[6:3]</i> if so enabled.
NscReset	O	SPC reset output ( $\overline{RESET}$ ). This active low output is asserted when <i>Nreset</i> is asserted or during a write to SCSI register 1.
scWrapLe	O	SCSI Wraparound Register Latch Enable. This active high signal when asserted indicates the <i>exDb[7:0]</i> holds data for the wraparound register. The wraparound register drives SCSI control lines NIO, NCD, NMSG, NBSY, no-connect, no-connect, NACK, and NREQ via <i>exDb[0:7]</i> inverted respectively.
NscLoopOe	O	SCSI Loopback Output Enable. This active low signal when asserted should enable the SCSI data bus on to <i>exDb[7:0]</i> .
scTrmPwr	I	SCSI Terminator Resistor Power. This input should be asserted if the SCSI terminator resistors are powered up.

NSCSIsel            I        SCSI/HS-HPIB selection. If asserted (low) during the deassertion of *NPuReset* then the SCSI module is enabled, assuming *NspcCs* is high (floating) during this period, otherwise HS-HPIB is enabled, again assuming *NMedusaCs* is high (floating).

**TABLE 5. High Speed HPIB Support Signals**

*Note: The following signals indicate only the HS-HPIB name of the signal. In the case of shared SCSI/HS-HPIB pins, these signals indicate the pin usage during HS-HPIB mode and may be redundant with the SCSI section. Use the Nikki pinout table to cross reference to actual signal name.*

Pin Name	Type	Description (Medusa pin name)
NhsDmrq	I	Medusa DMA Request ( $\overline{\text{DMARQ}}$ ). Active low input when asserted indicates that Medusa is ready for a DMA transfer.
NhsIogo	O	Medusa transfer handshake signal ( $\overline{\text{IOGO}}$ ). Active low output is asserted for all Medusa register accesses including DMA transfers.
NhsIoend	I	Medusa transfer handshake signal ( $\overline{\text{IOEND}}$ ). Active low input is asserted by Medusa to signal acceptance of a register read or write.
exDb[7:0]	I/O	Medusa data lines (D[8:15] respectively).
hsB[1:0]	I/O	Medusa D[1:0] 'data' lines. These are used to indicate what type of information <i>exDb[7:0]</i> contains.
NMedusaCs	O(I)	Medusa Chip Select ( $\overline{\text{CHSEL}}$ ). Active low output to select Medusa. If asserted, as an input, during <i>NPuReset</i> , the HS-HPIB module is completely disabled.
hsA[2:0]	O	Medusa address lines (A[13:15] respectively). During register accesses these lines are identical to <i>ba[3:1]</i> . During DMA transfers these lines specify access of the Medusa FIFO register.
hsWrite	O	Medusa write/read signal (W). Active high signal when asserted specifies a write operation, otherwise its a read operation.
NhsIntr	I	Medusa interrupt signal ( $\overline{\text{INT}}$ ). Active low input when asserted indicates an interrupting condition of Medusa.
hsSctrl	O	HPIB System Controller. Active high output when asserted indicates that this HPIB controller is a system controller. This signal level is specified in a HS-HPIB hidden register.
hsPon	O	Medusa Power on (PON). Active high output when not-asserted indicates a reset condition.
NhsFast	open drain	HPIB FAST select. This output should ultimately vary the Medusa timing resistor for tuned or "not-tuned" operation. This signal level is specified in a HS-HPIB hidden register.

**TABLE 6. Low Speed HPIB Support Signals**

*Note: The TMS9914 address lines (RS2-0) are shared with RS-232 on pins rsA[2:0]*

Pin Name	Type	Description (TMS9914 pin name)
NlsIntr	I	Active low LS-HPIB interrupt signal from TMS9914 ( $\overline{\text{INT}}$ ).
NlsAccRq	I	TMS9914 support signal, DMA ACCESS REQUEST ( $\overline{\text{ACCRQ}}$ ).
lsSctrl	O	If asserted, LS-HPIB is system controller. This signal should be used to control the direction of the HPIB control signals.
NlsCont	I	TMS9914 support signal, Controller in Charge ( $\overline{\text{CONT}}$ ).
NlsWe	O	TMS9914 support signal, WRITE ENABLE ( $\overline{\text{WE}}$ ).
lsDbin	O	TMS9914 support signal, DATA BUS IN (DBIN).
NlsAccGr	O	TMS9914 support signal, ACCESS GRANTED ( $\overline{\text{ACCGR}}$ ).
lsDb[7:0]	O	Shared LS-HPIB, RS232, and Centronics databus (D7-D0).
NkbNmi	I	Keyboard NMI signal reported in register 5 of LS-HPIB register set.
N9914cs	O(I)	TMS9914 chip enable ( $\overline{\text{CE}}$ ). If asserted (low), as an input, during <i>NPuReset</i> the LS-HPIB module is completely disabled except for register 5 if Nikki #1. Register 5 still exists for keyboard support when LS-HPIB is disabled. But, register 5 (along with the rest of LS-HPIB) does not exist if Nikki is configured as #2, #3, or #4.

**TABLE 7. Internal RS-232 support signals**

*Note: The data bus of the NS16550 is shared with LS-HPIB and Centronics on pins lsDb[7:0].*

Pin Name	Type	Description (NS16550 pin name)															
rsIntr	I	Active high interrupt request from RS-232 controller (INTR).															
NrsIoR	O	I/O read signal for RS-232 controller ( $\overline{RD}$ ).															
NrsIoW	O	I/O write signal for RS-232 controller ( $\overline{WR}$ ).															
rsReset	O	RS-232 reset, active high (MR).															
rsA[2:0]	O	Register select (address) for RS-232 controller (A2-0), and LS-HPIB. The signal level on rsA[1:0] when <i>NPuReset</i> is deasserted selects one of four Nikki configurations.															
		<table border="0"> <thead> <tr> <th>rsA[1]</th> <th>rsA[0]</th> <th></th> </tr> </thead> <tbody> <tr> <td>1</td> <td>1</td> <td>Nikki #1</td> </tr> <tr> <td>1</td> <td>0</td> <td>Nikki #2</td> </tr> <tr> <td>0</td> <td>1</td> <td>Nikki #3</td> </tr> <tr> <td>0</td> <td>0</td> <td>Nikki #4</td> </tr> </tbody> </table>	rsA[1]	rsA[0]		1	1	Nikki #1	1	0	Nikki #2	0	1	Nikki #3	0	0	Nikki #4
rsA[1]	rsA[0]																
1	1	Nikki #1															
1	0	Nikki #2															
0	1	Nikki #3															
0	0	Nikki #4															
NrsRemEn	O	RS-232 support signal for MODEM HANDSHAKE. When asserted this signal should assert the following NS16550 signals: $\overline{DSR}$ , $\overline{DCD}$ , $\overline{CTS}$ , $\overline{RI}$															
N8250cs	O(I)	NS16550 (NS8250) chip select, active low ( $\overline{CS2}$ ). If asserted, as an input, during <i>NPuReset</i> the RS-232 module will be disabled completely.															

**TABLE 8. Centronics Support Signals**

Pin Name	Type	Description
NcnStrobe	I/O	This is the Centronics NSTROBE control signal. It should be buffered from the Centronics bus with an open collector output gate and a schmidt triggered input gate.
cnAck	I	This is the Centronics NACK (acknowledge) control signal. It should be buffered from the Centronics bus with a schmidt triggered <i>inverting</i> input gate.
cnBusy	I/O	This is the Centronics BUSY control signal. It should be buffered from the Centronics bus with an open collector output gate and a schmidt triggered input gate.
NcnPE	I	This is the Centronics PE (paper error) status signal. It should be buffered from the Centronics bus with a schmidt triggered <i>inverting</i> input gate.
NcnSelect	I	This is the Centronics SELECT status signal. It should be buffered from the Centronics bus with a schmidt triggered <i>inverting</i> input gate.
cnError/ScanOut	I(O)	When <i>NTest</i> is not asserted (high) this is the Centronics NERROR status signal. It should be buffered from the Centronics bus with a schmidt triggered <i>inverting</i> input gate. When <i>NTest</i> is asserted (low) this is the scan chain output pin <i>ScanOut</i> .
NcnOut	I	This is the Centronics direction control signal. When asserted (low) <i>Nikki</i> is driving <i>NcnStrobe</i> and receiving <i>cnBusy</i> . The Centronics data bus register (74ALS654, see below) should also be driving the Centronics data bus. When deasserted (high), <i>Nikki</i> is driving <i>cnBusy</i> and receiving <i>NcnStrobe</i> . The Centronics data bus register should be enabled to drive <i>lsDb[7:0]</i> from the Centronics data bus when <i>NcnOut</i> is deasserted and <i>cnDclk</i> is asserted. While <i>NcnOut</i> is deasserted, the data register should not drive the Centronics data bus.
cnCclk	I	Centronics control clock. The rising edge of this signal should be used to latch <i>lsDb[2:0]</i> which are the respective Centronics control signals: NINIT, NSLCTIN, and WRnRD (AUTOFEEDXT). WRnRD and NSLCTIN should be inverted before driving the Centronics bus. NINIT is the same sense on <i>lsDb[2]</i> as it is on the Centronics bus.

cnDclk

I/O

This is the Centronics data control signal. When *NcnOut* is asserted (low) this signal should be used to latch data from the *lsDb[7:0]* data bus into a register (74ALS654) that drives the Centronics data bus DATA[7:0]. The register should latch on the rising edge of *cnDclk*. When *NcnOut* is not asserted (high) this signal should be used to enable data buffers onto the *lsDb[7:0]* data bus from the Centronics data bus DATA[7:0]. The buffers should drive when *cnDclk* is asserted (high).

132	signal pins
11	power pins
17	ground pins
160	total pins

**TABLE 9. Absolute Maximum Ratings**

Vdd	5.5V
Supply current	2000 mA
DC input voltage	Vdd
DC input current	+/- 100 mA
Storage temperature	-40 C to 125 C
Ambient temperature under bias	-20 C to 85 C

Stress beyond listed maximum ratings may cause permanent damage to the device. Exposure to maximum rated conditions for extended periods will adversely affect device reliability.

**TABLE 10. Input Protection**

Electrostatic Discharge (between any 2 pins)	+/- 2.0KV
DC input current (for latchup protection)	+/- 100mA

**TABLE 11. Electrical Characteristics Over Operating Range**

Parameter	Condition	MAX	TYP
Vdd	Operating Voltage	5.25 V	4.5 V
	Junction Temperature	85 C	0 C
Idd	Supply Current	200 mA	50 mA
Voh	High level output voltage	Ioh max	2.4V
Vol	Low level output voltage	Iol max	0.4V
Vih	High level input voltage		2.0V
Vil	Low level input voltage	0.8V	
Iil/Iih	Input leakage current	10µA	-10µA
Iolk	Tristate output leakage	10µA	-10µA



### 2.3 DC Electrical Characteristics

The following DC characteristics are over the range:  $4.5V \leq V_{dd} \leq 5.25V$ . Each pads output current is tested to these specifications in addition to one-half of the indicated value at  $V_{DD}=4.5V$ .

Group	Ioh (@Voh)	Iol (@ Vol)
Group 1	1.0ma	-4.0ma
Group 2	2.0ma	-4.0ma
Group 3	3.0ma	-7.4ma
Group 4		-7.4ma
Group 5	2.2ma	-4.5ma

Pin	Name	I/O	DC Current Group
1	hsSctrl	O	1
2	scDbP/NhsFast	Tri-state	1
3	NscLoopOe/hsB[0]	Tri-state	2
4	scWrapLe/hsB[1]	Tri-state	2
5	NscReset/hsPon	Output	1
6	CVDD		
7	NhsIoend	Input	-
8	NhsIntr	Input	-
9	NhsDmrq	Input	-
10	NSCSIsel/hsA[1]	Output	5
11	DGND		
12	hsA[2]	Output	1
13	NmedusaCs	Output(PU)	1
14	scHin/hsSctrl	Output	1
15	DVDD		
16	Nscwrite/NhsIogo	Output	1
17	Nscread/hsWrite	Output	1
18	NspcCs	Output(PU)	1
19	scDmrsp/hsA[0]	1	
20	DGND		
	BVDD		
	BVDD		
21	DVDD		
22	DGND		
23	NrsIntr	Input	-
24	rsA0	Output(PU)	1
25	rsA1	Output(PU)	1
26	rsA2	Output(PU)	1
27	rsReset	Output	1
28	NrsRemEn	Output	1
29	NrsIoW	Output	1
30	DGND		
31	NrsIoR	Output	1
32	N8250cs	Output(PU)	1
33	NcnStrobe	Input/Output	1
34	cnDclk	Output	1
35	cnCclk	Output	1
36	CGND		
37	NcnError	Output	1
38	NcnAck	Input	-
39	cnBusy	Input/Output	1
40	cnPe	Input	-

Pin	Name	I/O	DC Current Group
41	cnSelect	I/O(PU)	1
42	cnOut	I/O(PU)	1
43	oe	Input(PU)	-
44	Ntest	Input(PU)	-
45	lsDb0	Tri-state	1
46	lsDb1	Tri-state	1
47	lsDb2	Tri-state	1
48	lsDb3	Tri-state	1
49	lsDb4	Tri-state	1
50	lsDb5	Tri-state	1
51	lsDb6	Tri-state	1
52	lsDb7	Tri-state	1
53	NkbNmi	Input	-
54	NlsCont	Input	-
55	NlsAccRq	Input	-
56	NlsIntr	Input	-
57	N9914cs	Output(PU)	1
58	NlsAccGr	Output	1
59	DVDD		
60	DGND		
61	VDD	(core)	
62	lsDbIn	Output	1
63	NlsWe	Output	1
64	lsSctrl	Output	1
65	ba1	Input	-
66	DGND		
67	ba2	Input	-
68	ba3	Input	-
69	ba4	Input	-
70	ba5	Input	-
71	DVDD		
72	ba15	Input	-
73	ba16	Input	-
74	ba17	Input	-
75	ba18	Input	-
76	ba19	Input	-
77	ba20	Input	-
78	ba21	Input	-
79	ba22	Input	-
80	ba23	Input	-
	BVDD		
	BGND		

Pin	Name	I/O	DC Current Group
81	d16	TriState	3
82	d17	TriState	3
83	d18	TriState	3
84	d19	TriState	3
85	DGND		
86	d20	TriState	3
87	d21	TriState	3
88	d22	TriState	3
89	d23	TriState	3
90	DVDD		
91	d24	TriState	3
92	d25	TriState	3
93	d26	TriState	3
94	d27	TriState	3
95	DGND		
96	d28	TriState	3
97	d29	TriState	3
98	d30	TriState	3
99	d31	TriState	3
100	DGND		
101	d0	TriState	2
102	d1	TriState	2
103	d2	TriState	2
104	d3	TriState	2
105	DGND		
106	d4	TriState	2
107	d5	TriState	2
108	d6	TriState	2
109	d7	TriState	2
110	DVDD		
111	d8	TriState	2
112	d9	TriState	2
113	d10	TriState	2
114	d11	TriState	2
115	DGND		
116	d12	TriState	2
117	d13	TriState	2
118	d14	TriState	2
119	d15	TriState	2
	BVDD		
	BGND		
120	NpuReset	Input(PU)	-

Pin	Name	I/O	DC Current Group
121	Nreset	Input(PU)	-
122	Ndone	Input	-
123	Ndmack0	Input	-
124	Ndmack1	Input	-
125	Nlds	Input	-
126	Nbas24	Input	-
127	DGND		
128	Ndtkall	Input	-
129	Ndmardy	Open Drain	4
130	Ndmrq0	Open Drain	4
131	Ndmrq1	Open Drain	4
132	NdmaActive	Output	1
133	DVDD		
134	Nwrite	Input	-
135	Ndtack16	Open Drain	4
136	Nir3	Open Drain(PU)	1
137	Nir4	Open Drain(PU)	1
138	Nir5	Open Drain(PU)	1
139	Nir6	Open Drain(PU)	1
140	Nima	Open Drain	2
141	GND	(core)	
142	clk40M	Input	-
143	DVDD		
	BGND		
	BVDD		
144	clk8M	Output	1
145	DVDD		
146	DGND		
147	clk24576	1	
	BGND		
	BVDD		
148	scDmrq	Input	-
149	scIntr	Input	-
150	scTrmPwr	Input	-
151	DVDD		
152	exDb[0]	Tri-state	1
153	exDb[1]	Tri-state	1
154	exDb[2]	Tri-state	1
155	exDb[3]	Tri-state	1
156	DGND		
157	exDb[4]	Tri-state	1
158	exDb[5]	Tri-state	1
159	exDb[6]	Tri-state	1
160	exDb[7]	Tri-state	1

*2.4 AC Electrical Specifications*

## 2.5 Timing Diagrams

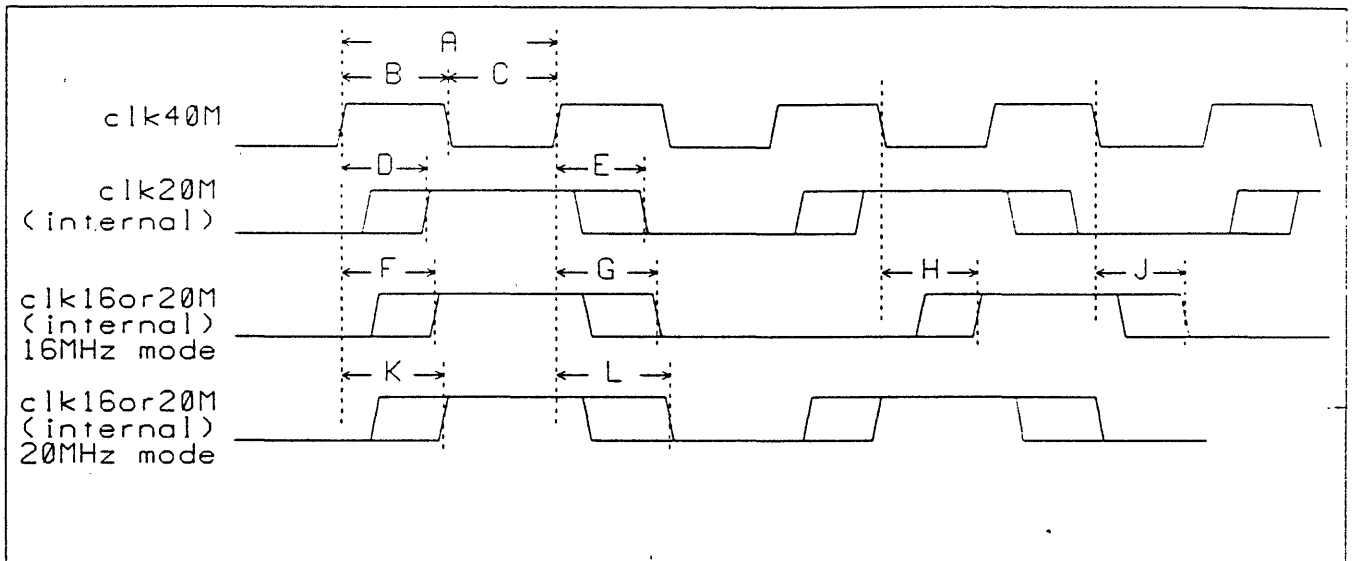
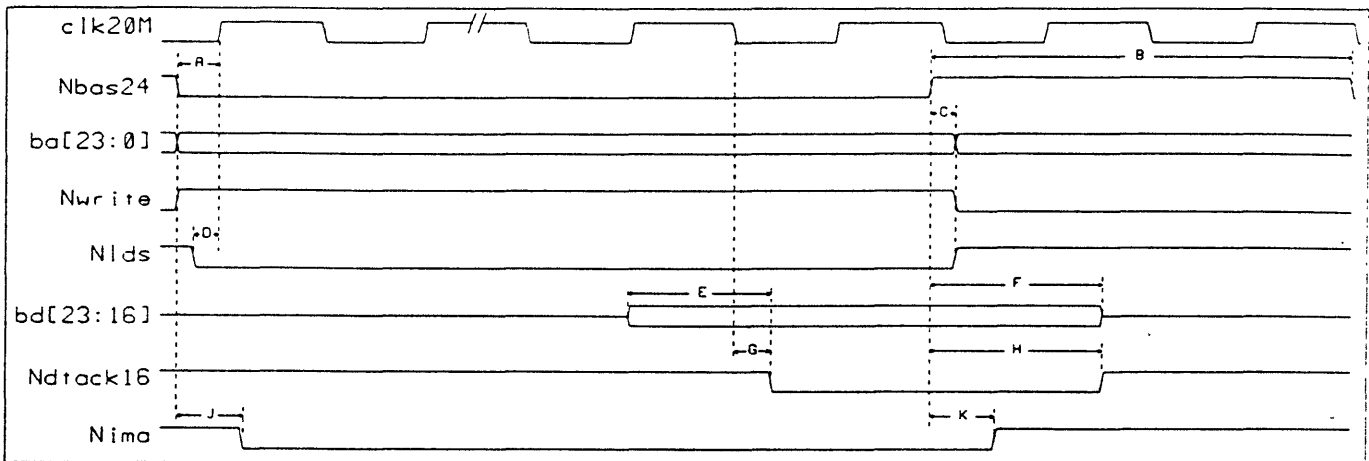


Figure 2. Clock Timing

REF	Parameter	MAX	MIN	NOTE
A	Input clock period	26.25ns	23.75ns	
B	Input clock high time	14.75ns	8.0ns	
C	Input clock low time	14.75ns	8.0ns	
D	Internal clk20M rising delay	9.93ns	2.95ns	
E	Internal clk20M falling delay	9.93ns	2.99ns	
F	Internal clk16or20M rising delay from clk40M rising edge (16MHz mode)	11.37ns	3.73ns	
G	Internal clk16or20M falling delay from clk40M rising edge (16MHz mode)	11.37ns	3.56ns	
H	Internal clk16or20M rising delay from clk40M falling edge (16MHz mode)	9.83ns	3.42ns	
J	Internal clk16or20M falling delay from clk40M falling edge (16MHz mode)	9.84ns	3.27ns	
K	Internal clk16or20M rising delay from clk40M rising edge (20MHz mode)	12.03ns	3.82ns	
L	Internal clk16or20M falling delay from clk40M rising edge (20MHz mode)	11.73ns	3.58ns	



**Figure 3. DIO Read Cycle Timing**

REF	Parameter	MAX	MIN	NOTE
A	Nbas24 setup to clk20M (tester ref)		12.82ns	
A	Nbas24 setup to clk16or20M (tester ref)		16.41ns	
B	Nbas24 high time		125ns	
C	Address and write hold from Nbas24 or Nlds		15ns	
D	Nlds setup (tester ref)		6.4ns	
E	Data setup to Ndtack16		35ns	
F	Data tristate after Nbas deassertion	20ns	0ns	
G	Clock falling to Ndtack16 falling	13ns	0ns	
H	Ndtack16 release after Nbas deassertion	15ns	0ns	
J	Nima assertion after Nbas assertion	17ns	0ns	
K	Nima deassertion after Nbas deassertion	22ns	0ns	



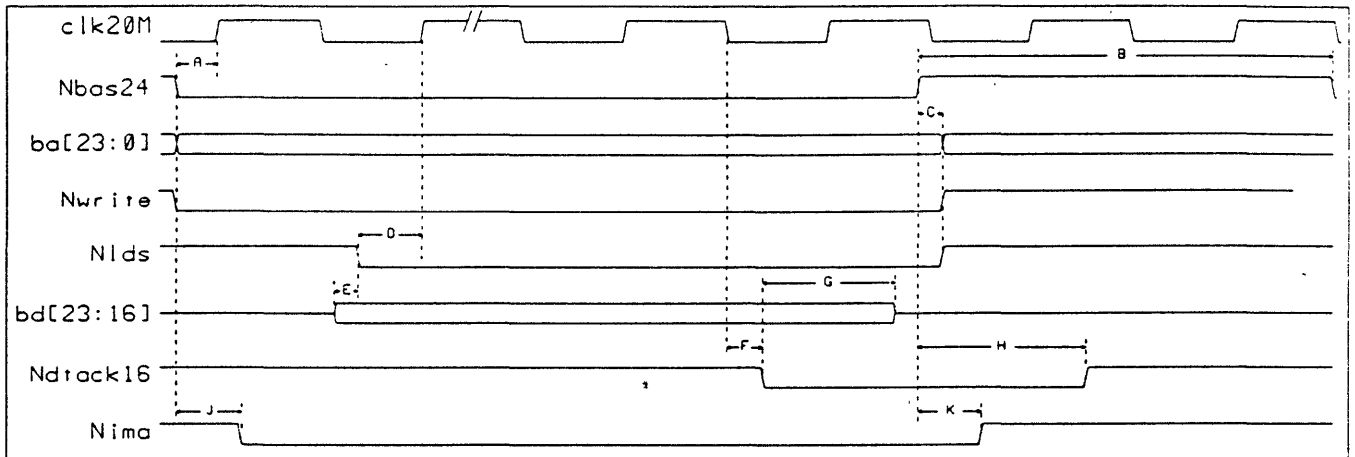
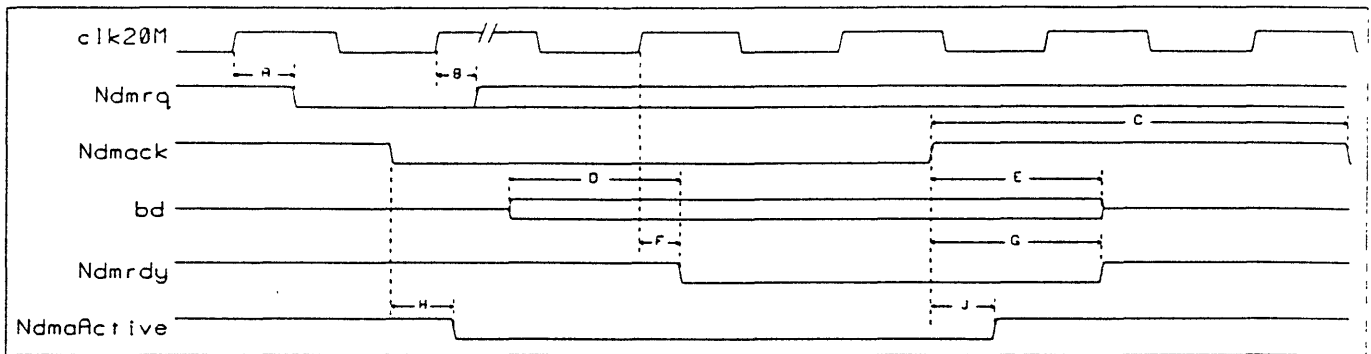


Figure 4. DIO Write Cycle Timing

REF	Parameter	MAX	MIN	NOTE
A	Nbas24 setup to clk20M (tester ref)		12.8ns	
A	Nbas24 setup to clk16or20M (tester ref)		16.4ns	
B	Nbas24 high time		125ns	
C	Address and write hold from Nbas24 or Nlds		15ns	
D	Nlds setup (tester ref)		6.4ns	
E	Data setup to Nlds		0ns	
F	Clock falling to Ndtack16 falling	13ns	0ns	
G	Data hold from Ndtack16 falling		50ns	
H	Ndtack16 release after Nbas deassertion	15ns	0ns	
J	Nima assertion after Nbas assertion	17ns	0ns	
K	Nima deassertion after Nbas deassertion	22ns	0ns	



**Figure 5. DIO Inbound DMA Cycle Timing**

REF	Parameter	MAX	MIN	NOTE
A	Clock to Ndmrq falling (tester ref)	21ns		
B	Clock to Ndmrq rising (tester ref)	21ns		
C	Ndmack high time		100ns	
D	Data set-up to Ndmrdy assertion		75ns	
E	Data tristate after Ndmack deassertion	40ns		
F	Clock falling to Ndmrdy falling (tester ref)	16ns		
G	Ndmrdy release after Ndmack deassertion	12ns		
H	NdmaActive falling from Ndmack assertion	17ns		
J	NdmaActive rising from Ndmack deassertion	15ns		

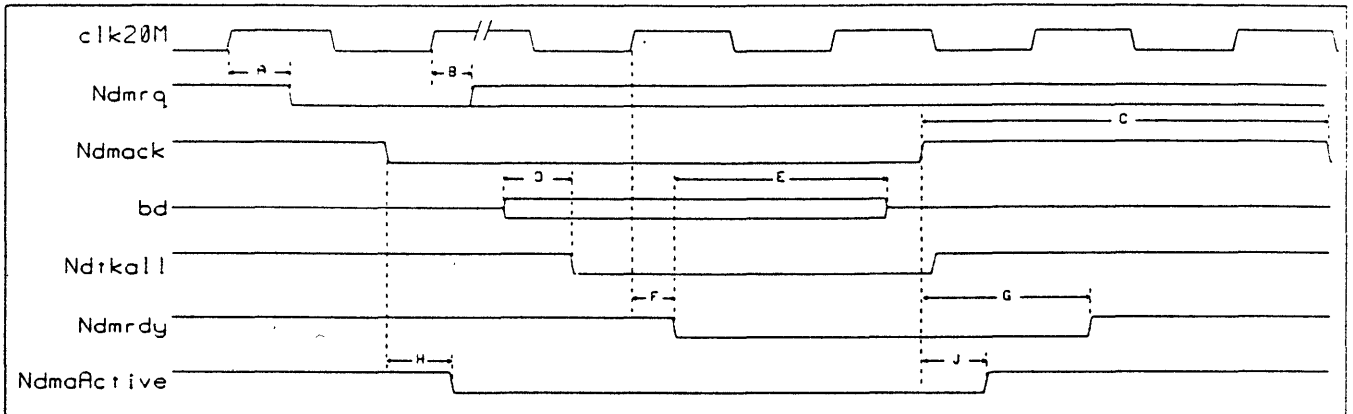


Figure 6. DIO Outbound DMA Cycle Timing

REF	Parameter	MAX	MIN	NOTE
A	Clock to Ndmrq falling (tester ref)	21ns		
B	Clock to Ndmrq rising (tester ref)	21ns		
C	Ndmack high time		100ns	
D	Data set-up to Ndtkall assertion longword mode		-75ns	
D	Data set-up to Ndtkall assertion word/byte mode		0ns	
E	Data hold after Ndmrdy assertion		85ns	
F	Clock falling to Ndmrdy falling (tester ref)	16ns		
G	Ndmrdy release after Ndmack deassertion	12ns		
H	NdmaActive falling from Ndmack assertion	17ns		
J	NdmaActive rising from Ndmack deassertion	15ns		

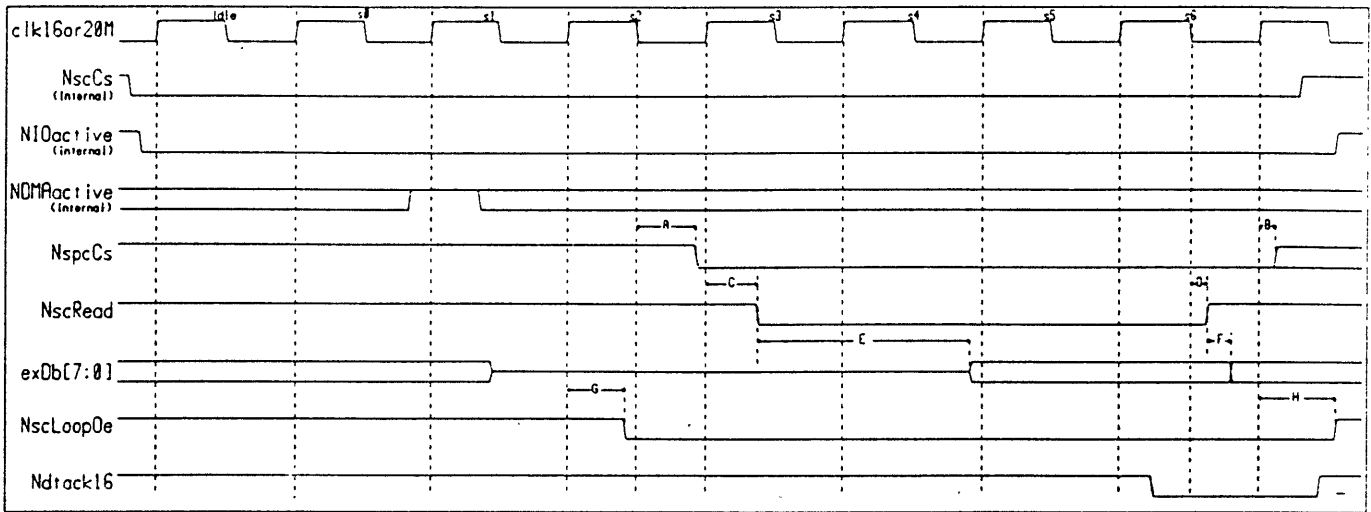
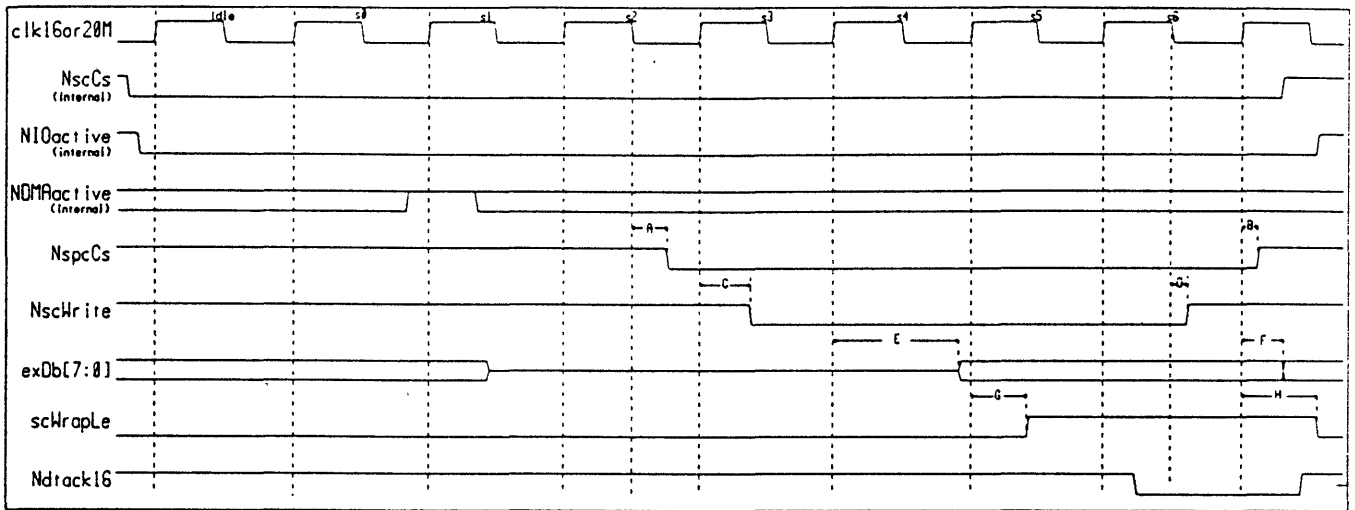


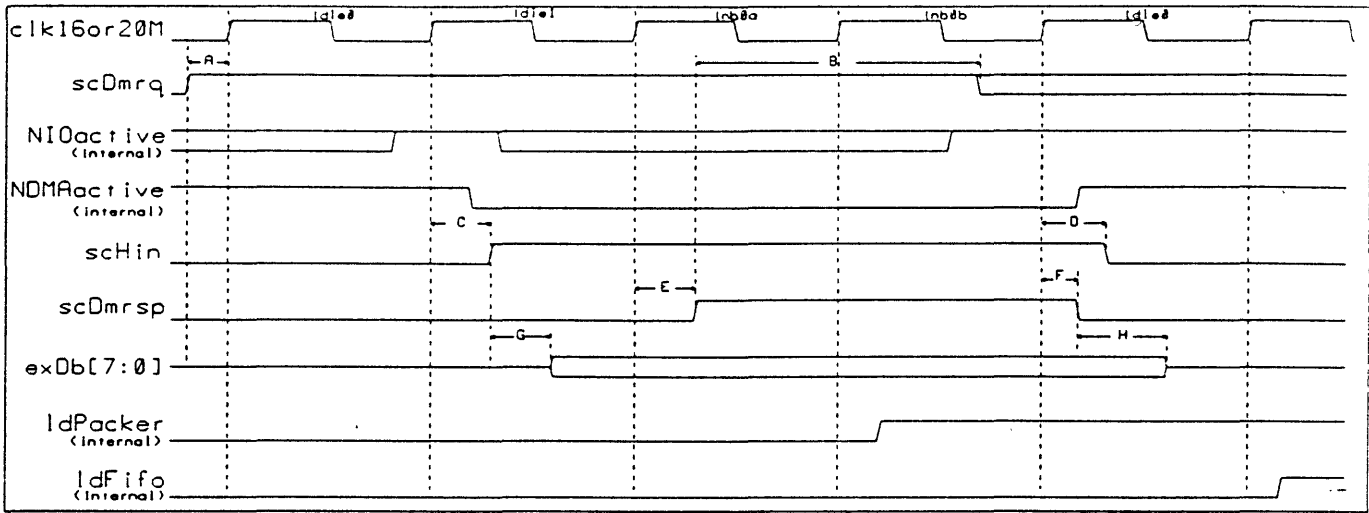
Figure 7. SCSI Read Cycle Timing

REF	Parameter	MAX	MIN	NOTE
A	Clock falling to NspcCs falling (tester ref)	19ns		
B	Clock rising to NspcCs rising (tester ref)	19ns		
C	Clock rising to NscRead falling (tester ref)	19ns		
D	Clock falling to NscRead rising (tester ref)	17ns		
E	NscRead to exDb data valid	70ns		
F	exDb data hold from NscRead rising		0ns	
G	Clock to NscLoopOe falling (tester ref)	25ns		
H	Clock to NscLoopOe rising (tester ref)	25ns		



**Figure 8. SCSI Write Cycle Timing**

REF	Parameter	MAX	MIN	NOTE
A	Clock falling to NspcCs falling (tester ref)	19ns		
B	Clock rising to NspcCs rising (tester ref)	19ns		
C	Clock rising to NscWrite falling (tester ref)	19ns		
D	Clock falling to NscWrite rising (tester ref)	17ns		
E	Clock to exDb data valid (tester ref)	0ns		
F	Clock to exDb data change (tester ref)	50ns		
G	Clock to scWrapLe rising (tester ref)	22ns		
H	Clock to scWrapLe falling (tester ref)	22ns		



**Figure 9. SCSI-FIFO Inbound DMA Cycle Timing**

REF	Parameter	MAX	MIN	NOTE
A	scDmrq set-up to clock (tester ref)		9ns	
B	scDmrsp assertion to deassertion scDmrq	70ns	0ns	
C	Clock to scHin rising (tester ref)	25ns	0ns	
D	Clock to scHin falling (tester ref)	22ns	0ns	
E	Clock to scDmrsp rising (tester ref)	20ns		
F	Clock to scDmrsp falling (tester ref)	20ns		
G	scHin rising to data valid	40ns		
H	scDmrsp falling to data change	90ns	10ns	

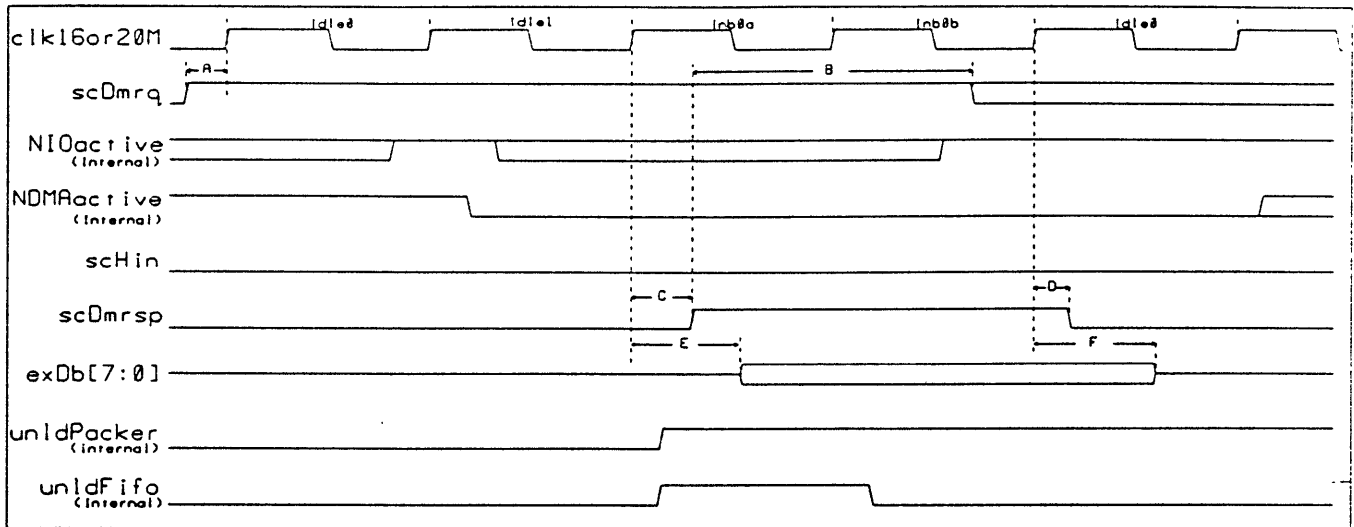
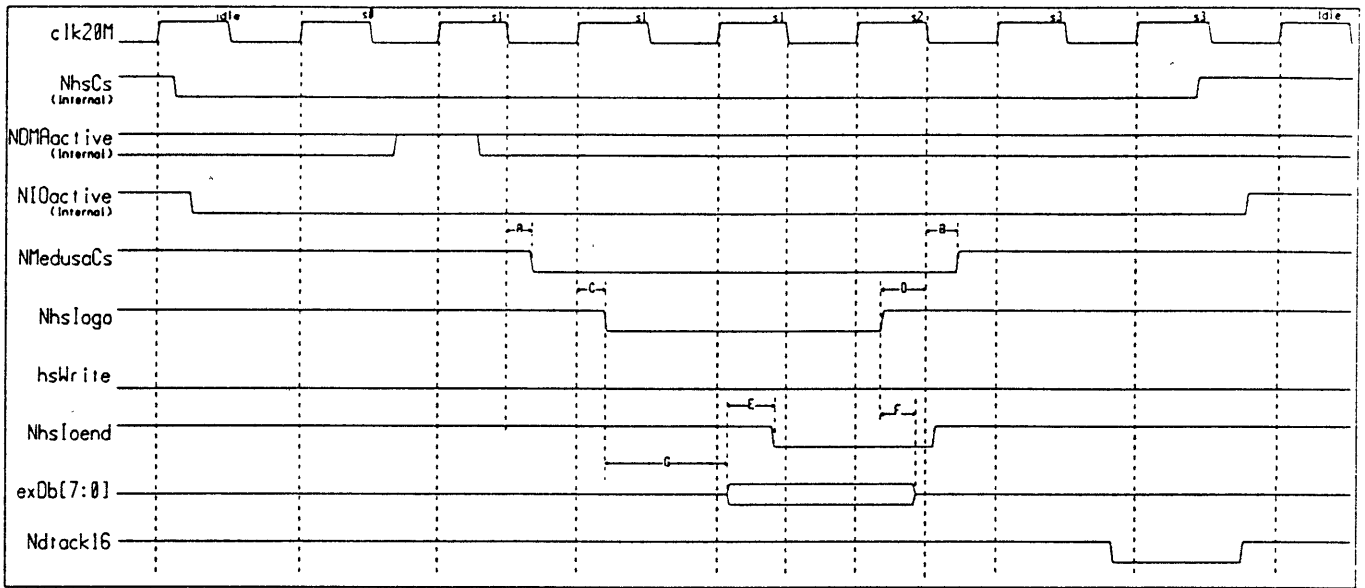


Figure 10. SCSI-FIFO Outbound DMA Cycle Timing

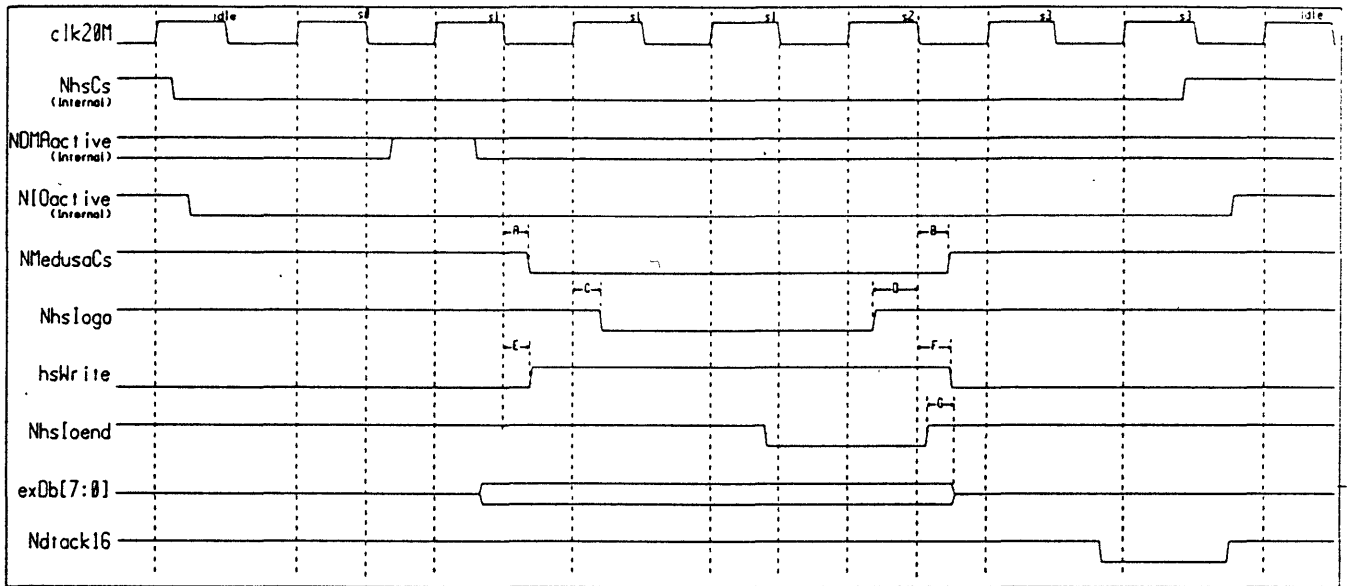
REF	Parameter	MAX	MIN	NOTE
A	scDmrq set-up to clock (tester ref)		9ns	
B	scDmrsp assertion to deassertion scDmrq	70ns	0ns	
C	Clock to scDmrsp rising (tester ref)	20ns		
D	Clock to scDmrsp falling (tester ref)	20ns		
E	Clock to data valid (tester ref)	30ns		
F	Clock to data tristate (tester ref)	27ns		



**Figure 11. HS-HPiB Read Cycle Timing**

REF	Parameter	MAX	MIN	NOTE
A	clk20M falling to NMedusaCs falling	17ns		
B	clk20M falling to NMedusaCs rising	15ns		
C	clk20M rising to NhsIogo falling	28ns		
D	clk20M rising to NhsIogo rising	27ns		
E	Data setup to NhsIoend falling		-30ns	
F	Data hold after NhsIogo rising		0ns	
G	Data valid from NhsIogo falling	140ns		





**Figure 12. HS-HPIB Write Timing**

REF	Parameter	MAX	MIN	NOTE
A	clk20M falling to NMedusaCs falling	17ns		
B	clk20M falling to NMedusaCs rising	15ns		
C	clk20M rising to NhsIogo falling	28ns		
D	clk20M rising to NhsIogo rising	27ns		
E	clk20M falling to hsWrite rising	19ns		
F	clk20M falling to hsWrite falling	18ns		
G	exDb[7:0] hold from NhsIoend rising		0ns	

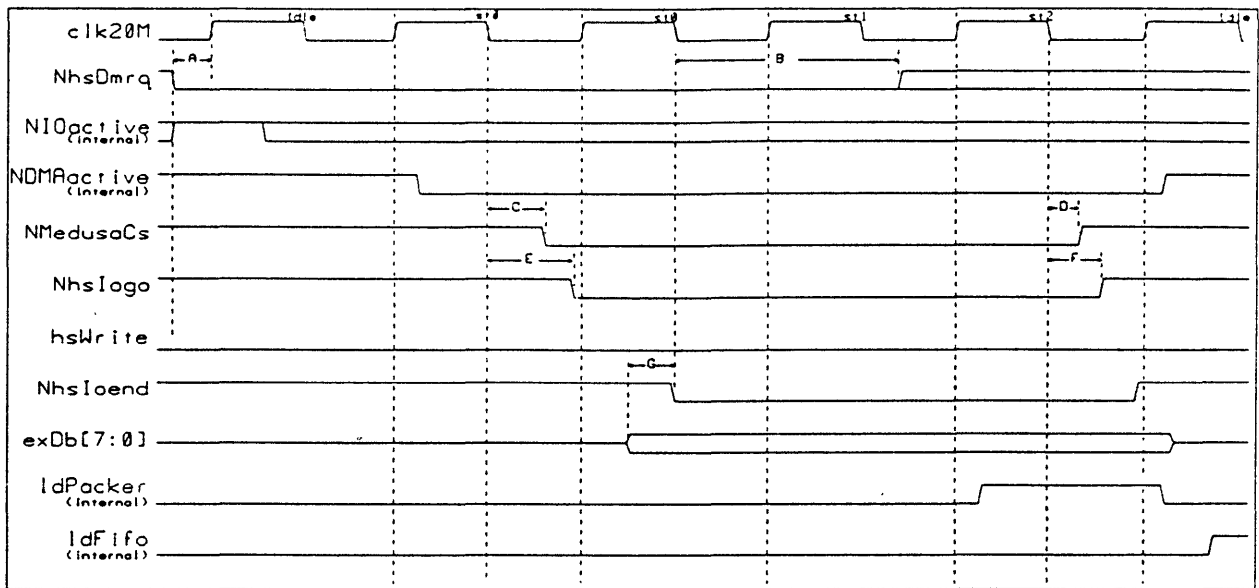


Figure 13. HS-HPiB Inbound DMA Cycle Timing

REF	Parameter	MAX	MIN	NOTE
A	NhsDmrq setup to clk20M (tester ref)		10ns	
B	NhsIoend falling to NhsDmrq rising	150ns		
C	clk20M falling to NMedusaCs falling	18ns		
D	clk20M falling to NMedusaCs rising	17ns		
E	clk20M falling to NhsIogo falling	34ns		
F	clk20M falling to NhsIogo rising	31ns		
G	Data setup to NhsIoend falling		-30ns	

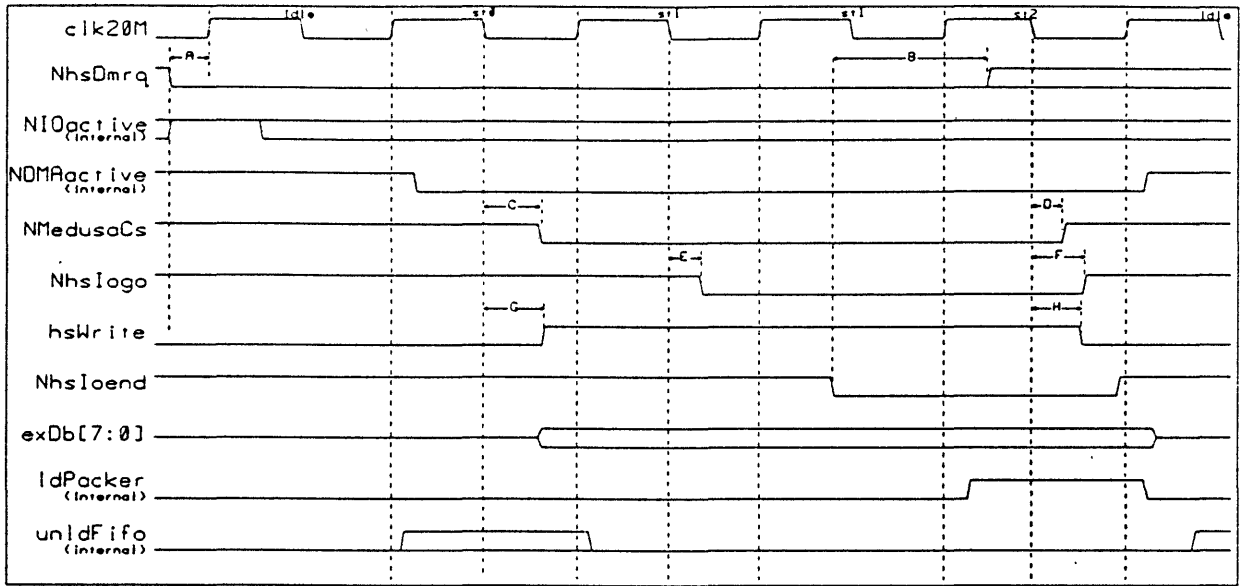
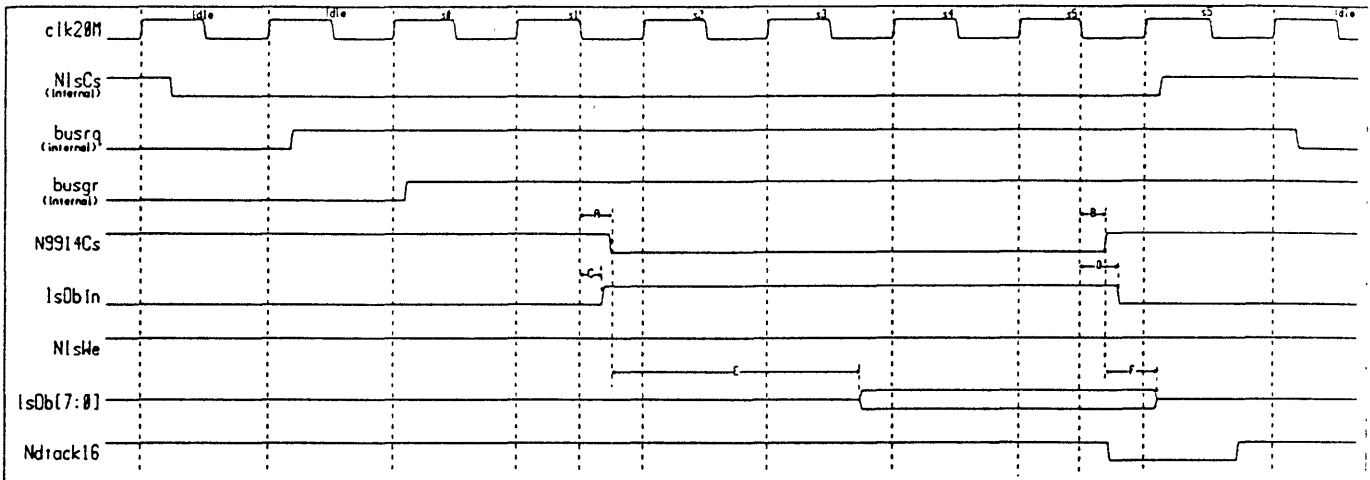


Figure 14. HS-HPiB Outbound DMA Cycle Timing

REF	Parameter	MAX	MIN	NOTE
A	NhsDmrq setup to clk20M (tester ref)		10ns	
B	NhsIoend falling to NhsDmrq rising	150ns		
C	clk20M falling to NMedusaCs falling	18ns		
D	clk20M falling to NMedusaCs rising	17ns		
E	clk20M falling to NhsIogo falling	34ns		
F	clk20M falling to NhsIogo rising	31ns		
G	clk20M falling to hsWrite rising	19ns		
H	clk20M falling to hsWrite falling	18ns		



**Figure 15. LS-HPIB Read Cycle Timing**

REF	Parameter	MAX	MIN	NOTE
A	clk20M falling to N9914Cs falling	23ns		
B	clk20M falling to N9914Cs rising	19ns		
C	clk20M falling to lsDb in rising	13ns		
D	clk20M falling to lsDb in falling	12ns		
E	data valid from N9914Cs falling	150ns		
F	data tristate from lsDb in rising	100ns		

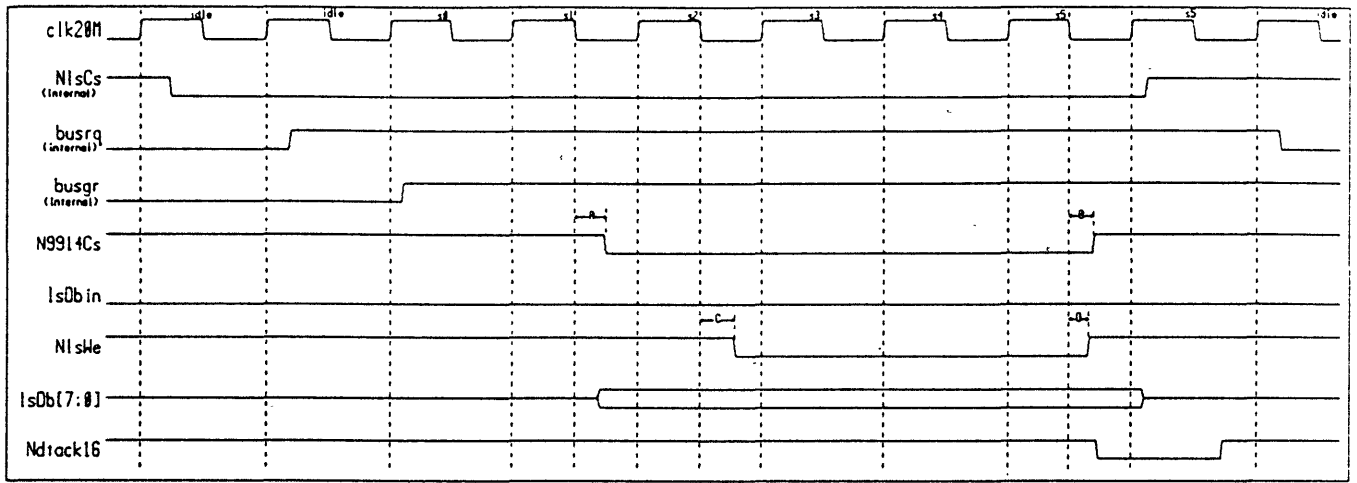


Figure 16. LS-HPiB Write Cycle Timing

REF	Parameter	MAX	MIN	NOTE
A	clk20M falling to N9914Cs falling	23ns		
B	clk20M falling to N9914Cs rising	19ns		
C	clk20M falling to N1sWe falling	13ns		
D	clk20M falling to N1sWe rising	11ns		

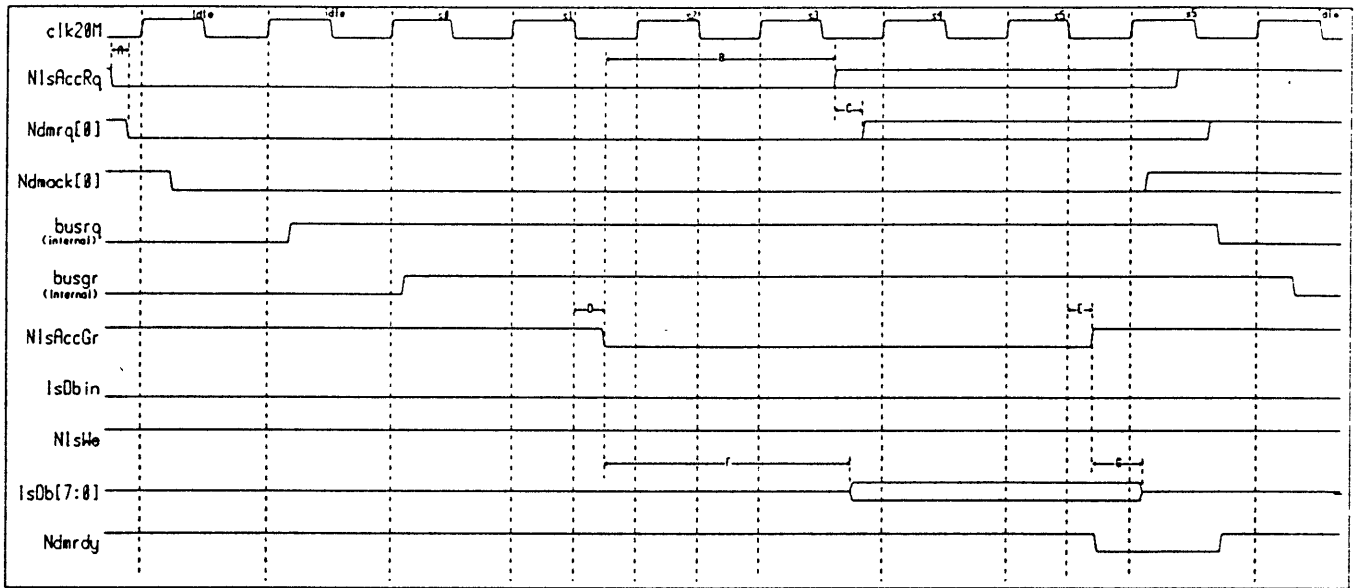


Figure 17. LS-HPIB Inbound DMA Cycle Timing

REF	Parameter	MAX	MIN	NOTE
A	NlsAccRq falling to Ndmrq[0] falling	12ns		
C	NlsAccRq rising to Ndmrq[0] rising	11ns		
D	clk20M falling to NlsAccGr falling	23ns		
E	clk20M falling to NlsAccGr rising	19ns		
F	Data valid from NlsAccGr falling	150ns		
G	Data tristate from NlsAccGr rising	100ns		

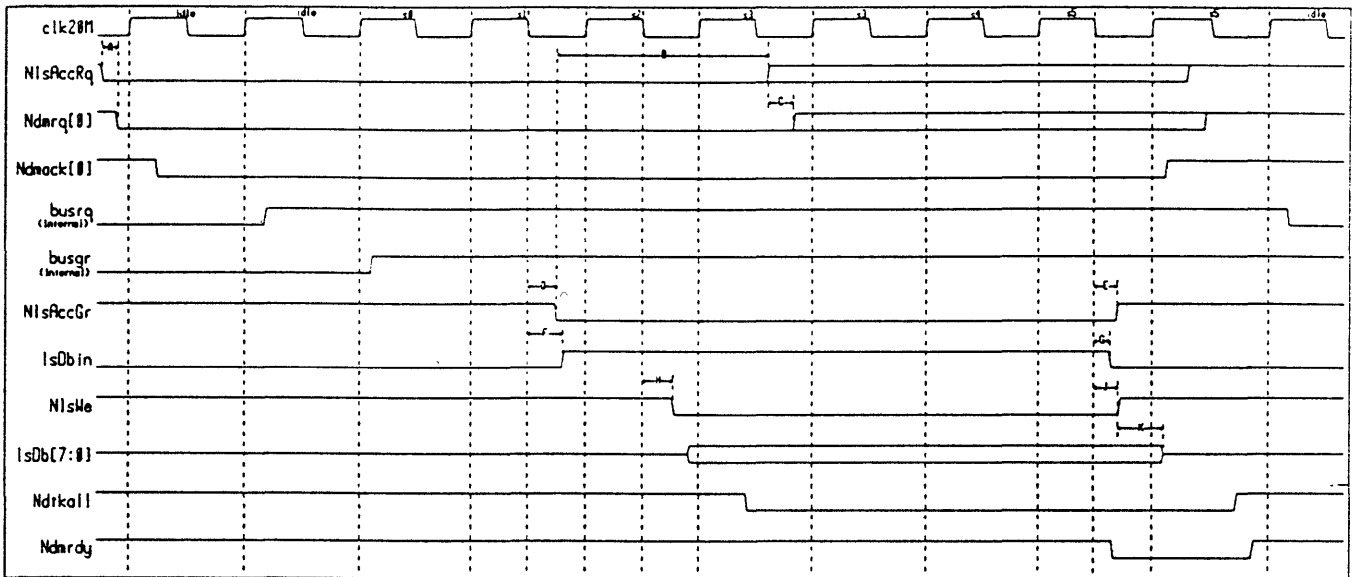
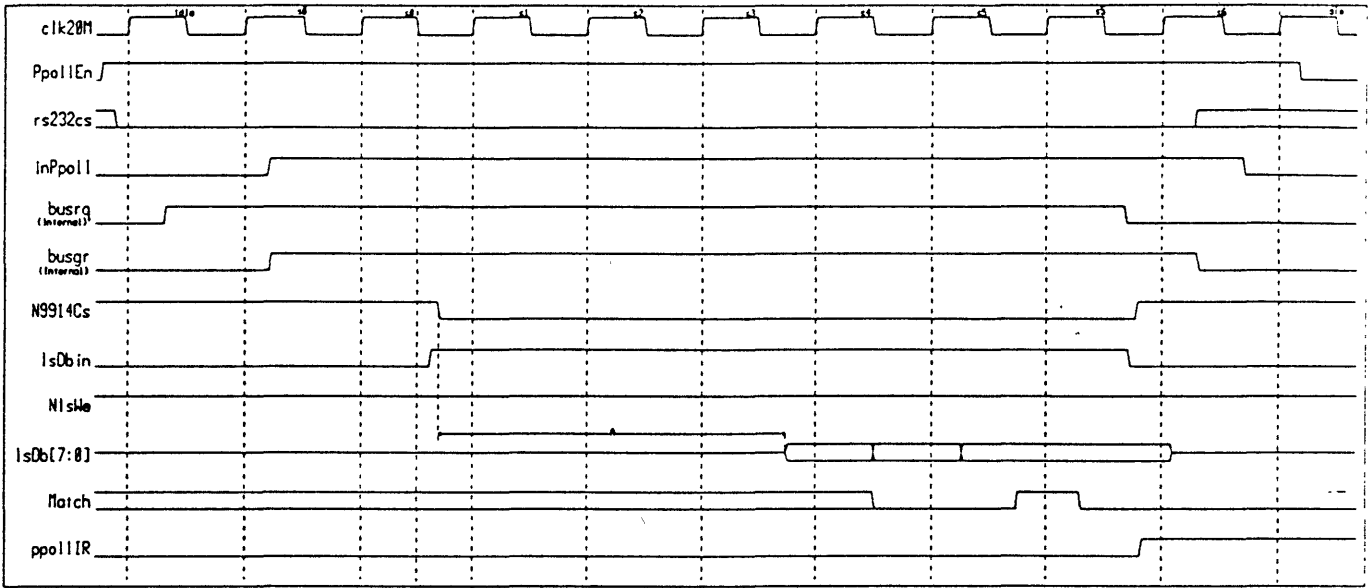


Figure 18. LS-HPIB Outbound DMA Cycle Timing

REF	Parameter	MAX	MIN	NOTE
A	NlsAccRq falling to Ndmrq[0] falling	12ns		
B	NlsAccGr falling to NlsAccRq rising	100ns		
C	NlsAccRq rising to Ndmrq[0] rising	11ns		
D	clk20M falling to NlsAccGr falling	23ns		
E	clk20M falling to NlsAccGr rising	19ns		
F	clk20M falling to lsDbin rising	13ns		
G	clk20M falling to lsDbin falling	12ns		
H	clk20M falling to NlsWe falling	13ns		
J	clk20M falling to NlsWe rising	11ns		
K	Data hold from NlsWe rising		0ns	



**Figure 19. LS-HPiB Parallel Poll Cycle Timing**

REF	Parameter	MAX	MIN	NOTE
A	N9914Cs falling to data valid	150ns		



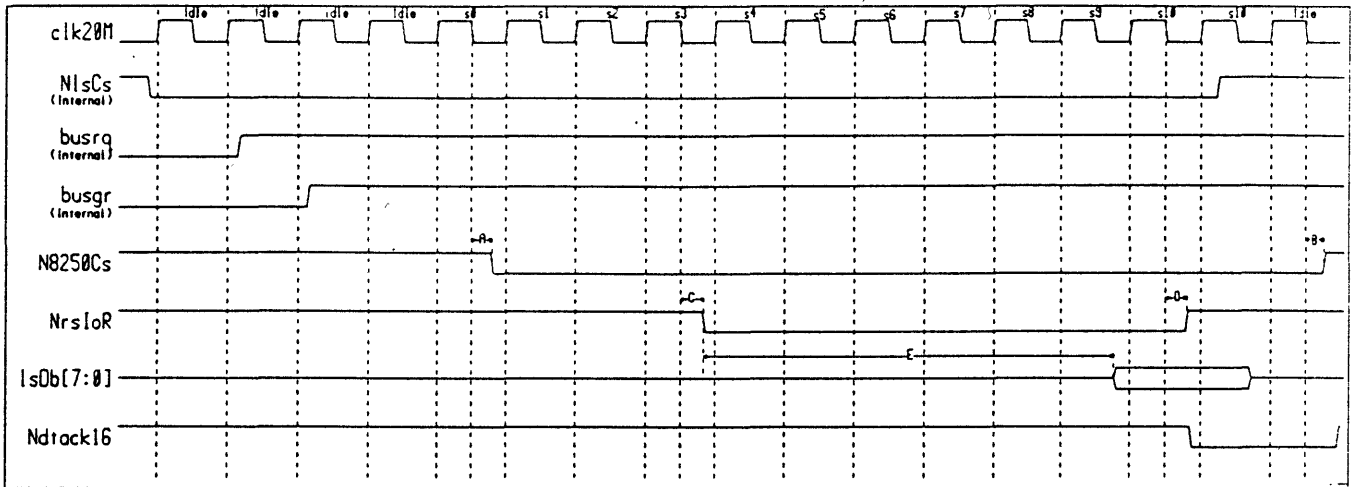
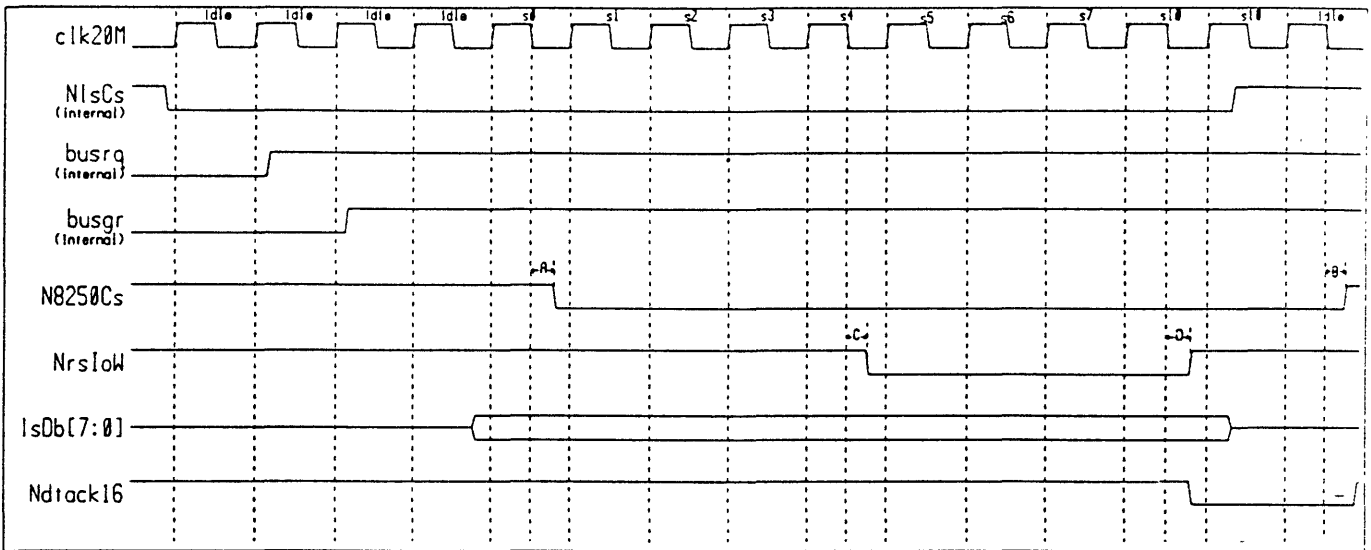


Figure 20. RS-232 Read Cycle Timing

REF	Parameter	MAX	MIN	NOTE
A	clk20M falling to N8250Cs falling	13ns		
B	clk20M falling to N8250Cs rising	12ns		
C	clk20M falling to NrsIoR falling	14ns		
D	clk20M falling to NrsIoR rising	13ns		
E	NrsIoR falling to data valid	300ns		



**Figure 21. RS-232 Read Cycle Timing**

REF	Parameter	MAX	MIN	NOTE
A	clk20M falling to N8250Cs falling	13ns		
B	clk20M falling to N8250Cs rising	12ns		
C	clk20M falling to NrsIoW falling	14ns		
D	clk20M falling to NrsIoW rising	13ns		

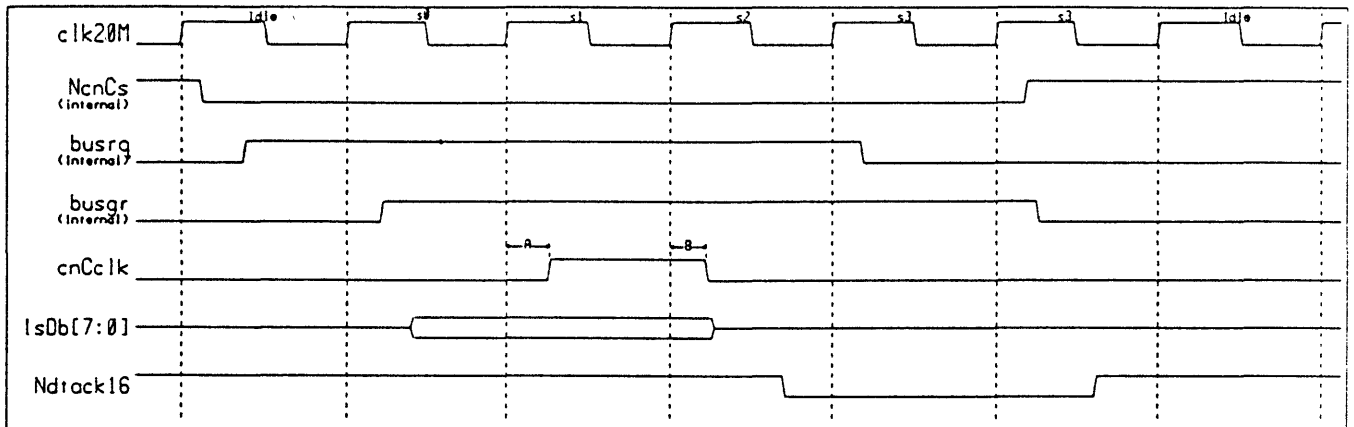
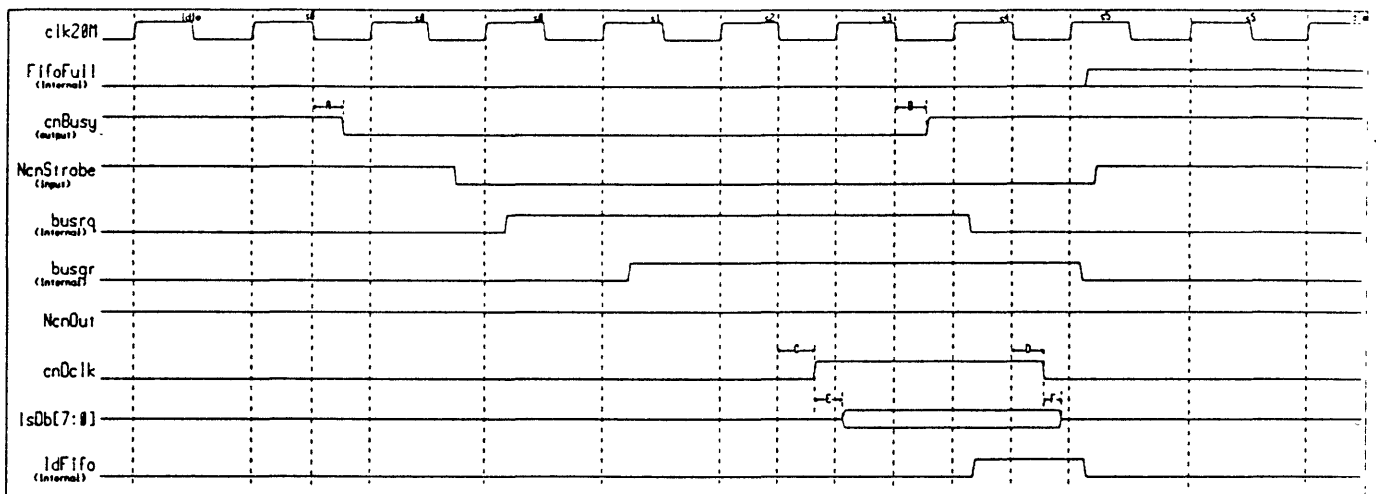


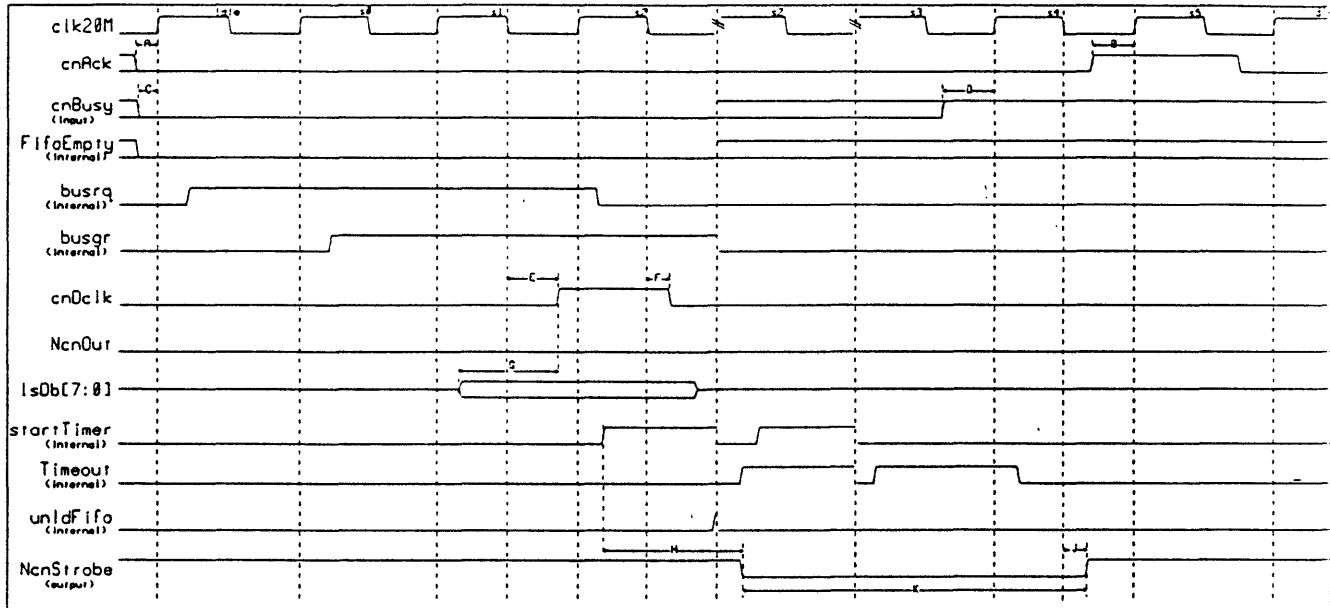
Figure 22. Centronics Write Register 7 Cycle Timing

REF	Parameter	MAX	MIN	NOTE
A	clk20M rising to cnCclk rising	22ns		
B	clk20M rising to cnCclk falling	23ns		



**Figure 23. Centronics Back-end Inbound Cycle Timing**

REF	Parameter	MAX	MIN	NOTE
A	clk20M falling to cnBusy falling	15ns		
B	clk20M falling to cnBusy rising	16ns		
C	clk20M falling to cnDclk rising	13ns		
D	clk20M falling to cnDclk falling	12ns		
E	cnDclk rising to data valid	30ns		
F	cnDclk falling to data tri-state	30ns		



**Figure 24. Centronics Back-end Outbound Cycle Timing**

REF	Parameter	MAX	MIN	NOTE
A	cnAck falling to clk20M falling (tester ref)	7ns		
B	cnAck rising to clk20M falling (tester ref)	7ns		
C	cnBusy falling to clk20M falling (tester ref)	7ns		
D	cnBusy rising to clk20M falling (tester ref)	7ns		
E	clk20M falling to cnDclk rising	13ns		
F	clk20M falling to cnDclk falling	12ns		
G	data setup to cnDclk rising		20ns	
H	Centronics data Setup to NcnStrobe		700ns	
J	clk20M falling to NcnStrobe falling	13ns		
K	NcnStrobe low time		800ns	

## *2.6 Packaging*

The -0302 part is in a EIAJ 160 pin plastic quad flat pack, for more information contact NID.

## *2.7 Mechanical Characteristics*

The packaged part will meet or exceed all HP class B environmental specifications.

## *2.8 Test Considerations*

Contact NID for tester configuration, specification and operation. Nikki is currently tested on a Sentry 15 IC tester. All of the tests were developed with Guide and exist on the same tapes as the schematics.

### 3. Software ERS and Theory of Operation

#### 3.1 Software Interface

Nikki is functionally 4 independent devices that share the DIO-II bus. Each has a standard register set, a hidden register set and a select code register set. The hidden register set contains configuration information that the system or user can set which previous designs set using switches. On the first product using Nikki (375, 345), an EEPROM contains the "switch" settings for each interface, among other things. Thus, a user can configure his system uniquely without disassembling his machine, and the configuration is non-volatile. It is the bootrom's responsibility to configure Nikki if needed.

TABLE 12. Select code register map

Address (HEX)	Interface	Nikki	Default Select Code (Decimal)	Value (HEX)
0x400013	LS-HPIB	1	internal 7	0x47
0x400015	SCSI/HS-HPIB	1	14	0x6E
0x400017	Centronics	1	23	0x77
0x40001B	RS-232	1	9	0x69
0x40001D	SCSI/HS-HPIB	2	24	0x78
0x40001F	Centronics	2	17	0x71
0x400023	RS-232	2	10	0x6A
0x400025	SCSI/HS-HPIB	3	26	0x7A
0x400027	Centronics	3	19	0x73
0x40002B	RS-232	3	15	0x6F
0x40002D	SCSI/HS-HPIB	4	30	0x7E
0x40002F	Centronics	4	31	0x7F
0x400033	RS-232	4	16	0x70

Each select code except for LS-HPIB can be modified by writing the current value into the select code register. This enables hidden registers for that device and enables the select code register to be modified. The value to be written is the upper 8 bits of a 24 bit address, which is the value 0x60 logically OR'd with a 5 bit select code value from 0 to 32. The address space of the interface is 0xSC0000 through 0xSCFFFF (Where SC is the upper 8 bits of address). LS-HPIB's hidden mode can be enabled by writing a 0x47 to address 0x400013. This is internal select code 7 and its value can not be modified.

When hidden mode is enabled (writing the current value back into the select code register), the normal register set is replaced by a hidden register set. Any time the select code register is read, hidden mode is disabled.

If an interface does not exist (or is disabled), Nikki will not assert *NDTACK16* which will cause a bus error.

### 3.1.1 SCSI

TABLE 13. SCSI Standard Register Set

Address (HEX)	Name		Contents							
			7	6	5	4	3	2	1	0
0x1	ID	read	R/L	E1	E0	0	0	1	1	1
0x1	Reset	write	X	X	X	X	X	X	X	X
0x3	Status	read	IE	IR	IL1	IL0	L/W	I/O	DE1	DE0
0x3	Control	write	IE	X	X	X	L/W	I/O	DE1	DE0
0x5	Loopback	read	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0
0x5	Wraparound	write	REQ	ACK	X	X	BSY	MSG	C/D	I/O
0x7	Configuration	read	TP	R1	R0	SD	P	SA2	SA1	SA0
0x7	Flush	write	X	X	X	X	X	X	X	X
0x9	Packer Status	read	0	8/10	DREQ	EID	0	PV2	PV1	PV0
0x9	Packer Status	write	X	X	X	EID	X	X	X	X
0xB	Packer Byte 0	read	D7	D6	D5	D4	D3	D2	D1	D0
0xB	Packer Byte 0	write	X	X	X	X	X	X	X	X
0xD	Packer Byte 1	read	D7	D6	D5	D4	D3	D2	D1	D0
0xD	Packer Byte 1	write	X	X	X	X	X	X	X	X
0xF	Packer Byte 2	read	D7	D6	D5	D4	D3	D2	D1	D0
0xF	Packer Byte 2	write	X	X	X	X	X	X	X	X
0x10	Byte DMA	read	D7	D6	D5	D4	D3	D2	D1	D0
0x10	Byte DMA	write	X	X	X	X	X	X	X	X
0x21	SPC Reg 0	BDID								
0x23	SPC Reg 1	SCTL								
0x25	SPC Reg 2	SCMD								
0x27	SPC Reg 3	TMOD								
0x29	SPC Reg 4	INTS								
0x2B	SPC Reg 5	PSNS								
0x2D	SPC Reg 6	SSTS								
0x2F	SPC Reg 7	SERR								
0x31	SPC Reg 8	PCTL								
0x33	SPC Reg 9	MBC								
0x35	SPC Reg A	DREG								
0x37	SPC Reg B	TEMP								
0x39	SPC Reg C	TCH								
0x3B	SPC Reg D	TCM								
0x3D	SPC Reg E	TCL								

#### 3.1.1.1 SCSI ID and Reset Register (1)

R/L Remote/Local: This is always 0 meaning LOCAL.  
 E1 Set if SCSI is differential drive; clear if single-ended  
 E0 Set if 16-bit DMA only; clear if 16 and 32 bit DMA capability  
 00111 Primary ID



† A write of any value to register 1 will generate a hardware reset.

### 3.1.1.2 SCSI Status and Control Register (3)

IE	Interrupt Enable: set to enable interrupts.															
IR	Set if SCSI is currently generating an interrupt (or would be if the <i>IE</i> bit were set).															
IL1,IL0	Indicate the interrupt level. <table border="0" style="margin-left: 40px;"> <thead> <tr> <th>IL1</th> <th>IL0</th> <th>Level</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>3</td> </tr> <tr> <td>0</td> <td>1</td> <td>4</td> </tr> <tr> <td>1</td> <td>0</td> <td>5</td> </tr> <tr> <td>1</td> <td>1</td> <td>6</td> </tr> </tbody> </table>	IL1	IL0	Level	0	0	3	0	1	4	1	0	5	1	1	6
IL1	IL0	Level														
0	0	3														
0	1	4														
1	0	5														
1	1	6														
L/W	Longword/Word: If set, select 32 bit DMA operations, otherwise 16 bit transfers. Should be set only if 32-bit capability was indicated in ID register (1).															
I/O	DMA direction: Set indicates an inbound transfer, otherwise, an outbound transfer. This bit must be properly set by controlling software prior to initiating a DMA transfer.															
DE1,DE0	DMA enables for channel 1 and 0. Only one may be set at a time. Once these are set, <i>Ndmrq</i> will be asserted, and DMA to/from the 87033 will begin as soon as possible. Unlike HS-HPIB, there is no loss of data if the DE0/1 bits are toggled during DMA. i.e. switched from channel 1 to channel 0.															
†	All bits except <i>IL1,IL0</i> are cleared by a software or hardware reset.															

### 3.1.1.3 SCSI Wraparound and Loopback Data (5)

SD[7:0]	SCSI data bus bit 7-0. Used to sample the SCSI data bus during testing/troubleshooting.
REQ	SCSI control signal <i>REQ</i> .
ACK	SCSI control signal <i>ACK</i> .
BSY	SCSI control signal <i>BSY</i> .
MSG	SCSI control signal <i>MSG</i> .
C/D	SCSI control signal <i>C/D</i> .
I/O	SCSI control signal <i>I/O</i> .
†	Writing to this register can aid in testing/troubleshooting. Settings within this register will override the other SCSI drivers. During normal operation this register should have had a 0 written to it.

### 3.1.1.4 SCSI Flush and Hardware Configuration register (7)

TP	Terminator Power: If set, termination resistors are properly powered.
R1,R0	Information on maximum synchronous transfer rate that should be used in synchronous negotiations
	8MHz mode
	R1 R0 32-bit mode 16-bit mode
	0 0 4.00 mbps 2.00 mbps
	0 1 2.67 mbps 1.60 mbps
	1 0 2.00 mbps Sync not recommended
	1 1 sync not available
	10MHz mode
	0 0 5.00 mbps 2.50 mbps
	0 1 3.33 mbps 2.00 mbps
	1 0 2.50 mbps Sync not recommended
	1 1 sync not available
SD	ShutDown: Indicates that Bus drivers have shut down if set. Always returns a zero value.
P	Parity selected: This bit should be read and then written out to the SPC <i>SCTL</i> register bit 3. It is set via a hidden register.
SA2,SA1,SA0	Inverted SCSI bus address 0-7. These three bits must be read, inverted, and written out to the SPC <i>BDID</i> register. These bits are set via a hidden register.
†	Writing to this register will flush any DMA data within the ASIC, reset the PACKER and all state machines. This may be used to recover after a peripheral disconnects from the bus in the middle of an outbound transfer.

### 3.1.1.5 SCSI Packer and Misc Status Register (9)

8/10	8MHz/10MHz mode. If clear, Fujitsu is running at 8MHz, otherwise 10MHz. This bit can be set only by hidden register 3.
DREQ	Fujitsu DMA request line. This bit <i>must</i> be set before the <i>Byte DMA</i> register is read.
EID	Enhanced ID. This bit is read/writable only when SCSI registers 0x9-0x10 exist. Previous SCSI implementations do not include these enhanced capabilities and will not have this bit read/writable.
PV2	Packer valid bit 2. If set, Packer byte 2, 1 and 0 are valid, and subsequent reads from registers 0xF, 0xD, and 0xB will return their data. This bit is only valid at the end of an inbound DMA transfer. It will be set if the Fujitsu transferred an odd number of bytes into Nikki.

- PV1 Packer valid bit 1. If set, Packer byte 1 and 0 are valid, and subsequent reads from register 0xD, and 0xB will return their data. This bit is only valid at the end of an inbound DMA transfer. It will be set if the Fujitsu didn't transfer an aligned (DMA width) amount of data into Nikki.
- PV0 Packer valid bit 0. If set, Packer byte 0 is valid, and a subsequent read from register 0xB will return its data. This bit is only valid at the end of an inbound DMA transfer. It will be set if the Fujitsu didn't transfer an aligned (DMA width) amount of data into Nikki.
- † A software or hardware reset will clear EID, PV2, PV1, and PV0.

#### *3.1.1.6 SCSI Packer Byte 0 Register (B)*

- D7-0 Packer byte 0 data. This data is valid only if PV2-0 indicates so. It will be valid if the Fujitsu transferred less than the DMA width on the last inbound DMA transfer. This byte will be "stuck" in the packer waiting for more Fujitsu to FIFO dma transfers to fill the packer. Use register 7 to clear the packer contents.

#### *3.1.1.7 SCSI Packer Byte 1 Register (D)*

- D7-0 Packer byte 1 data. This data is valid only if PV2-0 indicates so. It will be valid if the Fujitsu transferred only two or three bytes instead of four on the last inbound DMA transfer. Use register 7 to clear the packer contents.

#### *3.1.1.8 SCSI Packer Byte 2 Register (F)*

- D7-0 Packer byte 2 data. This data is valid only if PV2-0 indicates so. It will be valid if the Fujitsu transferred only three bytes instead of four on the last inbound DMA transfer. All three bytes will be "stuck" in the packer waiting for a forth. Use register 7 to clear the packer contents.

#### *3.1.1.9 SCSI Byte DMA Register (0x10)*

D7-0 A read from this register will perform a single byte DMA xfer from the Fujitsu and return the data. A read from this register can be performed *only* when the DE1,DE0 bits are zero, the I/O bit is set and the DREQ bit is set. This register is used for aligning DMA xfers if a peripheral disconnected on an unaligned boundary.

TABLE 14. SCSI Hidden Register Set

Address (HEX)		Contents							
		7	6	5	4	3	2	1	0
0x1	read	0	0	IL1	IL0	BD2	BD1	BD0	P
0x1	write	0	0	IL1	IL0	BD2	BD1	BD0	P
0x3	read	0	0	0	8/10	0	0	R1	R0
0x3	write	0	0	0	8/10	0	0	R1	R0

### 3.1.1.10 SCSI Hidden Register 1

IL1,IL0 Interrupt Level. Same as standard register 1.  
 BD2,BD1,BD0 Inverted bus address. Same as standard register 7.  
 P Enable, disable parity checking. Same as standard register 7.  
 † The default value is 0x11. This register is only reset when *NPuReset* is asserted.

### 3.1.1.11 SCSI Hidden Register 3

8/10 Sets clock frequency that drives the 87033 to 8MHz when clear or 10MHz when set.  
 R1,R0 Max synchronous transfer rate. Same as standard register 7.  
 † The default value is 0. This register is only reset when *NPuReset* is asserted.

### 3.1.2 HS-HPIB

There are actually two separate HS-HPIB standard register sets. One is compatible with the 98625B. The other adds 32-bit DMA transfer capability. The selection of one of the two sets is through a hidden register. The ID register uniquely identifies the two interfaces.

TABLE 15. HS-HPIB Register Set (98625B compatible)

Address (HEX)	Name		Contents							
			7	6	5	4	3	2	1	0
1	ID	read	R/L	0	0	0	1	0	0	0
1	Reset	write	X	X	X	X	X	X	X	X
3	Status	read	IE	IR	IL1	IL0	O/I	W/B	DE1	DE0
3	Control	write	IE	X	X	X	O/I	W/B	DE1	DE0
5	EOI	write	X	X	X	X	X	X	X	EOI
7	DMA Status	write	X	X	X	X	X	X	X	BYTE
11	Medusa Reg 0	INTR								
13	Medusa Reg 1	INTM								
15	Medusa Reg 2	DMA								
17	Medusa Reg 3	STS								
19	Medusa Reg 4	CTL								
1B	Medusa Reg 5	ADR								
1D	Medusa Reg 6	PPM								
1F	Medusa Reg 7	PPS								

TABLE 16. HS-HPIB Register Set (98625C)

Address (HEX)	Name		Contents							
			7	6	5	4	3	2	1	0
1	ID	read	R/L	1	0	0	1	0	0	0
1	Reset	write	X	X	X	X	X	X	X	X
3	Status	read	IE	IR	IL1	IL0	O/I	L/W	DE1	DE0
3	Control	write	IE	X	X	X	O/I	L/W	DE1	DE0
5	EOI	write	X	X	X	X	X	X	X	EOI
7	DMA Status	read	X	X	X	X	X	LC	VB1	VB0
7	DMA Status	write	X	X	X	X	X	X	VB1	VB0
11	Medusa Reg 0	INTR								
13	Medusa Reg 1	INTM								
15	Medusa Reg 2	DMA								
17	Medusa Reg 3	STS								
19	Medusa Reg 4	CTL								
1B	Medusa Reg 5	ADR								
1D	Medusa Reg 6	PPM								
1F	Medusa Reg 7	PPS								

### 3.1.2.1 HS-HPIB ID and Reset register (1)

R/L Remote/Local: This is always 0 meaning LOCAL.  
 0001000 (1001000) Device ID: Value is 8 for **98625B** mode (8/16 bit DMA) or 0x48 for **98625C** mode (16/32 bit DMA).  
 † Writing any value to this register will reset all of the state machines, the FIFO and *hsPon* will be deasserted for 500ns.

### 3.1.2.2 HS-HPIB Status and Control register (3)

IE Interrupt Enable: set to enable interrupts.  
 IR Set if HPIB is currently generating an interrupt (or would be if the *IE* bit were set).  
 IL1,IL0 Indicate the interrupt level read in from the configuration port.  

IL1	IL0	Level
0	0	3
0	1	4
1	0	5
1	1	6

  
 O/I Outbound/Inbound: If set, DMA transfers are outbound, else they are inbound. This bit must be properly set by controlling software prior to initiating a DMA transfer.  
 W/B (L/W) Word/Byte: If set, DMA transfers are 16-bits wide (word), else they are 8-bit wide (byte). **98625B** mode.  
 Longword/Word: If set, DMA transfers are 32-bits wide (longword), else they are 16-bits wide. **98625C** mode.  
 DE1,DE0 DMA enables for channel 1 and 0. Only one may be set at a time, and it must be different than the DMA enables for the SCSI interface.  
 † All of these bits except for IL1/IL0 are cleared during a soft/hard reset

### 3.1.2.3 HS-HPIB EOI Control (5)

EOI End-Of-transfer Interrupt: If set, the *Ndone* signal will not cause an end-of-transfer interrupt. Also, no EOI is included with the last byte of an outbound transfer (to Medusa on *Hb[1:0]*).  
 † This bit is cleared during a soft/hard reset.

### 3.1.2.4 HS-HPIB DMA status (7)

- BYTE** This bit indicates that the last word of data in a 16-bit inbound transfer contained only one byte of valid data. **98625B** mode.
- VB1,VB0** These two bits indicate how many bytes were valid in the last DMA transfer. **98625C** mode.
- | VB1 | VB0 | Valid Bytes |
|-----|-----|-------------|
| 0   | 0   | All         |
| 0   | 1   | 1           |
| 1   | 0   | 2           |
| 1   | 1   | 3           |
- L/C** This bit, if set, indicates that either 16 or 32 bit DMA can be selected. If clear, only 16 bit DMA can be performed. **98625C** mode.
- † All bits except L/C are cleared during a hard/soft reset.

**TABLE 17. HS-HPIB Hidden Register Set**

Address (HEX)		7	6	5	4	3	2	1	0
0x1	read	0	0	IL1	IL0	Sys	F/S	E1	E0
0x1	write	0	0	IL1	IL0	Sys	F/S	E1	E0

### 3.1.2.5 HS-HPIB Hidden Register 1

- IL1,IL0** Interrupt Level. Same as standard register 1.
- Sys** System controller "switch". If set, Medusa is system controller.
- F/S** Fast/slow "switch". If set, a tuned resistor is used by Medusa to optimize HPIB transfers, else a standard value is used.
- E1** Select HS-HPIB interface type. If set, 16/32 bit DMA interface, otherwise a 8/16 bit DMA interface.
- E0** DIO-I/II select. If set, the **98625C** interface can be used in either 16 or 32 bit DMA mode, otherwise only 16 bit DMA mode is possible.
- † The default value is 0x19. This register is only reset when *NPuReset* is asserted.

### 3.1.3 LS-HPIB

TABLE 18. LS-HPIB Standard Register Set

Address (HEX)			Contents							
			7	6	5	4	3	2	1	0
0x3	Status	read	1	INT	0	0	0	0	0	DE
0x3	Control	write	X	X	X	X	X	X	X	DE
0x5	Aux Status	read	SYS	NAC	0	0	0	KB NMI	0	1
0x7	Par. Poll Status	read	IE	IR	0	0	0	0	0	0
0x7	Par. Poll Control	write	IE	IR	X	X	X	X	X	X
0x9	Par. Poll Mask	read	d7	d6	d5	d4	d3	d2	d1	d0
0x9	Par. Poll Mask	write	d7	d6	d5	d4	d3	d2	d1	d0
0x11	TMS9914 Reg	0								
0x13	TMS9914 Reg	1								
0x15	TMS9914 Reg	2								
0x17	TMS9914 Reg	3								
0x19	TMS9914 Reg	4								
0x1B	TMS9914 Reg	5								
0x1D	TMS9914 Reg	6								
0x1F	TMS9914 Reg	7								

#### 3.1.3.1 LS-HPIB status/DMA control (3)

INT Interrupt from TMS9914. If set the TMS9914 is interrupting.  
 DE Writing a one to this bit will enable DMA on channel 0. Clearing it will disable DMA.

#### 3.1.3.2 LS-HPIB auxillary status (5)

SYS System controller line on HPiB. If set, the the TMS9914 is the system controller.  
 NAC Active controller status. If clear, the TMS9914 is currently the Controller in Charge.  
 KB NMI Keyboard non-maskable interrupt. If set, signal *NkbNmi* is asserted

#### 3.1.3.3 LS-HPiB Parallel Poll Status and Control (7)



- IE Interrupt enable. If set, Nikki will start performing a Parallel poll. If the poll response AND'd with the poll mask is not zero, the parallel poll will terminate and the IR bit will be set. This will cause *Nir[3]* to be asserted until IR is cleared.
- IR Interrupt request. If set, the parallel poll has terminated with a match, and *Nir[3]* will be asserted. Writing a zero to this bit will clear the interrupt request.
- † Both of these bits will be cleared by a hard/soft reset.

### 3.1.3.4 LS-HPIB Parallel Poll Mask (9)

- d7-d0 These bits form a mask for enabling parallel poll responses. If a bit is clear, then a parallel poll response which has the corresponding bit set (DIO8-DIO1) will be ignored. Any time a response bit and its corresponding mask bit is set, the parallel poll will be terminated and the IR bit set.
- † All bits will be cleared during a hard/soft reset.

TABLE 19. LS-HPIB Hidden Register Set

Address (HEX)	Contents								
	7	6	5	4	3	2	1	0	
0x1 read	0	0	0	0	0	0	0	0	SYS
0x1 write	X	X	X	X	X	X	X	X	SYS

### 3.1.3.5 LS-HPIB Hidden Register 1

- SYS This bit replaces the System Controller switch. It is reflected in standard register 5.
- † The power-up value of this register is 0x1. This bit is set whenever *NPuReset* is asserted.

### 3.1.4 RS-232

TABLE 20. RS-232 Standard Register Set

Address (HEX)			Contents							
			7	6	5	4	3	2	1	0
0x1	read	R/L	1	0	0	0	0	1	0	
0x1	write	X	X	X	X	X	X	X	X	
0x3	read	IE	IR	IL1	IL0	0	0	HH	HS	
0x3	write	IE	X	X	X	X	X	X	X	
0x11	NS16550 Reg	0								
0x13	NS16550 Reg	1								
0x15	NS16550 Reg	2								
0x17	NS16550 Reg	3								
0x19	NS16550 Reg	4								
0x1B	NS16550 Reg	5								
0x1D	NS16550 Reg	6								
0x1F	NS16550 Reg	7								

#### 3.1.4.1 RS-232 ID/Reset Register (1)

R/L Remote/Local. If set, the RS-232 port should be used as the system console.

† Writing any value to this register will reset the interface and assert *NrsReset* for 26 × seconds.

#### 3.1.4.2 RS-232 Status/Control Register (3)

IE Interrupt Enable. If set, the interface is enabled to interrupt on the level specified by IL1,IL0.

IR Interrupt Request. If set, the NS16550 is currently interrupting. This bit does not depend on IE.

IL1,IL0 Indicate the interrupting level.

IL1	IL0	Level
0	0	3
0	1	4
1	0	5
1	1	6

HH Hardware Handshake. If set, hardware handshake circuitry is available on this interface.

- HS High Speed mode. If set, the higher baud rate generator clock is active. This frequency is 7.2727MHz. If clear the baud rate generator clock is 2.4614MHz
- † The IE bit is cleared during a soft/hard reset.

**TABLE 21. RS-232 Hidden Register Set**

Address (HEX)		Contents							
		7	6	5	4	3	2	1	0
0x1	read	R/L	0	IL1	IL1	0	HH	HS	REN
0x1	write	R/L	X	IL1	IL1	X	HH	HS	REN

- R/L This bit replaces the Remote/Local switch. It is reflected in standard register 1.
- IL1/ILO These bits replace the Interrupt level switches. They are reflected in standard register 3.
- HH This bit should be set if "hardware handshake" circuitry is implemented on this particular RS-232 interface. It is reflected in standard register 3.
- HS This bit when set will enable the 7.2727MHz baud rate generator clock, otherwise the standard 2.4576MHz frequency is used. It is reflected in standard register 3.
- REN This bit replaces the Remote Enable switch. When set, the modem handshake lines on the NS16550 are permanently driven true. When clear, the levels are those driven in from the RS-232 connector. This bit is reflected in standard register 3.
- † The default value for this register is 0x21. It is set to this value when *NPuReset* is asserted.

### 3.1.5 Centronics Software ERS

TABLE 22. Centronics Standard Register Set

Address (HEX)			Contents							
			7	6	5	4	3	2	1	0
0x1	ID	read	0	0	0	0	0	1	1	0
0x1	Reset	write	X	X	X	X	X	X	X	X
0x3	Status	read	IE	IR	IL1	IL0	IOM	I/O	DE1	DE0
0x3	Control	write	IE	X	X	X	IOM	I/O	DE1	DE0
0x5	Device Status	read	FULL	EMPTY	STROBE	BUSY	ACK	ERROR	SELECT	PE
0x5	write	X	X	X	X	X	X	X	X	
0x7	Device Control	read	0	0	0	0	0	INIT	SLCTIN	WRnRD
0x7	Device Control	write	X	X	X	X	X	INIT	SLCTIN	WRnRD
0x9	Intr Status	read	FULL	EMPTY	0	nBUSY	ACK	ERROR	SELECT	PE
0x9	Intr Control	write	FULL	EMPTY	0	nBUSY	ACK	ERROR	SELECT	PE
0xB	FIFO data	read	d7	d6	d5	d4	d3	d2	d1	d0
0xB	FIFO data	write	d7	d6	d5	d4	d3	d2	d1	d0

#### 3.1.5.1 Centronics ID/Reset Register (1)

0x6 This is the ID value of the Centronics interface.

† Writing any value to this register will reset the interface and clear the FIFO.

#### 3.1.5.2 Centronics Interface Control/Status Register (3)

IE	Interrupt Enable. Setting this bit enables interrupting conditions to assert the <i>Nir</i> signal indicated by IL1,IL0.
IR	Interrupt Request. If this bit is set, the interface is currently generating an interrupt, or would be if IE was set.
IL1,IL0	Interrupt Level. Specifies interrupting level 3,4,5, or 6 (see SCSI reg 3).
IOM	IO modifier. If cleared, outbound transfers handshake with both BUSY and NACK and inbound transfers will use the FIFO. If set, outbound transfers will handshake with BUSY only and inbound transfers will only use 1 location in the FIFO (FIFO disable).
I/O	I/O direction. If cleared, the interface will be performing outbound transfers. If set, the interface will be performing inbound transfers.
DE1,DE0	DMA enable channel 1 and 0. When one (and only one) of these bits is set, the DMA machines will be enabled to start transferring. Once inbound DMA has been enabled it (DMA) must be used for the entire data transfer (until NACK for a ScanJet).

† A hard/soft reset will clear all except the IL1,IL0 bits.

### 3.1.5.3 Centronics Device Status Register (5)

FULL	FIFO full. If set, the FIFO is currently full. This is valid for FIFO disable mode also.
EMPTY	FIFO empty. If set, the FIFO is currently empty. This is valid for FIFO disable mode also.
STROBE	Centronics STROBE. If set, the NSTROBE signal is asserted (low); otherwise NSTROBE is not asserted.
BUSY	Centronics BUSY. If set, the BUSY signal is asserted (high); otherwise BUSY is not asserted.
ACK	Centronics ACK. If set, the NACK signal is asserted (low); otherwise NACK is not asserted.
ERROR	Centronics NERROR. If set, the NERROR signal is asserted (low); otherwise NERROR is not asserted.
SELECT	Centronics SELECT. If set, the SELECT signal is asserted (high); otherwise SELECT is not asserted.
PE	Centronics PE (paper error). If set, the PE signal is asserted (high); otherwise PE is not asserted.
†	A hard/soft reset will clear FULL and STROBE, and set EMPTY.

### 3.1.5.4 Centronics Device Control Register (7)

INIT	Centronics NINIT. If set the NINIT signal will be asserted (low); otherwise NINIT will be deasserted.
SLCTIN	Centronics NSLCTIN (select in). If set the NSLCTIN will be asserted (low); otherwise NSLCTIN will be deasserted.
WRnRD	Centronics NAUTOFDXT, ScanJet WRnRD signal. If set, WRnRD will be asserted (high); otherwise WRnRD will be deasserted.
†	A hard/soft reset will affect this register but <i>will not</i> affect the actual lines. Therefore, to keep this register consistent with the signals, a write must be performed after a reset.

### 3.1.5.5 Centronics Interrupt Control/Status Register (9)

FULL IE/IR	FIFO full Interrupt Enable and Request. If a one is written to this bit then a FIFO full condition will be reflected in this register and IR of register 3.
EMPTY IE/IR	FIFO empty Interrupt Enable and Request. If a one is written to this bit then a FIFO empty condition will be reflected in this register and IR of register 3.

nBUSY IE/IR	NOT BUSY Interrupt Enable and Request. If a one is written to this bit then a NOT BUSY (BUSY LOW) condition will be reflected in this register and IR of register 3.
ACK IE/IR	ACK Interrupt Enable and Request. If a one is written to this bit then a NACK falling edge will set the read version of this bit and set IR of register 3. The interrupting condition is cleared by writing a zero to this bit.
ERROR IE/IR	ERROR Interrupt Enable and Request. If a one is written to this bit then a NERROR falling <i>or</i> rising edge will set the read version of this bit and set IR of register 3. The interrupting condition is cleared by writing a zero to this bit.
SELECT IE/IR	SELECT Interrupt Enable and Request. If a one is written to this bit then a NSELECT falling <i>or</i> rising edge will set the read version of this bit and set IR of register 3. The interrupting condition is cleared by writing a zero to this bit.
PE IE/IR	PE (paper error) Interrupt Enable and Request. If a one is written to this bit then a PE falling <i>or</i> rising edge will set the read version of this bit and set IR of register 3. The interrupting condition is cleared by writing a zero to this bit.
†	A soft/hard reset will clear all bits in this register.

### 3.1.5.6 Centronics FIFO Data Register (11)

- D7-0 If the I/O bit of register 3 is cleared, a write to this register will write data into the FIFO. Software must guarantee that FULL is not set. The back end machine upon seeing the FIFO is not empty will perform an output transaction (wait for NOT BUSY, assert data, then NSTROBE, etc).
- If the I/O bit of register 3 is set, a read from this register will return the next data in the FIFO. Software must guarantee that EMPTY is not set. The back end machine upon seeing the FIFO is not full will perform an input transaction (deassert BUSY, wait for NSTROBE, latch data into FIFO, etc).

TABLE 23. Centronics Hidden Register Set

Address (HEX)		Contents							
		7	6	5	4	3	2	1	0
0x1	read	0	0	IL1	IL0	0	0	0	0
0x1	write	X	X	IL1	IL0	X	X	X	X

IL1,IL0 Interrupting level. The values 0,1,2 and 3 correspond to interrupt levels 3,4,5 and 6. The value of these bits are reflected in standard register 3.

### *3.2 Software Performance Specifications and Considerations*

Read the Errata section in conjunction with the following sections.

#### *3.2.1 SCSI*

Software must be careful to set the direction and size bits in register 3 prior to setting either of the DMA enable bits.

During inbound DMA when the Fujitsu interrupts, Nikki immediately requests a DMA transfer, if there is data in the FIFO. If the DMA channel is in priority mode, all of the DATA should be transferred to main memory. But there is a chance given other DMA hardware designs that the FIFO may not be empty even if the 87033 interrupt indicating an end of transfer. One should make sure that the DMA chip reports all bytes transferred prior to disabling DMA.

#### *3.2.2 HS-HPIB*

To work around a Nikki bug, the Medusa interrupt signal should be delayed 30 microseconds (time span of the internal timer). If this is not done, it is possible for software to respond to the Medusa interrupt and disable the DMA controller before Nikki has transferred the rest of its data.

#### *3.2.3 LS-HPIB*

Do not enable DMA if parallel polling is enabled! (This is not a new requirement)

#### *3.2.4 RS-232*

Hardware Handshake is completely independent from the presence of this ASIC.

#### *3.2.5 Centronics*

There is a state machine bug that requires the direction bit in register 3 to be toggled just before enabling INBOUND DMA. Basically, the machine was hand kludged incorrectly in a previous mask turn and this was never corrected. Thus this is a strange bug. The I/O bit should be cleared, and then BOTH the I/O bit AND the proper DMA enable bit should be set at the SAME time. This is for INBOUND DMA ONLY! If this is not done, the transfer will lock up because Nikki will never request a DMA transfer.

There is a design bug which inhibits the use of mixed DMA and programmed transfers for INBOUND ONLY! Once an INBOUND transfer uses DMA, all future inbound transfers, UNTIL THE NACK INTERRUPT (end of transfer) or an output transfer occurs, should use DMA. The hardware unloads a byte from the FIFO into a holding register and waits for the next DMA transfer to take that byte. If the DMA



enable bit is cleared, this byte will remain in the holding register. Thus if the FIFO register is read, the next byte will be read instead of the one in the holding register. Once DMA is enabled this byte (the one in the holding register) will be the next one out. There is no way to tell if there is a byte in the holding register. An acceptable sequence follows: outbound DMA/non-DMA, inbound NON-DMA, inbound NON-DMA, inbound DMA, inbound DMA, NACK interrupt indicating end of transfer, outbound DMA/non-DMA, outbound DMA/non-DMA, inbound non-DMA, etc.



#### 4. Errata

Below is a list of things I would change/fix given a chance to reroute/mask-turn Nikki.

- The Centronics FIFO counters do not count in a Grey code manner. There is currently a screen at NID for chips that might be flakey due to this. My test vectors do not pass HILO MIN timing with NO capacitance simulations, but even +3sigma parts at -20 C aren't that fast.
- The Centronics DMA state machine (cnDMA<sub>sm</sub>) was kludged incorrectly at one time and this bad fix has propagated through the last turn. The PD source should generate proper logic to fix this problem.
- The Centronics DMA state machine should be modified to not unload the FIFO into the holding register until *Ndmack* is asserted during inbound DMA. This would enable mixed DMA and programmed inbound transfers.
- The HS-HPIB DMA machine should generate a FIFO timeout when a Medusa EOT occurs during inbound DMA. This would eliminate the need for the 30microsecond delay of the Medusa interrupt signal.
- It would be nice for software to have a Centronics "IDLE" interrupting condition. Centronics would be IDLE during outbound transfers when the FIFO is empty and the last transfer has completed (including the detection of the NACK pulse). This would be used by software to guarantee that the last byte was accepted by the peripheral so that it could turn the bus around.

## 5. State Machine Algorithms

The algorithms for most of the state machines in the chip are included in this section. The equations are expressed in a Pascal-like format which is required for the PAL equation generation aid: PD.

### 5.1 SCSI/HS-HPIB DIO-FIFO DMA Machine

```

1:pal DIOdmaSM;
2:
3:{
4:    PAL Designer DIO DMA State Machine program.
5:
6:    This file generates the Next State Equations and Output
7:    equations for the DIO-DMA half of the SCSI portion of the interface.
8:
9:    Don Soltis      v0.1   9/27/88
10:   Initial equations
11:   Don Soltis      v0.2   1/8/89
12:   modified for active low outputs
13:
14:}
15:
16:input .          DMAen,I0active,FifoFull,FifoEmpty,inbound,
17:                rst,word,almostFull,
18:                dmack,dmrq,dtacki,
19:                timeout,
20:                EOT,                {End of transfer}
21:                SCSI,                {SCSI/~HPIB select}
22:                state[2..0];
23:
24:output          state[2..0],
25:                SldFifo,                { set ldFifo }
26:                RldFifo,                { reset ldFifo }
27:                SunldFifo,            { set unldFifo }
28:                RunldFifo,            { reset unldFifo }
29:                Sdmrq,                { set DMA request }
30:                Rdmrq,                { reset DMA request }
31:                oDataEn,            { Output data Enable }
32:                intr,                { End of xfer interrupt }
33:                dmrq;
34:
35:pal_type        'ASIC';
36:pal_pin_order   ~SldFifo,~SunldFifo,~Sdmrq;
37:
38:state_vars      state[2..0];
39:
40:macro idle;     begin 0 end;
41:
42:macro inb0;     begin 1 end; {Inbound state 0}
43:macro inb1;     begin 3 end; {Inbound state 1}
44:macro inb2;     begin 2 end; {Inbound state 2}
45:
46:macro outb0;    begin 4 end; {outbound state 0}
47:macro outb1;    begin 5 end; {outbound state 1}
48:macro outb2;    begin 6 end; {outbound state 2}
49:
50:macro next_state; begin state[2..0] end;
51:
52:begin
53:
54:if rst then begin { Reset }
55:
56:  state[2..0] := idle;
57:  Rdmrq      := 1;
58:  RldFifo    := 1;
59:  RunldFifo  := 1;
60:
61:  end;
62:

```



```
63:case state[2..0] of
64:
65:   idle:      {idle states}
66:     begin
67:
68:       if DMAen then begin { Make sure DMA is enabled }
69:
70:         {*****}
71:         { start inbound if Fifofull or timeout }
72:         {*****}
73:         if( inbound AND (FifoFull or (timeout~FifoEmpty)) ) then begin
74:           SunldFifo := 1;
75:           Sdmrq     := 1;
76:           next_state := inb0;
77:         end
78:
79:         {*****}
80:         { start outbound if FifoEmpty }
81:         {*****}
82:         else if( ~inbound AND FifoEmpty ) then begin
83:           Sdmrq := 1;
84:           next_state := outb0;
85:         end
86:         else
87:           next_state := idle;
88:
89:         end {if DMAen}
90:       else
91:         next_state := idle;
92:
93:     end; {in idle0}
94:
95:   inb0: begin
96:     RunldFifo := 1;
97:     if dmack then
98:       next_state := inb1
99:     else
100:      next_state := inb0; { wait for dmack }
101:   end; {case inb0}
102:
103:   inb1: begin
104:     if FifoEmpty then
105:       Rdmrq := 1; {no more DMA requests}
106:
107:     oDataEn := 1;
108:     RunldFifo := 1;
109:     next_state := inb2;
110:   end; {case inb1}
111:
112:   inb2: begin
113:     dmardy := 1;
114:     oDataEn := 1;
115:     if dmack then begin
116:       if ( ~SCSI and EOT ) then begin
117:         intr = 1; { interrupt if we are done and in HS-HPIB mode}
118:         Rdmrq := 1; {no more DMA requests}
119:       end;
120:       next_state := inb2;
121:     end
122:     else begin
123:       if ~darq then begin
124:         next_state := idle;
125:       end
126:       else begin { Another transfer }
127:         SunldFifo := 1;
128:         next_state := inb0;
129:       end;
130:     end;
131:   end; {case inb2}
132:
133:   outb0: begin
134:
135:     if DMAen then begin
136:       if dmack then begin
137:         if almostFull then {Only have one slot left}
138:           Rdmrq := 1 {no more requests}
139:         else
```



```
140:          Sdarq := 1;          {continue Request}
141:          if dtacki then begin
142:              next_state := outb1
143:          end
144:          else
145:              next_state := outb0;
146:          end
147:          else
148:              next_state := outb0; { wait for dmack }
149:          end
150:
151:      else begin
152:          next_state := idle;
153:          Rdarq := 1;          {don't darq}
154:          end;
155:
156:      end; {case outb0}
157:
158:  outb1: begin
159:      dardy := 1;          {data hold 85 ns after this}
160:      SldFifo := 1;      {Actually occurs in next state}
161:      next_state := outb2;
162:      end; {case outb1}
163:
164:  outb2: begin
165:      dardy := 1;
166:      RldFifo := 1;
167:      if dmack then
168:          next_state := outb2
169:      else begin
170:          if ~darq then
171:              next_state := idle
172:          else begin
173:              next_state := outb0;
174:          end;
175:      end;
176:
177:      end; {case outb2}
178:
179:  end; { case }
180: end. {END}
```



```
1:<no-name>
2:PAL Designer 2.2 88/05/05 14:27
3:Mon Sep 18 21:16:52 1989
4:PALASH Description for DIODMASH
5:/SLDFIFO
6:/SUNLDFIFO /SDMRQ
7:
8;;; PAL Attributes File is
9;;; Using structure info for "ASIC"
10:
11:DMRDY := STATE[1] * /STATE[0]
12: + STATE[2] * /STATE[1] * STATE[0]
13:
14:INTR = DMACK * EOT * /SCSI * /STATE[2] * STATE[1] * /STATE[0]
15:
16:ODATAEN := /STATE[2] * STATE[1]
17:
18:RDMRQ := DMACK * STATE[2] * /STATE[1] * /STATE[0] * ALMOSTFULL
19: + /DMAEN * STATE[2] * /STATE[1] * /STATE[0]
20: + DMACK * EOT * /SCSI * /STATE[2] * STATE[1] * /STATE[0]
21: + /STATE[2] * STATE[1] * STATE[0] * FIFOEMPTY
22: + RST
23:
24:RLDFIFO := STATE[2] * STATE[1] * /STATE[0]
25: + RST
26:
27:RUNLDFIFO := /STATE[2] * STATE[0]
28: + RST
29:
30:/SDMRQ := /DMAEN
31: + /DMACK * STATE[2]
32: + STATE[2] * ALMOSTFULL
33: + STATE[1]
34: + STATE[0]
35: + /STATE[2] * /FIFOFULL * FIFOEMPTY * INBOUND
36: + /STATE[2] * /FIFOEMPTY * /INBOUND
37: + /STATE[2] * /FIFOFULL * INBOUND * /TIMEOUT
38:
39:/SLDFIFO := /STATE[2]
40: + STATE[1]
41: + /STATE[0]
42:
43:STATE[0] := DMAEN * DMACK * STATE[2] * /STATE[1] * /STATE[0] * DTACKI
44: + /DMACK * DMRQ * /STATE[2] * STATE[1] * /STATE[0]
45: + DMAEN * /STATE[2] * /STATE[1] * FIFOFULL * INBOUND
46: + DMAEN * /STATE[2] * /STATE[1] * /FIFOEMPTY * INBOUND * TIMEOUT
47: + /STATE[2] * /STATE[1] * STATE[0]
48:
49:STATE[1] := DMACK * STATE[1] * /STATE[0]
50: + STATE[2] * /STATE[1] * STATE[0]
51: + DMACK * /STATE[2] * STATE[0]
52: + /STATE[2] * STATE[1] * STATE[0]
53:
54:STATE[2] := DMRQ * STATE[2] * STATE[1] * /STATE[0]
55: + DMACK * STATE[2] * STATE[1] * /STATE[0]
56: + DMAEN * STATE[2] * /STATE[1]
57: + STATE[2] * /STATE[1] * STATE[0]
58: + DMAEN * /STATE[1] * /STATE[0] * FIFOEMPTY * /INBOUND
59:
60:/SUNLDFIFO := /DMAEN * /STATE[1]
61: + STATE[2]
62: + DMACK * STATE[1]
63: + /DMRQ * STATE[1]
64: + STATE[0]
65: + /STATE[1] * /FIFOFULL * FIFOEMPTY
66: + /STATE[1] * /INBOUND
67: + /STATE[1] * /FIFOFULL * /TIMEOUT
68:
69:
```

## 5.2 SCSI Register Access Machine

```
1:pal SCSIIdioSM;
2:
```



```

3:{
4:    PAL Designer SCSI DIO State Machine program.
5:
6:    This file generates the Next State Equations and Output
7:    equations for DIO register accesses of the SCSI portion
8:    of Nikki.
9:
10:   Don Soltis      v0.1    1/11/89
11:   Initial equations
12:   Don Soltis      v0.2    1/14/89
13:   Stretched out SPC (87033) signals
14:
15:}
16:
17:input      CS, write, DMAactive, rst, a[5..0],
18:           hiddenEn, state[2..0];
19:
20:output     state[2..0],
21:           IOactive,
22:           writeReg1, writeReg3, wrapLe, writeReg7, { DIO reg writes }
23:           writeReg9,
24:           readReg1, readReg3, loop0e, readReg7, { DIO reg reads }
25:           readReg9, readRegB, readRegD, readRegF,
26:           schin, dmrsp, { byte DMA reg }
27:           dtack, OdioEn1,
28:           OscsiEn,
29:           swrite, sread; { SPC control sigs }
30:                       { spcCs, Nswrite and Nsread }
31:                       { are derived from these }
32:
33:pal_type   'ASIC';
34:
35:{ ONLY the active low outputs need to be specified here for logic reduction }
36:{ use the psn version of pd2ddl for proper generation }
37:
38:state_vars      state[2..0];
39:
40:macro idle;      begin 0 end;
41:macro s0;        begin 1 end;
42:macro s1;        begin 3 end;
43:macro s2;        begin 2 end;
44:macro s3;        begin 6 end;
45:macro s4;        begin 7 end;
46:macro s5;        begin 5 end;
47:macro s6;        begin 4 end;
48:
49:macro address;   begin a[5..0] end;
50:
51:macro dio_reg;   begin
52:    ( a[5] == 0 )
53:    end;
54:
55:macro spc_reg;   begin
56:    ( a[5] == 1 )
57:    end;
58:
59:macro next_state; begin state[2..0] end;
60:
61:Procedure enable_dio_reg;
62:    begin
63:        case a[4..1] of
64:            0: {reg 1}
65:                readReg1 = 1;
66:            1: {reg 3}
67:                readReg3 = 1;
68:            2: {reg 5}
69:                loop0e = 1;
70:            3: {reg 7}
71:                readReg7 = 1;
72:            4: {reg 9}
73:                readReg9 = 1;
74:            5: {reg B}
75:                readRegB = 1;
76:            6: {reg D}
77:                readRegD = 1;
78:            7: {reg F}
79:                readRegF = 1;

```



```
80:         end;
81:     end;
82:
83: Procedure write_dio_reg;
84:     begin
85:         case a[4..1] of
86:             0: {reg 1}
87:                 writeReg1 = 1;
88:             1: {reg 3}
89:                 writeReg3 = 1;
90:             2: {reg 5}
91:                 begin
92:                     wrapLe = 1;
93:                     OscsiEn = 1;
94:                 end;
95:             3: {reg 7}
96:                 writeReg7 = 1;
97:             4: {reg 9}
98:                 writeReg9 = 1;
99:         end;
100:    end;
101:
102: Procedure SCSI_state_hold;
103: { This procedure is used in states s2,s3,s4,s5 }
104:     begin
105:         if CS then begin
106:             IOactive = 1; {continue to assert IOactive the entire time}
107:             if ~write then begin
108:                 OdioEn1 = 1;
109:                 if a[5..1] = 8 then begin
110:                     schin = 1;          {schin for read DMA reg}
111:                     dmarsp = 1;
112:                 end;
113:             end;
114:             if spc_reg then begin {spc reg access}
115:                 if write then begin
116:                     OscsiEn = 1;
117:                     swrite = 1;
118:                 end
119:                 else sread = 1
120:                 end
121:             else {dio_reg} begin
122:                 if ~write then begin
123:                     enable_dio_reg;
124:                 end
125:                 else begin
126:                     if a[5..1] = 0 then writeReg1 = 1;          {Soft reset signal}
127:                     if a[5..1] = 2 then OscsiEn = 1;          {drive dio onto scsi}
128:                     if a[5..1] = 3 then writeReg7 = 1;          {Fifo reset signal}
129:                 end;
130:                 end; {else dio_reg}
131:             case state[2..0] of
132:                 s2: next_state := s3;
133:                 s3: next_state := s4;
134:                 s4: next_state := s5;
135:                 s5: begin
136:                     if write~"spc_reg then write_dio_reg;
137:                     next_state := s6;
138:                 end;
139:             end;
140:         end {if CS}
141:     else {~CS}
142:         next_state := idle;
143:     end; {procedure SCSI_state_hold}
144:
145: {*****
146: * Main
147: *****}
148: begin
149:
150: if rst then begin      { Reset }
151:
152:     state[2..0]      := idle;
153:
154:     end;
155:
156: case state[2..0] of
```



```
157:
158:   idle:          {idle states}
159:   begin
160:     if CS then begin
161:       IOactive = 1;
162:       next_state := s0;
163:     end
164:   else
165:     next_state := idle;
166:   end; {in idle}
167:
168:   s0:            { continue to assert IOactive, DMA has priority }
169:   begin
170:     if CS then begin
171:       IOactive = 1;
172:       next_state := s1
173:     end
174:   else
175:     next_state := idle;
176:   end; {case s0}
177:
178:   s1:
179:   begin
180:     if CS then begin
181:       IOactive = 1;
182:       if DMAactive then
183:         next_state := s1
184:       else begin {"DMAactive"}
185:
186:         if ~write then begin
187:           OdioEn1 = 1;
188:           if a[5..1] = 8 then schin = 1; {schin for read DMA reg}
189:         end;
190:
191:         if spc_reg then begin {spc reg access}
192:           if write then OacsiEn = 1;
193:         end
194:       else begin {dio_reg}
195:         if ~write then begin { read }
196:           enable_dio_reg;
197:         end;
198:       end; {else dio_reg}
199:       next_state := s2;
200:     end; {else ~DMAactive}
201:   end {if CS}
202:   else {"CS"}
203:     next_state := idle;
204:
205:   end; {case s1}
206:
207:   s2:
208:   begin
209:     SCSI_state_hold;
210:   end; {case s2}
211:
212:   s3:
213:   begin
214:     SCSI_state_hold;
215:   end; {case s3}
216:
217:   s4:
218:   begin
219:     SCSI_state_hold;
220:   end; {case s4}
221:
222:   s5:
223:   begin
224:     SCSI_state_hold;
225:   end; {case s5}
226:
227:   s6:
228:   begin
229:     if CS then begin
230:       IOactive = 1; {continue to assert IOactive the entire time}
231:       dtack = 1;
232:       if ~write then begin
233:         OdioEn1 = 1;
```





```
234:         if a[S..1] = 8 then schin = 1;    {schin for read DMA reg}
235:         end;
236:     if spc_reg then begin {spc reg access}
237:         if write then begin
238:             OscsiEn = 1;
239:             swrite = 0; {end Transaction}
240:         end;
241:     end
242:     else {dio_reg} begin
243:         if ~write then begin
244:             enable_dio_reg;
245:         end;
246:         {else {dio reg write} write_dio_reg;}
247:     end; {else dio_reg}
248:     next_state := s6;
249: end {if CS}
250: else {~CS}
251:     next_state := idle;
252: end; {case s3}
253:
254: end;
255:end.
```



```
1:<no-name>
2:PAL Designer 2.2 88/05/05 14:27
3:Tue Oct 17 18:52:13 1989
4:PALASM Description for SCSIDIOSH
5:
6:
7:;; PAL Attributes File is
8:;; Using structure info for "ASIC"
9:
10:DMRSP = CS * /WRITE * /A[5] * A[4] * /A[3] * /A[2] * /A[1] * STATE[2] * STATE[0]
11: + CS * /WRITE * /A[5] * A[4] * /A[3] * /A[2] * /A[1] * STATE[1] * /STATE[0]
12:
13:DTACK = CS * STATE[2] * /STATE[1] * /STATE[0]
14:
15:IOACTIVE = CS
16:
17:LOOPOE = CS * /WRITE * /A[5] * /A[4] * /A[3] * A[2] * /A[1] * STATE[2]
18: + CS * /WRITE * /A[5] * /A[4] * /A[3] * A[2] * /A[1] * STATE[1] * /DMAACTIVE
19: + CS * /WRITE * /A[5] * /A[4] * /A[3] * A[2] * /A[1] * STATE[1] * /STATE[0]
20:
21:ODIOEN1 = CS * /WRITE * STATE[2]
22: + CS * /WRITE * STATE[1] * /DMAACTIVE
23: + CS * /WRITE * STATE[1] * /STATE[0]
24:
25:OSCSIEN = CS * WRITE * A[5] * STATE[2]
26: + CS * WRITE * /A[4] * /A[3] * A[2] * /A[1] * STATE[2] * STATE[0]
27: + CS * WRITE * A[5] * STATE[1] * /DMAACTIVE
28: + CS * WRITE * /A[4] * /A[3] * A[2] * /A[1] * STATE[1] * /STATE[0]
29: + CS * WRITE * A[5] * STATE[1] * /STATE[0]
30:
31:READREG1 = CS * /WRITE * /A[5] * /A[4] * /A[3] * /A[2] * /A[1] * STATE[2]
32: + CS * /WRITE * /A[5] * /A[4] * /A[3] * /A[2] * /A[1] * STATE[1] * /DMAACTIVE
33: + CS * /WRITE * /A[5] * /A[4] * /A[3] * /A[2] * /A[1] * STATE[1] * /STATE[0]
34:
35:READREG3 = CS * /WRITE * /A[5] * /A[4] * /A[3] * /A[2] * A[1] * STATE[2]
36: + CS * /WRITE * /A[5] * /A[4] * /A[3] * /A[2] * A[1] * STATE[1] * /DMAACTIVE
37: + CS * /WRITE * /A[5] * /A[4] * /A[3] * /A[2] * A[1] * STATE[1] * /STATE[0]
38:
39:READREG7 = CS * /WRITE * /A[5] * /A[4] * /A[3] * A[2] * A[1] * STATE[2]
40: + CS * /WRITE * /A[5] * /A[4] * /A[3] * A[2] * A[1] * STATE[1] * /DMAACTIVE
41: + CS * /WRITE * /A[5] * /A[4] * /A[3] * A[2] * A[1] * STATE[1] * /STATE[0]
42:
43:READREG9 = CS * /WRITE * /A[5] * /A[4] * A[3] * /A[2] * /A[1] * STATE[2]
44: + CS * /WRITE * /A[5] * /A[4] * A[3] * /A[2] * /A[1] * STATE[1] * /DMAACTIVE
45: + CS * /WRITE * /A[5] * /A[4] * A[3] * /A[2] * /A[1] * STATE[1] * /STATE[0]
46:
47:READREGB = CS * /WRITE * /A[5] * /A[4] * A[3] * /A[2] * A[1] * STATE[2]
48: + CS * /WRITE * /A[5] * /A[4] * A[3] * /A[2] * A[1] * STATE[1] * /DMAACTIVE
49: + CS * /WRITE * /A[5] * /A[4] * A[3] * /A[2] * A[1] * STATE[1] * /STATE[0]
50:
51:READREGD = CS * /WRITE * /A[5] * /A[4] * A[3] * A[2] * /A[1] * STATE[2]
52: + CS * /WRITE * /A[5] * /A[4] * A[3] * A[2] * /A[1] * STATE[1] * /DMAACTIVE
53: + CS * /WRITE * /A[5] * /A[4] * A[3] * A[2] * /A[1] * STATE[1] * /STATE[0]
54:
55:READREGF = CS * /WRITE * /A[5] * /A[4] * A[3] * A[2] * A[1] * STATE[2]
56: + CS * /WRITE * /A[5] * /A[4] * A[3] * A[2] * A[1] * STATE[1] * /DMAACTIVE
57: + CS * /WRITE * /A[5] * /A[4] * A[3] * A[2] * A[1] * STATE[1] * /STATE[0]
58:
59:SCHIN = CS * /WRITE * /A[5] * A[4] * /A[3] * /A[2] * /A[1] * STATE[2]
60: + CS * /WRITE * /A[5] * A[4] * /A[3] * /A[2] * /A[1] * STATE[1] * /DMAACTIVE
61: + CS * /WRITE * /A[5] * A[4] * /A[3] * /A[2] * /A[1] * STATE[1] * /STATE[0]
62:
63:SREAD = CS * /WRITE * A[5] * STATE[2] * STATE[0]
64: + CS * /WRITE * A[5] * STATE[1] * /STATE[0]
65:
66:STATE[0] := CS * STATE[2] * STATE[1]
67: + CS * /STATE[2] * STATE[0] * DMAACTIVE
68: + CS * /STATE[2] * /STATE[1]
69:
70:STATE[1] := CS * STATE[1] * /STATE[0]
71: + CS * /STATE[2] * STATE[0]
72:
73:STATE[2] := CS * STATE[2]
74: + CS * STATE[1] * /STATE[0]
75:
76:SWRITE = CS * WRITE * A[5] * STATE[2] * STATE[0]
77: + CS * WRITE * A[5] * STATE[1] * /STATE[0]
```

```

78:
79:WRAPLE = CS * WRITE * /A[5] * /A[4] * /A[3] * A[2] * /A[1] * STATE[2] * /STATE[1] * STATE[0]
80:
81:WRITEREG1 = CS * WRITE * /A[5] * /A[4] * /A[3] * /A[2] * /A[1] * STATE[2] * STATE[0]
82: + CS * WRITE * /A[5] * /A[4] * /A[3] * /A[2] * /A[1] * STATE[1] * /STATE[0]
83:
84:WRITEREG3 = CS * WRITE * /A[5] * /A[4] * /A[3] * /A[2] * A[1] * STATE[2] * /STATE[1] * STATE[0]
85:
86:WRITEREG7 = CS * WRITE * /A[5] * /A[4] * /A[3] * A[2] * A[1] * STATE[2] * STATE[0]
87: + CS * WRITE * /A[5] * /A[4] * /A[3] * A[2] * A[1] * STATE[1] * /STATE[0]
88:
89:WRITEREG9 = CS * WRITE * /A[5] * /A[4] * A[3] * /A[2] * /A[1] * STATE[2] * /STATE[1] * STATE[0]

```

### 5.3 SCSI FIFO-87033 DMA Machine

```

1:pal SCsIdmaSM;
2:
3:{
4:    PAL Designer SCSI DMA State Machine program.
5:
6:    This file generates the Next State Equations and Output
7:    equations for the DMA half of the SCSI portion of the interface.
8:
9:    Don Soltis    v0.1    9/8/88
10:        Initial equations
11:    Don Soltis    v0.2    9/12/88
12:        Fixed Reseting of unld- and ld-Packer[1] for
13:        ldFifo
14:    Don Soltis    v0.3    9/13/88
15:        Moved reset of unld- and ld-Packer[1] to during
16:        clk8 low period (removed spikes)
17:    Don Soltis    v0.4    9/15/88
18:        changed from 8MHz to 16MHz clock to get rid of
19:        spikes on rld- and runld-Packer.
20:    Don Soltis    v0.5    9/19/88
21:        modified state transitions to increase performance
22:        and fixed outbound equations
23:    Don Soltis    v0.6    10/6/88
24:        Modified ldFifo and unldFifo as RS flip flops
25:        to fix race condition
26:    Don Soltis    v0.7    12/19/88
27:        Added pal_pin_order to specify active low outputs and
28:        used pd2ddl with '-p' switch.
29:    Don Soltis    v0.8    3/1/89
30:        Added assertion of DMAactive during idle1
31:    Don Soltis    v1.1    9/18/89
32:        Removed usage of DONE, and fixed DMA enable signal
33:        so that no data is lost.
34:
35:}
36:
37:input          DMAen,IOactive,FifoFull,FifoEmpty,inbound,
38:               rst,
39:               darq,           {This should be delayed one}
40:                               {extra clock cycle}
41:
42:               word,
43:               ldPacker[3..0],unldPacker[3..0],
44:               state[1..0],clk16,
45:               delayedDmrsp;
46:output         state[1..0],
47:               SldFifo,           { set ldFifo }
48:               RldFifo,           { reset ldFifo }
49:               SunldFifo,         { set unldFifo }
50:               RunldFifo,         { reset unldFifo }
51:               SldPacker[3..0],   { set ldPacker }
52:               RldPacker[3..0],   { reset ldPacker }
53:               SunldPacker[3..0], { set unldPacker }
54:               RunldPacker[3..0], { reset unldPacker }
55:               DMAactive,
56:               OdataEn,           { SCSI data output enable }
57:               dmrsp;
58:
59:pal_type      'ASIC';

```



```
60:
61: { ONLY the active low outputs need to be specified here for logic reduction }
62: { use the psd version of pd2ddl for proper generation }
63:
64: pal_pin_order      ~SldFifo, ~SunldFifo,
65:                   ~SldPacker[3..0],
66:                   ~SunldPacker[3..0],
67:                   ~rst;
68:
69: state_vars          state[1..0];
70:
71: macro idle0;        begin 0 end;
72: macro idle1;        begin 1 end;
73: macro inb0a;        begin 3 end; {Inbound/outbound, byte 0}
74: macro inb0b;        begin 2 end; {Inbound/outbound, byte 0}
75:
76: macro next_state;   begin state[1..0] end;
77:
78: macro iBYTE0;       begin 0 end;
79: macro iBYTE1;       begin 1 end;
80: macro iBYTE2;       begin 2 end;
81: macro iBYTE3;       begin 4 end;
82: macro iBYTE4;       begin 8 end;
83:
84: macro packer_full; begin
85:     (ldPacker[3]*~word or ldPacker[1]*word)
86:     end;
87:
88:
89: macro oBYTE0;       begin 0 end;
90: macro oBYTE1;       begin 1 end;
91: macro oBYTE2;       begin 2 end;
92: macro oBYTE3;       begin 4 end;
93: macro oBYTE4;       begin 8 end;
94:
95: macro unpacker_empty; begin
96:     (~unldPacker[3]*~unldPacker[2]*~unldPacker[1]*~unldPacker[0])
97:     end;
98:
99: macro unpacker_emptied; begin
100:    (unldPacker[3]*~word or unldPacker[1]*word)
101:    end;
102:
103: begin
104:
105: if rst then begin    { Reset }
106:
107:     state[1..0]      := idle0;
108:     RldPacker[3..0]  := 15;
109:     RunldPacker[3..0] := 15;
110:     RldFifo          := 1;
111:     RunldFifo        := 1;
112:     DMAactive        = 0;
113:     dmrsp            := 0;
114:
115:     end;
116:
117: case state[1..0] of
118:
119:     idle0:          {idle states}
120:         begin
121:
122:             dmrsp := 0;    {dma respond is always deasserted }
123:             DMAactive =0;  {dma active is always deasserted }
124:
125:             {*****}
126:             { load fifo if the packer is full and the Fifo is not full }
127:             {*****}
128:             if inbound*packer_full*~FifoFull then begin
129:                 SldFifo := 1;
130:                 RldPacker[1] := 1;    {if word mode}
131:                 RldPacker[3] := 1;    {if longword mode}
132:             end;
133:
134:             {*****}
135:             { unload fifo if unpacker is empty and Fifo is not empty }
136:             {*****}
```



```
137:         if ~inbound*unpacker_emptied then begin
138:             RunldPacker[1] := 1;           {if word mode}
139:             RunldPacker[3] := 1;           {if longword mode}
140:         end;
141:
142:         if ~inbound*delayedDarsp then begin
143:             DMAactive = 1;
144:             OdataEn = 1;
145:         end;
146:
147:         next_state := idle1;
148:     end; {case idle0}
149:
150: idle1:
151:
152:     begin
153:
154:         RldFifo := 1;
155:
156:         if DMAen then begin
157:
158:             {*****}
159:             { inbound transfers }
160:             {*****}
161:
162:             { load bytes for inbound transfers }
163:             if ~IOactive*inbound*dmrq*~packer_full then begin
164:                 DMAactive = 1;
165:                 next_state := inb0a;
166:             . end
167:
168:         else begin
169:             {*****}
170:             { outbound transfers }
171:             {*****}
172:
173:             { unload bytes for outbound transfers }
174:             if ~IOactive*~inbound*dmrq then begin
175:                 if unpacker_empty*~FifoEmpty then begin
176:                     SunldFifo := 1;           {load packer}
177:                     SunldPacker[0] := 1;       {enable unpacker byte 0 onto data bus}
178:                     DMAactive = 1;
179:                     next_state := inb0a;
180:                 end;
181:                 if unpacker_empty*FifoEmpty then begin
182:                     next_state := idle0;
183:                 end;
184:                 if unldPacker[0] then begin
185:                     RunldPacker[0] := 1;       {disable unpacker byte 0 onto data bus}
186:                     SunldPacker[1] := 1;       {enable unpacker byte 1 onto data bus}
187:                     DMAactive = 1;
188:                     next_state := inb0a;
189:                 end;
190:                 if unldPacker[1] then begin
191:                     RunldPacker[1] := 1;       {disable unpacker byte 0 onto data bus}
192:                     SunldPacker[2] := 1;       {enable unpacker byte 1 onto data bus}
193:                     DMAactive = 1;
194:                     next_state := inb0a;
195:                 end;
196:                 if unldPacker[2] then begin
197:                     RunldPacker[2] := 1;       {disable unpacker byte 0 onto data bus}
198:                     SunldPacker[3] := 1;       {enable unpacker byte 1 onto data bus}
199:                     DMAactive = 1;
200:                     next_state := inb0a;
201:                 end;
202:             end {if ~IOactive, etc.}
203:
204:         else {if not above}
205:             next_state := idle0;
206:
207:         end; {else begin}
208:
209:     end {if DMAen}
210:
211:     else {DMAen is not true}
212:         next_state := idle0;
213:
```



```
214:         end; {case idle1}
215:
216:     inb0a:
217:
218:         begin
219:
220:         RunldFifo      := 1;
221:
222:         DMAactive = 1;
223:         darsp     := 1;
224:         if inbound then begin
225:             if ldPacker[3..0] == 0 then
226:                 SldPacker[0] := 1;           { load packer byte 0 }
227:             if ldPacker[0] then begin
228:                 RldPacker[0] := 1;         { complete load packer byte 0 clock }
229:                 SldPacker[1] := 1;         { load packer byte 1 }
230:             end;
231:             if ldPacker[1] then begin
232:                 RldPacker[1] := 1;         { complete load packer byte 0 clock }
233:                 SldPacker[2] := 1;         { load packer byte 1 }
234:             end;
235:             if ldPacker[2] then begin
236:                 RldPacker[2] := 1;         { complete load packer byte 0 clock }
237:                 SldPacker[3] := 1;         { load packer byte 1 }
238:             end;
239:             next_state := inb0b;
240:         end {if inbound}
241:         else begin {outbound}
242:             OdataEn = 1;
243:             next_state := inb0b;
244:         end;
245:     end; {case inb0a}
246:
247:     inb0b:
248:         begin
249:             DMAactive = 1;           { Finish transfer regardless of DMAen}
250:             if ~inbound then
251:                 OdataEn = 1;
252:             darsp     := 1;
253:             next_state := idle0;
254:         end; {case inb0b}
255:
256:     end; { case state }
257:
258: end.
```



# HEWLETT PACKARD

```
1:<no-name>
2:PAL Designer 2.2 88/05/05 14:27
3:Mon Sep 18 17:19:17 1989
4:PALASM Description for SCSIDMASH
5:/SLDFIFO /SUNLDFIFO /SLDPACKER[3] /SLDPACKER[2] /SLDPACKER[1]
6:/SLDPACKER[0] /SUNLDPACKER[3] /SUNLDPACKER[2] /SUNLDPACKER[1] /SUNLDPACKER[0] /RST
7:
8:;; PAL Attributes File is
9:;; Using structure info for "ASIC"
10:
11:DMAACTIVE = /STATE[0] * /INBOUND * DELAYEDDMRSP
12: + STATE[1]
13: + DMAEN * DMRQ * STATE[0] * /IOACTIVE * /FIFOEMPTY * /INBOUND * /UNLDPACKER[3]
14: + DMAEN * DMRQ * STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[2]
15: + DMAEN * DMRQ * STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[1]
16: + DMAEN * DMRQ * STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[0]
17: + DMAEN * DMRQ * /WORD * STATE[0] * /IOACTIVE * INBOUND * /LDPACKER[3]
18: + DMAEN * DMRQ * WORD * STATE[0] * /IOACTIVE * INBOUND * /LDPACKER[1]
19:
20:DMRSP := STATE[1]
21:
22:ODATAEN = /STATE[0] * /INBOUND * DELAYEDDMRSP
23: + STATE[1] * /INBOUND
24:
25:RLDFIFO := /STATE[1] * STATE[0]
26: + RST
27:
28:RLDPACKER[0] := STATE[1] * STATE[0] * INBOUND * LDPACKER[0]
29: + RST
30:
31:RLDPACKER[1] := STATE[1] * STATE[0] * INBOUND * LDPACKER[1]
32: + /WORD * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[3]
33: + WORD * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[1]
34: + RST
35:
36:RLDPACKER[2] := STATE[1] * STATE[0] * INBOUND * LDPACKER[2]
37: + RST
38:
39:RLDPACKER[3] := /WORD * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[3]
40: + WORD * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[1]
41: + RST
42:
43:RUNLDFIFO := STATE[1] * STATE[0]
44: + RST
45:
46:RUNLDPACKER[0] := DMAEN * DMRQ * /STATE[1] * STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[0]
47: + RST
48:
49:RUNLDPACKER[1] := DMAEN * DMRQ * /STATE[1] * STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[1]
50: + /WORD * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[3]
51: + WORD * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[1]
52: + RST
53:
54:RUNLDPACKER[2] := DMAEN * DMRQ * /STATE[1] * STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[2]
55: + RST
56:
57:RUNLDPACKER[3] := /WORD * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[3]
58: + WORD * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[1]
59: + RST
60:
61:/SLDFIFO := /INBOUND
62: + FIFOFULL
63: + STATE[0]
64: + STATE[1]
65: + /WORD * /LDPACKER[3]
66: + WORD * /LDPACKER[1]
67:
68:/SLDPACKER[0] := /STATE[1]
69: + /STATE[0]
70: + /INBOUND
71: + LDPACKER[3]
72: + LDPACKER[2]
73: + LDPACKER[1]
74: + LDPACKER[0]
75:
76:/SLDPACKER[1] := /STATE[1]
77: + /STATE[0]
```



```
78: + /INBOUND
79: + /LDPACKER[0]
80:
81:/SLDPACKER[2] := /STATE[1]
82: + /STATE[0]
83: + /INBOUND
84: + /LDPACKER[1]
85:
86:/SLDPACKER[3] := /STATE[1]
87: + /STATE[0]
88: + /INBOUND
89: + /LDPACKER[2]
90:
91:STATE[0] := DMAEN * DMRQ * /STATE[1] * /IOACTIVE * /FIFOEMPTY * /INBOUND * /UNLDPACKER[3]
92: + DMAEN * DMRQ * /STATE[1] * /IOACTIVE * /INBOUND * UNLDPACKER[2]
93: + DMAEN * DMRQ * /STATE[1] * /IOACTIVE * /INBOUND * UNLDPACKER[1]
94: + DMAEN * DMRQ * /STATE[1] * /IOACTIVE * /INBOUND * UNLDPACKER[0]
95: + DMAEN * DMRQ * /WORD * /STATE[1] * /IOACTIVE * INBOUND * /LDPACKER[3]
96: + DMAEN * DMRQ * WORD * /STATE[1] * /IOACTIVE * INBOUND * /LDPACKER[1]
97: + /STATE[1] * /STATE[0]
98:
99:STATE[1] := DMAEN * DMRQ * STATE[0] * /IOACTIVE * /FIFOEMPTY * /INBOUND * /UNLDPACKER[3]
100: + DMAEN * DMRQ * STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[2]
101: + DMAEN * DMRQ * STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[1]
102: + DMAEN * DMRQ * STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[0]
103: + STATE[1] * STATE[0]
104: + DMAEN * DMRQ * /WORD * STATE[0] * /IOACTIVE * INBOUND * /LDPACKER[3]
105: + DMAEN * DMRQ * WORD * STATE[0] * /IOACTIVE * INBOUND * /LDPACKER[1]
106:
107:/SUNLDFIFO := /DMAEN
108: + /DMRQ
109: + STATE[1]
110: + /STATE[0]
111: + IOACTIVE
112: + FIFOEMPTY
113: + INBOUND
114: + UNLDPACKER[3]
115: + UNLDPACKER[2]
116: + UNLDPACKER[1]
117: + UNLDPACKER[0]
118:
119:/SUNLDPACKER[0] := /DMAEN
120: + /DMRQ
121: + STATE[1]
122: + /STATE[0]
123: + IOACTIVE
124: + FIFOEMPTY
125: + INBOUND
126: + UNLDPACKER[3]
127: + UNLDPACKER[2]
128: + UNLDPACKER[1]
129: + UNLDPACKER[0]
130:
131:/SUNLDPACKER[1] := /DMAEN
132: + /DMRQ
133: + STATE[1]
134: + /STATE[0]
135: + IOACTIVE
136: + INBOUND
137: + /UNLDPACKER[0]
138:
139:/SUNLDPACKER[2] := /DMAEN
140: + /DMRQ
141: + STATE[1]
142: + /STATE[0]
143: + IOACTIVE
144: + INBOUND
145: + /UNLDPACKER[1]
146:
147:/SUNLDPACKER[3] := /DMAEN
148: + /DMRQ
149: + STATE[1]
150: + /STATE[0]
151: + IOACTIVE
152: + INBOUND
153: + /UNLDPACKER[2]
```



### 5.4 HS-HPIB Register Access Machine

```

1:pal HPIBdioSM;
2:
3:{
4:    PAL Designer HPIB DIO State Machine program.
5:
6:    This file generates the Next State Equations and Output
7:    equations for DIO register accesses of the HS-HPIB portion
8:    of Nikki.
9:
10:   Don Soltis      v0.1    2/6/89
11:   Initial equations
12:   Don Soltis      v0.2    2/18/89
13:   Added extra state between ioend and dtack
14:
15:}
16:
17:input      CS, write, DMAactive, rst, a[5..0],
18:           hiddenEn, state[2..0],
19:           rstWait,      {500ns POW delay}
20:           ioend;
21:
22:output     state[2..0],
23:           IOactive,
24:           writeReg1, writeReg3, writeReg5, writeReg7, { DIO reg writes }
25:           readReg1, readReg3, readReg5, readReg7, { DIO reg reads }
26:           dtack, OdioEn1,
27:           OhpibEn,
28:           MedusaCs,
29:           iogo;
30:
31:
32:pal_type      'ASIC';
33:
34:{ ONLY the active low outputs need to be specified here for logic reduction }
35:{ use the psn version of pd2ddl for proper generation }
36:
37:state_vars    state[2..0];
38:
39:macro idle;      begin 0 end;
40:macro s0;        begin 1 end;
41:macro s1;        begin 4 end;
42:macro s2;        begin 6 end;
43:macro s3;        begin 2 end;
44:
45:macro address;   begin a[5..0] end;
46:
47:macro dio_reg;   begin
48:    ( a[4] == 0 )
49:    end;
50:
51:macro MedusaReg; begin
52:    ( a[4] == 1 )
53:    end;
54:
55:macro next_state; begin state[2..0] end;
56:
57:Procedure enable_dio_reg;
58:    begin
59:        case a[3..1] of
60:            0: {reg 1}
61:                readReg1 = 1;
62:            1: {reg 3}
63:                readReg3 = 1;
64:            2: {reg 5}
65:                readReg5 = 1;
66:            3: {reg 7}
67:                readReg7 = 1;
68:        end;
69:    end;
70:
71:Procedure write_dio_reg;

```



```
72: begin
73:     case a[3..1] of
74:         0: {reg 1}
75:             writeReg1 = 1;
76:         1: {reg 3}
77:             writeReg3 = 1;
78:         2: {reg 5}
79:             writeReg5 = 1;
80:         3: {reg 7}
81:             writeReg7 = 1;
82:     end;
83: end;
84:
85: {*****}
86: * Main
87: {*****}
88: begin
89:
90: if rst then begin      { Reset }
91:
92:     state[2..0]      := idle;
93:
94:     end;
95:
96: case state[2..0] of
97:
98:     idle:           {idle states}
99:         begin
100:            if CS then begin
101:                IOactive = 1;
102:                next_state := s0;
103:            end
104:            else
105:                next_state := idle;
106:            end; {in idle}
107:
108:     s0:             { continue to assert IOactive, DMA has priority }
109:         begin
110:            if CS then begin
111:                IOactive = 1;
112:                next_state := s1
113:            end
114:            else
115:                next_state := idle;
116:            end; {case s0}
117:
118:     s1:
119:         begin
120:            if CS then begin
121:                IOactive = 1;
122:                if DMAactive then
123:                    next_state := s1
124:                else begin {DMA is not active}
125:
126:                    if ~write then OdioEn1 = 1;
127:
128:                    if MedusaReg then begin {Medusa reg access}
129:                        if write then OhpibEn = 1;
130:                        MedusaCs = 1;
131:                        iogo = 1;                {IOGO should be delayed 1 clock}
132:                    end
133:                    else begin {dio_reg}
134:                        if ~write then begin {read }
135:                            enable_dio_reg;
136:                        end
137:                        else {write}
138:                            if a[3..1] = 0 then writeReg1 = 1; {Reset}
139:                        end; {else dio_reg}
140:                        if MedusaReg and ~ioend then
141:                            next_state := s1      { wait for ioend }
142:                        else
143:                            next_state := s2;
144:                        end {else DMA is not active}
145:                    end {if CS}
146:                else {~CS}
147:                    next_state := idle;
148:
```



```
149:     end; {case s1}
150:
151:   s2:                                     { Give data time to settle }
152:   begin
153:     IOactive = 1;
154:     if dio_reg then
155:       if write then begin
156:         write_dio_reg;                    { latch data }
157:         if a[3..1] = 0 then              {reset, extend}
158:           if rstWait then
159:             next_state := s2            {wait for 500ns}
160:           else
161:             next_state := s3
162:           else
163:             next_state := s3;            {not reset register}
164:         end
165:       else begin
166:         OdioEn1 = 1;
167:         enable_dio_reg;
168:         next_state := s3;
169:       end
170:     else {MedusaReg} begin
171:       if ~write then OdioEn1 = 1;
172:       next_state := s3;
173:     end;
174:   end; {case s2}
175:
176:   s3:
177:   begin
178:     dtack = 1;
179:     IOactive = 1;
180:     if dio_reg then begin
181:       {if write then}
182:       {write_dio_reg}                    { latch data }
183:     {else begin}
184:     if ~write then begin
185:       OdioEn1 = 1;
186:       enable_dio_reg;
187:     end;
188:   else {MedusaReg}
189:   if ~write then OdioEn1 = 1;
190:
191:   if CS or ioend then
192:     next_state := s3
193:   else
194:     next_state := idle;
195:   end; {case s3}
196: end;
197:
198: end;
199: end.
```

```

1:<no-name>
2:PAL Designer 2.2 88/05/05 14:27
3:Fri Apr 28 13:24:33 1989
4:PALASM Description for HPIBDIOSM
5:
6:
7:;; PAL Attributes File is
8:;; Using structure info for "ASIC"
9:
10:DTACK = /STATE[2] * STATE[1] * /STATE[0]
11:
12:IOACTIVE = CS * /STATE[0]
13: + STATE[1] * /STATE[0]
14: + CS * /STATE[2] * /STATE[1]
15:
16:IOGO = CS * A[4] * STATE[2] * /STATE[1] * /STATE[0] * /DMAACTIVE
17:
18:MEDUSACS = CS * A[4] * STATE[2] * /STATE[1] * /STATE[0] * /DMAACTIVE
19:
20:ODIOEM1 = /WRITE * STATE[1] * /STATE[0]
21: + CS * /WRITE * STATE[2] * /STATE[0] * /DMAACTIVE
22:
23:OHPIBEM = CS * WRITE * A[4] * STATE[2] * /STATE[1] * /STATE[0] * /DMAACTIVE
24:
25:READREG1 = /WRITE * /A[4] * /A[3] * /A[2] * /A[1] * STATE[1] * /STATE[0]
26: + CS * /WRITE * /A[4] * /A[3] * /A[2] * /A[1] * STATE[2] * /STATE[0] * /DMAACTIVE
27:
28:READREG3 = /WRITE * /A[4] * /A[3] * /A[2] * A[1] * STATE[1] * /STATE[0]
29: + CS * /WRITE * /A[4] * /A[3] * /A[2] * A[1] * STATE[2] * /STATE[0] * /DMAACTIVE
30:
31:READREG5 = /WRITE * /A[4] * /A[3] * A[2] * /A[1] * STATE[1] * /STATE[0]
32: + CS * /WRITE * /A[4] * /A[3] * A[2] * /A[1] * STATE[2] * /STATE[0] * /DMAACTIVE
33:
34:READREG7 = /WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[1] * /STATE[0]
35: + CS * /WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[2] * /STATE[0] * /DMAACTIVE
36:
37:STATE[0] := CS * /STATE[2] * /STATE[1] * /STATE[0]
38:
39:STATE[1] := CS * STATE[1] * /STATE[0]
40: + STATE[1] * /STATE[0] * IOEND
41: + CS * STATE[2] * /STATE[0] * IOEND * /DMAACTIVE
42: + STATE[2] * STATE[1] * /STATE[0]
43: + CS * /A[4] * STATE[2] * /STATE[0] * /DMAACTIVE
44:
45:STATE[2] := WRITE * /A[4] * /A[3] * /A[2] * /A[1] * STATE[2] * STATE[1] * /STATE[0] * RSTWAIT
46: + CS * STATE[2] * /STATE[1] * /STATE[0]
47: + CS * /STATE[2] * /STATE[1] * STATE[0]
48:
49:WRITEREG1 = CS * WRITE * /A[4] * /A[3] * /A[2] * /A[1] * STATE[2] * /STATE[0] * /DMAACTIVE
50: + WRITE * /A[4] * /A[3] * /A[2] * /A[1] * STATE[2] * STATE[1] * /STATE[0]
51:
52:WRITEREG3 = WRITE * /A[4] * /A[3] * /A[2] * A[1] * STATE[2] * STATE[1] * /STATE[0]
53:
54:WRITEREG5 = WRITE * /A[4] * /A[3] * A[2] * /A[1] * STATE[2] * STATE[1] * /STATE[0]
55:
56:WRITEREG7 = WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[2] * STATE[1] * /STATE[0]

```

### 5.5 HS-HPIB FIFO-Medusa DMA Machine

```

1:pal HPIBdmaSM;
2:
3:{
4:    PAL Designer HPIB DMA State Machine program.
5:
6:    This file generates the Next State Equations and Output
7:    equations for the DMA half of the HPIB portion of the interface.
8:
9:    Don Soltis      v0.1    1/31/89
10:    Initial equations
11:
12:}
13:
14:input          DMAen,IOactive,FifoFull,FifoEmpty,inbound,
15:              done,rst,dmrq,word,byte,

```

```

16:          ldPacker[3..0],unldPacker[3..0],
17:          state[1..0],
18:          ioend,b[1..0],EOT;
19:
20:output      state[1..0],
21:          SldFifo,          { set ldFifo }
22:          RldFifo,          { reset ldFifo }
23:          SunldFifo,        { set unldFifo }
24:          RunldFifo,        { reset unldFifo }
25:          SldPacker[3..0],  { set ldPacker }
26:          RldPacker[3..0],  { reset ldPacker }
27:          SunldPacker[3..0], { set unldPacker }
28:          RunldPacker[3..0], { reset unldPacker }
29:          DMAactive,
30:          OdataEn,          { HPiB data output enable }
31:          hpibCs,iogo,
32:                                {I can only set the odd bits, }
33:                                {a reset or write to reg 7(?) }
34:                                {is the only way to clear }
35:          Sodd0,            { True if last inbound xfer }
36:          Sodd1,            { was not full width }
37:                                { odd1 odd0 residual }
38:                                { 0 0 none, full xfer }
39:                                { 0 1 one valid byte in Fifo}
40:                                { 1 0 two valid bytes in Fifo}
41:                                { 1 1 three valid bytes in Fifo}
42:                                {*****}
43:          intr;            { interrupt output }
44:                                { when asserted, beginning }
45:                                { of interrupt. }
46:
47:pal_type      'ASIC';
48:
49:{ ONLY the active low outputs need to be specified here for logic reduction }
50:{ use the psam version of pd2ddl for proper generation }
51:
52:pal_pin_order  ~SldFifo,~SunldFifo,
53:               ~SldPacker[3..0],
54:               ~SunldPacker[3..0],
55:               ~Sodd1,~Sodd0;
56:
57:state_vars     state[1..0];
58:
59:macro idle;    begin 0 end;
60:macro st0;     begin 1 end;
61:macro st1;     begin 3 end; {Inbound/outbound, byte 0}
62:macro st2;     begin 2 end; {Inbound/outbound, byte 0}
63:
64:macro next_state; begin state[1..0] end;
65:
66:macro packer_full;
67:  begin
68:    ( (ldPacker[3]*~word*~byte) {longword}
69:      or (ldPacker[1]*word*~byte) {word}
70:      or (ldPacker[1]*byte) {byte}
71:    )
72:  end;
73:
74:macro packer_empty;
75:  begin
76:    (
77:      ~ldPacker[3]*~ldPacker[2]*~ldPacker[1]*~ldPacker[0]
78:    )
79:  end;
80:
81:macro unpacker_empty; begin
82:  (
83:    ~unldPacker[3]*~unldPacker[2]*~unldPacker[1]*~unldPacker[0]
84:  )
85:  end;
86:
87:macro unpacker_emptied; begin
88:  (
89:    (unldPacker[3]*~word*~byte)
90:    or (unldPacker[1]*word*~byte)
91:    or (unldPacker[1]*byte)
92:  )

```



```
93:         end;
94:
95:begin
96:
97:if rst then begin      { Reset }
98:
99:  state[1..0]      := idle;
100:  RldPacker[3..0]  := 15;
101:  RunldPacker[3..0] := 15;
102:  RldFifo          := 1;
103:  RunldFifo       := 1;
104:  DMAActive        = 0;
105:  hpibCs          := 0;
106:  iogo            := 0;
107:
108:  end;
109:
110:case state[1..0] of
111:
112:  idle:          {idle states}
113:    begin
114:
115:      hpibCs := 0;
116:      iogo  := 0;
117:      DMAActive =0;  {dma active is always deasserted }
118:
119:      if DMAen then begin { Make sure DMA is enabled }
120:
121:
122:        {*****}
123:        { load fifo if the packer is full and the Fifo is not full }
124:        {*****}
125:        if inbound*packer_full*~FifoFull then begin
126:          SldFifo := 1;
127:          RldPacker[0] := 1;          {if byte mode}
128:          RldPacker[1] := 1;          {if word mode}
129:          RldPacker[3] := 1;          {if longword mode}
130:          next_state := idle;
131:        end;
132:
133:        {*****}
134:        { reset the ldFifo pulse when done }
135:        {*****}
136:        if inbound*packer_empty then
137:          RldFifo = 1;
138:
139:        {*****}
140:        { unload fifo if unpacker is empty and Fifo is not empty }
141:        {*****}
142:        if ~inbound*unpacker_emptied then begin
143:          RunldPacker[0] := 1; {if word mode}
144:          RunldPacker[1] := 1; {if byte mode}
145:          RunldPacker[3] := 1; {if longword mode}
146:          next_state := idle;
147:        end;
148:
149:        {*****}
150:        { inbound transfers }
151:        {*****}
152:
153:        { load bytes for inbound transfers }
154:        {***** don't need done in below *****}
155:        {if ~IOactive*inbound*dmrq*~packer_full*~done then begin}
156:        {*****}
157:        if ~IOactive*inbound*dmrq*~packer_full then begin
158:          next_state := st0;
159:        end
160:
161:        else begin
162:          {*****}
163:          { outbound transfers }
164:          {*****}
165:
166:          { unload bytes for outbound transfers }
167:          if ~IOactive*~inbound*dmrq then begin
168:            if unpacker_empty*~FifoEmpty then begin
169:              SunldFifo := 1;          {load packer}
```



```
170:         if ~byte then
171:             SunldPacker[0] := 1 {enable unpacker byte 0 onto data bus}
172:         else
173:             SunldPacker[1] := 1; {enable unpacker byte 1 onto data bus}
174:             next_state := st0;
175:         end;
176:     if unpacker_empty*FifoEmpty then begin
177:         next_state := idle;
178:     end;
179:     if unldPacker[0] then begin
180:         RunldPacker[0] := 1; {disable unpacker byte 0 onto data bus}
181:         SunldPacker[1] := 1; {enable unpacker byte 1 onto data bus}
182:         next_state := st0;
183:     end;
184:     if unldPacker[1]*~word*~byte then begin
185:         RunldPacker[1] := 1; {disable unpacker byte 0 onto data bus}
186:         SunldPacker[2] := 1; {enable unpacker byte 1 onto data bus}
187:         next_state := st0;
188:     end;
189:     if unldPacker[2] then begin
190:         RunldPacker[2] := 1; {disable unpacker byte 0 onto data bus}
191:         SunldPacker[3] := 1; {enable unpacker byte 1 onto data bus}
192:         next_state := st0;
193:     end;
194:     end {if ~IOactive, etc.}
195:
196:     else {if not above}
197:         next_state := idle;
198:
199:     end; {else begin}
200:
201:     end {if DMAen}
202:
203:     else {DMAen is not true}
204:         next_state := idle;
205:
206:     end; {case idle}
207:
208: st0:
209:     begin
210:         RunldFifo := 1; {finish FIFO pulse}
211:
212:         if DMAen then begin
213:
214:             DMAactive = 1;
215:             hpibCs = 1;
216:             if inbound then begin
217:                 iogo = 1;
218:                 if ioend then { wait for ioend }
219:                     next_state := st1
220:                 else
221:                     next_state := st0;
222:                 end
223:             else begin {outbound}
224:                 OdataEn = 1;
225:                 next_state := st1;
226:             end;
227:
228:         end {if DMAen}
229:     else begin
230:         next_state := idle;
231:     end;
232:
233:     end; {case inb0a}
234:
235: st1:
236:     begin { Finish transfer regardless of DMAen}
237:         DMAactive = 1;
238:         hpibCs = 1;
239:         iogo = 1;
240:         if inbound then begin
241:             if ldPacker[3..0] == 0 then
242:                 if ~byte then
243:                     SldPacker[0] := 1 { load packer byte 0 }
244:                 else
```



```
247:         SldPacker[1] := 1;           { load packer byte 0 }
248:     if ldPacker[0] then begin
249:         RldPacker[0] := 1;           { complete load packer byte 0 clock }
250:         SldPacker[1] := 1;           { load packer byte 1 }
251:     end;
252:     if ldPacker[1] then begin
253:         RldPacker[1] := 1;           { complete load packer byte 0 clock }
254:         SldPacker[2] := 1;           { load packer byte 1 }
255:     end;
256:     if ldPacker[2] then begin
257:         RldPacker[2] := 1;           { complete load packer byte 0 clock }
258:         SldPacker[3] := 1;           { load packer byte 1 }
259:     end;
260:     next_state := st2;
261: end {if inbound}
262: else begin{ outbound }
263:     iogo = 1;
264:     OdataEn = 1;
265:     if ioend then                     { wait for ioend }
266:         next_state := st2
267:     else
268:         next_state := st1;
269:     end
270: end; {case inb0b}
271:
272: st2:
273: begin                                 { Finish transfer regardless of DMAen}
274:     DMAactive = 1;
275:     if ~inbound then begin
276:         OdataEn = 1;
277:     end; {if outbound}
278:     if ioend then
279:         next_state := st2
280:     else
281:         if ~inbound*unpucker_emptied*done then
282:             intr = 1
283:         else
284:             {*****}
285:             { Medusa gave an EOT before filling packer, }
286:             { set odd0,1 depending on how many bytes are valid }
287:             { in last fifo entry }
288:             {*****}
289:             if inbound*~packer_full*EOT then begin         {odd transfer}
290:                 if ldPacker[0] = 1 then
291:                     Sodd0 = 1;
292:                 if ldPacker[1] = 1 then
293:                     Sodd1 = 1;
294:                 if ldPacker[2] = 1 then begin
295:                     Sodd0 = 1;
296:                     Sodd1 = 1;
297:                 end;
298:                 {odd = 1;}                               {Set Odd}
299:                 SldFifo := 1;                             {Load Fifo anyhow}
300:                 RldPacker[0] := 1;                       {Clear Packer}
301:                 RldPacker[1] := 1;
302:                 RldPacker[2] := 1;
303:                 RldPacker[3] := 1;
304:             end;
305:             next_state := idle;
306:         end;
307:
308:
309:     end; { case state }
310:
311: end.
```





```
1:<no-name>
2:PAL Designer 2.2 88/05/05 14:27
3:Thu Apr 27 20:58:56 1989
4:PALASH Description for HPIBDMASH
5:/SLDFIFO /SUNLDFIFO /SLDPACKER[3] /SLDPACKER[2] /SLDPACKER[1] /SLDPACKER[0]
6:/SUNLDPACKER[3] /SUNLDPACKER[2] /SUNLDPACKER[1] /SUNLDPACKER[0] /SODD1 /SODD0
7:
8;;; PAL Attributes File is
9;;; Using structure info for "ASIC"
10:
11:DMAACTIVE = STATE[1]
12: + DMAEN * STATE[0]
13:
14:HPIBCS = DMAEN * STATE[0]
15: + STATE[1] * STATE[0]
16:
17:INTR = DONE * WORD * STATE[1] * /STATE[0] * /IOEND * /INBOUND * UNLDPACKER[1]
18: + DONE * /WORD * /BYTE * STATE[1] * /STATE[0] * /IOEND * /INBOUND * UNLDPACKER[3]
19: + DONE * BYTE * STATE[1] * /STATE[0] * /IOEND * /INBOUND * UNLDPACKER[1]
20:
21:IOGO = DMAEN * STATE[0] * INBOUND
22: + STATE[1] * STATE[0]
23:
24:ODATAEN = STATE[1] * /INBOUND
25: + DMAEN * STATE[0] * /INBOUND
26:
27:RLDFIFO = DMAEN * /STATE[1] * /STATE[0] * INBOUND * /LDPACKER[3] * /LDPACKER[2] * /LDPACKER[1] * /LDPACKER[0]
28: + RST
29:
30:RLDPACKER[0] := BYTE * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[1]
31: + /WORD * /BYTE * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[3]
32: + WORD * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[1]
33: + STATE[1] * STATE[0] * INBOUND * LDPACKER[0]
34: + DMAEN * WORD * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[1]
35: + DMAEN * /WORD * /BYTE * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[3]
36: + DMAEN * BYTE * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[1]
37: + RST
38:
39:RLDPACKER[1] := BYTE * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[1]
40: + /WORD * /BYTE * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[3]
41: + WORD * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[1]
42: + STATE[1] * STATE[0] * INBOUND * LDPACKER[1]
43: + DMAEN * WORD * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[1]
44: + DMAEN * /WORD * /BYTE * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[3]
45: + DMAEN * BYTE * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[1]
46: + RST
47:
48:RLDPACKER[2] := BYTE * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[1]
49: + /WORD * /BYTE * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[3]
50: + WORD * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[1]
51: + STATE[1] * STATE[0] * INBOUND * LDPACKER[2]
52: + RST
53:
54:RLDPACKER[3] := BYTE * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[1]
55: + /WORD * /BYTE * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[3]
56: + WORD * STATE[1] * /STATE[0] * /IOEND * EOT * INBOUND * /LDPACKER[1]
57: + DMAEN * WORD * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[1]
58: + DMAEN * /WORD * /BYTE * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[3]
59: + DMAEN * BYTE * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND * LDPACKER[1]
60: + RST
61:
62:RUNLDFIFO := /STATE[1] * STATE[0]
63: + RST
64:
65:RUNLDPACKER[0] := DMAEN * DMRQ * /STATE[1] * /STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[0]
66: + DMAEN * WORD * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[1]
67: + DMAEN * /WORD * /BYTE * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[3]
68: + DMAEN * BYTE * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[1]
69: + RST
70:
71:RUNLDPACKER[1] := DMAEN * DMRQ * /STATE[1] * /STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[1]
72: + DMAEN * WORD * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[1]
73: + DMAEN * /WORD * /BYTE * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[3]
74: + DMAEN * BYTE * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[1]
75: + RST
76:
77:RUNLDPACKER[2] := DMAEN * DMRQ * /STATE[1] * /STATE[0] * /IOACTIVE * /INBOUND * UNLDPACKER[2]
```



# HEWLETT PACKARD

```
78: + RST
79:
80: RUNLDPACKER[3] := DMAEN * WORD * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[1]
81: + DMAEN * /WORD * /BYTE * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[3]
82: + DMAEN * BYTE * /STATE[1] * /STATE[0] * /INBOUND * UNLDPACKER[1]
83: + RST
84:
85: /SLDFIFO := /WORD * /BYTE * STATE[1] * LDPACKER[3]
86: + BYTE * /STATE[1] * /LDPACKER[1]
87: + BYTE * STATE[1] * LDPACKER[1]
88: + /DMAEN * /STATE[1]
89: + /WORD * /BYTE * /STATE[1] * /LDPACKER[3]
90: + WORD * STATE[1] * LDPACKER[1]
91: + STATE[1] * /EOT
92: + STATE[1] * IOEND
93: + STATE[0]
94: + /STATE[1] * FIFOFULL
95: + /INBOUND
96: + WORD * /STATE[1] * /LDPACKER[1]
97:
98: /SLDPACKER[0] := BYTE
99: + /STATE[1]
100: + /STATE[0]
101: + /INBOUND
102: + LDPACKER[3]
103: + LDPACKER[2]
104: + LDPACKER[1]
105: + LDPACKER[0]
106:
107: /SLDPACKER[1] := /STATE[1]
108: + /STATE[0]
109: + /INBOUND
110: + /BYTE * /LDPACKER[0]
111: + LDPACKER[3] * /LDPACKER[0]
112: + LDPACKER[2] * /LDPACKER[0]
113: + LDPACKER[1] * /LDPACKER[0]
114:
115: /SLDPACKER[2] := /STATE[1]
116: + /STATE[0]
117: + /INBOUND
118: + /LDPACKER[1]
119:
120: /SLDPACKER[3] := /STATE[1]
121: + /STATE[0]
122: + /INBOUND
123: + /LDPACKER[2]
124:
125: /SODDO = BYTE * LDPACKER[1]
126: + /WORD * /BYTE * LDPACKER[3]
127: + /STATE[1]
128: + STATE[0]
129: + IOEND
130: + /EOT
131: + /INBOUND
132: + WORD * LDPACKER[1]
133: + /LDPACKER[2] * /LDPACKER[0]
134:
135: /SODD1 = /LDPACKER[2] * /LDPACKER[1]
136: + /STATE[1]
137: + STATE[0]
138: + IOEND
139: + /EOT
140: + /INBOUND
141: + /WORD * /BYTE * LDPACKER[3]
142: + WORD * LDPACKER[1]
143: + BYTE * LDPACKER[1]
144:
145: STATE[0] := STATE[1] * STATE[0] * /IOEND * /INBOUND
146: + DMAEN * DMRQ * /STATE[1] * /IOACTIVE * /FIFOEMPTY * /INBOUND * /UNLDPACKER[3] * /UNLDPACKER[1]
147: + DMAEN * DMRQ * /STATE[1] * /IOACTIVE * /INBOUND * UNLDPACKER[2]
148: + DMAEN * DMRQ * /WORD * /BYTE * /STATE[1] * /IOACTIVE * /INBOUND * UNLDPACKER[1]
149: + DMAEN * DMRQ * /STATE[1] * /IOACTIVE * /INBOUND * UNLDPACKER[0]
150: + DMAEN * /STATE[1] * STATE[0]
151: + DMAEN * DMRQ * BYTE * /STATE[1] * /IOACTIVE * INBOUND * /LDPACKER[1]
152: + DMAEN * DMRQ * /WORD * /BYTE * /STATE[1] * /IOACTIVE * INBOUND * /LDPACKER[3]
153: + DMAEN * DMRQ * WORD * /STATE[1] * /IOACTIVE * INBOUND * /LDPACKER[1]
154:
```



```
155:STATE[1] := STATE[1] * IOEND
156: + DMAEN * STATE[0] * /INBOUND
157: + STATE[1] * STATE[0]
158: + DMAEN * STATE[0] * IOEND
159:
160:/SUNLDFIFO := /DMAEN
161: + /DMRQ
162: + STATE[1]
163: + STATE[0]
164: + IOACTIVE
165: + FIFOEMPTY
166: + INBOUND
167: + UNLDPACKER[3]
168: + UNLDPACKER[2]
169: + UNLDPACKER[1]
170: + UNLDPACKER[0]
171:
172:/SUNLDPACKER[0] := /DMAEN
173: + /DMRQ
174: + BYTE
175: + STATE[1]
176: + STATE[0]
177: + IOACTIVE
178: + FIFOEMPTY
179: + INBOUND
180: + UNLDPACKER[3]
181: + UNLDPACKER[2]
182: + UNLDPACKER[1]
183: + UNLDPACKER[0]
184:
185:/SUNLDPACKER[1] := /DMAEN
186: + /DMRQ
187: + STATE[1]
188: + STATE[0]
189: + IOACTIVE
190: + INBOUND
191: + /BYTE * /UNLDPACKER[0]
192: + FIFOEMPTY * /UNLDPACKER[0]
193: + UNLDPACKER[3] * /UNLDPACKER[0]
194: + UNLDPACKER[2] * /UNLDPACKER[0]
195: + UNLDPACKER[1] * /UNLDPACKER[0]
196:
197:/SUNLDPACKER[2] := /DMAEN
198: + /DMRQ
199: + WORD
200: + BYTE
201: + STATE[1]
202: + STATE[0]
203: + IOACTIVE
204: + INBOUND
205: + /UNLDPACKER[1]
206:
207:/SUNLDPACKER[3] := /DMAEN
208: + /DMRQ
209: + STATE[1]
210: + STATE[0]
211: + IOACTIVE
212: + INBOUND
213: + /UNLDPACKER[2]
```

### 5.6 LS-HPiB Register Access, DMA, and Parallel poll Machine

```
1:pal LShpibSM;
2:
3:{
4:    PAL Designer low speed HPiB State Machine program.
5:
6:    This file generates the Next State Equations and Output
7:    equations for DIO register accesses, parallel polling and
8:    DMA sequence of the LS-HPiB portion of Nikki.
9:
10:    Don Soltis    v0.1    2/19/89
11:    Initial equations
12:
```



```

13:}
14:
15:input          CS, write, rst, a[5..0],
16:              state[2..0],
17:              bg,                {Bus grant from arbiter}
18:              CnchsBr,          {Bus request from Centronics}
19:              RS232cs,          {RS-232 Chip select}
20:              inppoll,          {in parallel poll}
21:              dtackin,          {for DMA use}
22:              DMAen,            {DMA enabled}
23:              dmack,            {DMA acknowledge}
24:              accrq,            {9914 DMA request}
25:              ppollen,          {parallel poll enable}
26:              match             {ppoll match}
27:              ;
28:
29:output         state[2..0],
30:              writeReg1, writeReg3, writeReg5,
31:              writeReg7, writeReg9,    { DIO reg writes }
32:
33:              readReg1, readReg3, readReg5,
34:              readReg7, readReg9,    { DIO reg reads }
35:
36:              dtack, OdioEn1,
37:              dmrdy,
38:              a9914cs, we, dbin, accgr,    {9914 signals}
39:              busReq,            {bus request to arbiter}
40:              Sinppoll,          {Set 'in ppoll' flip-flop}
41:              Rinppoll,          {Reset 'in ppoll' flip-flop}
42:              ppollir,
43:              OlsHpibEn,
44:              ls2dio
45:              ;
46:
47:
48:pal_type       'ASIC';
49:
50:{ ONLY the active low outputs need to be specified here for logic reduction }
51:{ use the psd version of pd2ddl for proper generation }
52:
53:state_vars     state[2..0];
54:
55:macro idle;    begin 0 end;
56:macro s0;      begin 1 end;
57:macro s1;      begin 3 end;
58:macro s2;      begin 2 end;
59:macro s3;      begin 6 end;
60:macro s4;      begin 7 end;
61:macro s5;      begin 5 end;
62:macro s6;      begin 4 end;
63:
64:{macro idle;  begin 0 end;
65:macro s0;      begin 4 end;
66:macro s1;      begin 7 end;
67:macro s2;      begin 3 end;
68:macro s3;      begin 2 end;
69:macro s4;      begin 6 end;
70:macro s5;      begin 1 end;
71:macro s6;      begin 5 end;}
72:
73:macro address; begin a[5..0] end;
74:
75:macro dio_reg; begin
76:    ( a[4] == 0 )
77:    end;
78:
79:macro a9914Reg; begin
80:    ( a[4] == 1 )
81:    end;
82:
83:macro next_state; begin state[2..0] end;
84:
85:Procedure enable_dio_reg;
86:    begin
87:        case a[3..1] of
88:            0: {reg 1}
89:                readReg1 = 1;

```



```
90:         1: {reg 3}
91:         readReg3 = 1;
92:         2: {reg 5}
93:         readReg5 = 1;
94:         3: {reg 7}
95:         readReg7 = 1;
96:         end;
97:         if a[3] = 1 then readReg9 = 1;
98:         end;
99:
100:Procedure write_dio_reg;
101:  begin
102:     case a[3..1] of
103:     0: {reg 1}
104:         writeReg1 = 1;
105:     1: {reg 3}
106:         writeReg3 = 1;
107:     2: {reg 5}
108:         writeReg5 = 1;
109:     3: {reg 7}
110:         writeReg7 = 1;
111:     end;
112:     if a[3] = 1 then writeReg9 = 1;
113:     end;
114:
115:     { ***** }
116:     { hold signals for DIO register transaction }
117:     { ***** }
118:Procedure do_reg_io;
119:  begin
120:     if (~write and ~inppoll) then OdioEn1 = 1;
121:
122:     if inppoll then begin
123:         a9914cs = 1;
124:         dbin = 1;
125:     end
126:     else begin {~inppoll}
127:         if a9914Reg then begin {a 9914 reg access, or p. poll}
128:             if write then begin
129:                 OlsHpibEn = 1;
130:                 a9914cs = 1;
131:                 dbin = 0;
132:                 if state[2..0] = s1 then
133:                     ve = 0
134:                 else
135:                     ve = 1;
136:                 end {if write}
137:             else {read 9914 reg} begin
138:                 ls2dio = 1;
139:                 a9914cs = 1;
140:                 dbin = 1;
141:             end;
142:         end
143:     else begin {dio_reg}
144:         if ~write then begin { read }
145:             enable_dio_reg;
146:         end
147:         end; {else dio_reg}
148:     end; {else ~inppoll}
149: end; {Procedure do_reg_io}
150:
151:Procedure do_dma_io;
152:  begin
153:     if write then OdioEn1 = 1;           {inbound DMA}
154:
155:     if ~write then begin                {outbound xfer}
156:         OlsHpibEn = 1;
157:         accGr = 1;
158:         dbin = 1;
159:         if state[2..0] = s1 then
160:             ve = 0
161:         else
162:             ve = 1;
163:         end {if write}
164:     else {inbound} begin
165:         ls2dio = 1;
166:         accGr = 1;
```



```
167:         dbin = 0;
168:         end;
169:     end; {Procedure do_dma_io}
170:
171:     { ***** }
172:     { Wait in ppoll s5 until we get a match, or we need to do a }
173:     { dio/dma request, or Centronics/RS-232 wants the bus }
174:     { ***** }
175: Procedure ppoll_wait;
176:     begin
177:         busReq = 1;
178:         if match then begin
179:             ppollir = 1;           {assert interrupt (set enable)}
180:             Rinppoll = 1;         {not in ppoll, won't come here again}
181:             next_state := s6;     {one more state before going to idle}
182:         end {match}
183:         else {~match} begin
184:             if CS or (dmack and DMAen) or RS232cs or CnccsBr then begin
185:                 Rinppoll = 1;     {Get out of ppoll}
186:                 next_state := s6;
187:             end
188:             else {CS, etc} begin
189:                 a9914cs = 1;
190:                 dbin = 1;
191:                 next_state := s5;
192:             end;
193:         end; {else ~match}
194:     end; {Procedure ppoll_wait}
195: {*****}
196: * Main
197: {*****}
198: begin
199:
200: if rst then begin         { Reset }
201:
202:     state[2..0]         := idle;
203:
204:     end;
205:
206: case state[2..0] of
207:
208:     { ***** }
209:     idle:         {idle states}
210:     begin
211:         if CS or (dmack and DMAen) then begin
212:             busReq = 1;
213:             next_state := s0;     {wait for bus}
214:         end
215:         else {nothing to do} begin
216:             if ppollEn and ~RS232cs then begin         {Start a ppoll}
217:                 Sinppoll = 1;     { I'm in ppoll}
218:                 busReq = 1;
219:                 next_state := s0;     {wait for bus}
220:             end
221:             else
222:                 next_state := idle;
223:             end; {else nothing to do}
224:         end; {in idle}
225:
226:     { ***** }
227:     s0:         { wait for bg and start a cycle }
228:     begin
229:         busReq = 1;
230:         if bg = 0 then begin
231:             { ***** }
232:             { if we get a CS or dmack, cancel ppoll }
233:             { ***** }
234:             if inppoll and (CS or (dmack and DMAen)) then begin
235:                 Rinppoll = 1;
236:                 next_state := s0;
237:             end
238:             else
239:                 next_state := s0;
240:             end
241:             else {bg = 1} begin
242:                 {*****}
243:                 { * Kludge for deglitching output Enable *}

```



```
244:             {*****}
245:             if( (CS=~write) or (dmack~write) ) then begin
246:                 OdioEn1 = 1;
247:                 if( ~a9914reg and CS ) then
248:                     ls2dio = 0
249:                 else
250:                     ls2dio = 1;
251:                 end;
252:                 if inppoll then begin
253:                     a9914cs = 1;
254:                     dbin = 1;
255:                 end;
256:                 next_state := s1;      {go for it}
257:             end;
258:         end; {case s0}
259:
260:             { ***** }
261:         s1:         { Set up 9914 signals }
262:         begin
263:             busReq = 1;
264:             if CS or inppoll then begin
265:                 do_reg_io;
266:                 next_state := s2;
267:             end
268:             else {~CS} begin
269:                 if (dmack and DMAen and ~inppoll) then begin
270:                     do_dma_io;
271:                     next_state := s2;
272:                 end
273:                 else
274:                     next_state := idle;          { Shouldn't be here }
275:                 end; {else ~CS}
276:             end; {case s1}
277:
278:             { ***** }
279:         s2:         { Give data time to settle }
280:         begin
281:             busReq = 1;
282:             if CS or inppoll then begin
283:                 do_reg_io;
284:                 next_state := s3;
285:             end
286:             else {~CS} begin
287:                 if (dmack and DMAen and ~inppoll) then begin
288:                     do_dma_io;
289:                     next_state := s3;
290:                 end
291:                 else
292:                     next_state := idle;          { Shouldn't be here }
293:                 end; {else ~CS}
294:             end; {case s2}
295:
296:             { ***** }
297:         s3:         { Continue to wait }
298:         begin
299:             busReq = 1;
300:             if CS or inppoll then begin
301:                 do_reg_io;
302:                 next_state := s4;
303:             end
304:             else {~CS} begin
305:                 if (dmack and DMAen and ~inppoll) then begin
306:                     do_dma_io;
307:                     if ~write~dtackin then
308:                         next_state := s3          {Wait for DTACKIN on outbound xfers}
309:                     else
310:                         next_state := s4;
311:                     end
312:                 else
313:                     next_state := idle;          { Shouldn't be here }
314:                 end; {else ~CS}
315:             end; {case s3}
316:
317:             { ***** }
318:         s4:         { Last state }
319:         begin
320:             busReq = 1;
```



```
321:   if CS or inppoll then begin
322:     if ~inppoll and ^a9914Reg and write then
323:       write_dio_reg;
324:     do_reg_io;
325:     next_state := s5;
326:   end
327:   else {^CS} begin
328:     if (dmack and DMAen and ^inppoll) then begin
329:       do_dma_io;
330:       next_state := s5;
331:     end
332:     else
333:       next_state := idle;           { Shouldn't be here }
334:     end; {else ^CS}
335:   end; {case s4}
336:
337:           { ***** }
338:   s5:     { finish cycle }
339:   begin
340:     busReq = 1;
341:     if CS and ^inppoll then begin
342:       if ^write then begin
343:         OdioEn1 = 1;
344:         if a9914Reg then ls2dio = 1
345:         else
346:           enable_dio_reg;
347:         end
348:         else
349:           if a9914Reg then OlsHpibEn = 1;
350:           dtack = 1;
351:           next_state := s5;           { wait for CS to go away }
352:         end
353:         else {^CS or inppoll} begin
354:           if (dmack and DMAen and ^inppoll) then begin
355:             dardy = 1;
356:             if write then begin {inbound}
357:               OdioEn1 = 1;
358:               ls2dio = 1;
359:             end
360:             else {outbound}
361:               OlsHpibEn = 1;
362:               next_state := s5;           { wait for dmack to go away }
363:             end
364:             else {inppoll or we're done} begin
365:               if inppoll then begin
366:                 ppoll_wait;
367:               end
368:               else {we're done}
369:                 next_state := idle;           { Done }
370:               end; {inppoll or done}
371:             end; {else ^CS}
372:           end; {case s5}
373:         s6:
374:         begin
375:           next_state := idle;
376:         end; {case s6}
377:       end;
378:     end;
379: end.
```





```
1:<no-name>
2:PAL Designer 2.2 88/05/05 14:27
3:Mon May 8 15:02:42 1989
4:PALASM Description for LSHPIBSM
5:
6:
7:;; PAL Attributes File is
8:;; Using structure info for "ASIC"
9:
10:A9914CS = /CS * STATE[2] * STATE[0] * /DMAEN * /MATCH * /CNCSBR * /RS232CS * INPPOLL
11: + /CS * STATE[2] * STATE[0] * /DMACK * /MATCH * /CNCSBR * /RS232CS * INPPOLL
12: + CS * A[4] * STATE[1]
13: + STATE[1] * INPPOLL
14: + /STATE[2] * STATE[0] * BG * INPPOLL
15:
16:ACCGR = /CS * STATE[1] * DMAEN * DMACK * /INPPOLL
17:
18:BUSREQ = STATE[0]
19: + STATE[1]
20: + /STATE[2] * /RS232CS * PPOLLEN
21: + CS * /STATE[2]
22: + /STATE[2] * DMAEN * DMACK
23:
24:DBIN = /CS * STATE[2] * STATE[0] * /DMAEN * /MATCH * /CNCSBR * /RS232CS * INPPOLL
25: + /CS * STATE[2] * STATE[0] * /DMACK * /MATCH * /CNCSBR * /RS232CS * INPPOLL
26: + /CS * /WRITE * STATE[1] * DMAEN * DMACK
27: + CS * /WRITE * A[4] * STATE[1]
28: + STATE[1] * INPPOLL
29: + /STATE[2] * STATE[0] * BG * INPPOLL
30:
31:DMRDY = /CS * STATE[2] * /STATE[1] * STATE[0] * DMAEN * DMACK * /INPPOLL
32:
33:DTACK = CS * STATE[2] * /STATE[1] * STATE[0] * /INPPOLL
34:
35:LS2DIO = /CS * WRITE * STATE[2] * STATE[0] * DMAEN * DMACK * /INPPOLL
36: + CS * /WRITE * A[4] * STATE[2] * STATE[0] * /INPPOLL
37: + /CS * WRITE * STATE[1] * DMAEN * DMACK * /INPPOLL
38: + CS * /WRITE * A[4] * STATE[1] * /INPPOLL
39: + /CS * WRITE * /STATE[2] * /STATE[1] * STATE[0] * BG * DMACK
40: + CS * /WRITE * A[4] * /STATE[2] * /STATE[1] * STATE[0] * BG
41: + WRITE * A[4] * /STATE[2] * /STATE[1] * STATE[0] * BG * DMACK
42:
43:ODIOEN1 = /CS * WRITE * STATE[2] * STATE[0] * DMAEN * DMACK * /INPPOLL
44: + CS * /WRITE * STATE[2] * STATE[0] * /INPPOLL
45: + /CS * WRITE * STATE[1] * DMAEN * DMACK * /INPPOLL
46: + CS * /WRITE * STATE[1] * /INPPOLL
47: + CS * /WRITE * /STATE[2] * /STATE[1] * STATE[0] * BG
48: + WRITE * /STATE[2] * /STATE[1] * STATE[0] * BG * DMACK
49:
50:OLSHPIBEN = /CS * /WRITE * STATE[2] * STATE[0] * DMAEN * DMACK * /INPPOLL
51: + CS * WRITE * A[4] * STATE[2] * STATE[0] * /INPPOLL
52: + /CS * /WRITE * STATE[1] * DMAEN * DMACK * /INPPOLL
53: + CS * WRITE * A[4] * STATE[1] * /INPPOLL
54:
55:PPOLLIR = STATE[2] * /STATE[1] * STATE[0] * MATCH * INPPOLL
56:
57:READREG1 = CS * /WRITE * /A[4] * /A[3] * /A[2] * /A[1] * STATE[2] * STATE[0] * /INPPOLL
58: + CS * /WRITE * /A[4] * /A[3] * /A[2] * /A[1] * STATE[1] * /INPPOLL
59:
60:READREG3 = CS * /WRITE * /A[4] * /A[3] * /A[2] * A[1] * STATE[2] * STATE[0] * /INPPOLL
61: + CS * /WRITE * /A[4] * /A[3] * /A[2] * A[1] * STATE[1] * /INPPOLL
62:
63:READREG5 = CS * /WRITE * /A[4] * /A[3] * A[2] * /A[1] * STATE[2] * STATE[0] * /INPPOLL
64: + CS * /WRITE * /A[4] * /A[3] * A[2] * /A[1] * STATE[1] * /INPPOLL
65:
66:READREG7 = CS * /WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[2] * STATE[0] * /INPPOLL
67: + CS * /WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[1] * /INPPOLL
68:
69:READREG9 = CS * /WRITE * /A[4] * A[3] * STATE[2] * STATE[0] * /INPPOLL
70: + CS * /WRITE * /A[4] * A[3] * STATE[1] * /INPPOLL
71:
72:RINPPOLL = CS * /STATE[1] * STATE[0] * /BG * INPPOLL
73: + /STATE[1] * STATE[0] * /BG * DMAEN * DMACK * INPPOLL
74: + CS * STATE[2] * /STATE[1] * STATE[0] * INPPOLL
75: + STATE[2] * /STATE[1] * STATE[0] * DMAEN * DMACK * INPPOLL
76: + STATE[2] * /STATE[1] * STATE[0] * RS232CS * INPPOLL
77: + STATE[2] * /STATE[1] * STATE[0] * CNCSBR * INPPOLL
```

```

78: + STATE[2] * /STATE[1] * STATE[0] * MATCH * INPPOLL
79:
80: SINPPOLL = /CS * /STATE[2] * /STATE[1] * /STATE[0] * /DMAEN * /RS232CS * PPOLLEN
81: + /CS * /STATE[2] * /STATE[1] * /STATE[0] * /DMACK * /RS232CS * PPOLLEN
82:
83: STATE[0] := /CS * STATE[2] * STATE[0] * /DMAEN * /MATCH * /CNCSBR * /RS232CS * INPPOLL
84: + /CS * STATE[2] * STATE[0] * /DMACK * /MATCH * /CNCSBR * /RS232CS * INPPOLL
85: + STATE[2] * STATE[0] * DMAEN * DMACK * /INPPOLL
86: + CS * STATE[2] * STATE[0] * /INPPOLL
87: + WRITE * STATE[2] * STATE[1] * DMAEN * DMACK
88: + STATE[2] * STATE[1] * DMAEN * DMACK * DTACKIN
89: + CS * STATE[2] * STATE[1]
90: + STATE[2] * STATE[1] * INPPOLL
91: + /STATE[2] * /STATE[1] * STATE[0]
92: + /STATE[2] * /STATE[1] * /RS232CS * PPOLLEN
93: + CS * /STATE[2] * /STATE[1]
94: + /STATE[2] * /STATE[1] * DMAEN * DMACK
95:
96: STATE[1] := STATE[1] * /STATE[0] * DMAEN * DMACK
97: + CS * STATE[1] * /STATE[0]
98: + STATE[1] * /STATE[0] * INPPOLL
99: + /STATE[2] * STATE[1] * DMAEN * DMACK
100: + CS * /STATE[2] * STATE[1]
101: + /STATE[2] * STATE[1] * INPPOLL
102: + /STATE[2] * /STATE[1] * STATE[0] * BG
103:
104: STATE[2] := STATE[2] * STATE[0] * DMAEN * DMACK
105: + CS * STATE[2] * STATE[0]
106: + STATE[2] * STATE[0] * INPPOLL
107: + STATE[1] * /STATE[0] * DMAEN * DMACK
108: + CS * STATE[1] * /STATE[0]
109: + STATE[1] * /STATE[0] * INPPOLL
110:
111: WE = /CS * /WRITE * STATE[2] * STATE[1] * DMAEN * DMACK * /INPPOLL
112: + CS * WRITE * A[4] * STATE[2] * STATE[1] * /INPPOLL
113: + /CS * /WRITE * STATE[1] * /STATE[0] * DMAEN * DMACK * /INPPOLL
114: + CS * WRITE * A[4] * STATE[1] * /STATE[0] * /INPPOLL
115:
116: WRITEREG1 = CS * WRITE * /A[4] * /A[3] * /A[2] * /A[1] * STATE[2] * STATE[1] * STATE[0] * /INPPOLL
117:
118: WRITEREG3 = CS * WRITE * /A[4] * /A[3] * /A[2] * A[1] * STATE[2] * STATE[1] * STATE[0] * /INPPOLL
119:
120: WRITEREG5 = CS * WRITE * /A[4] * /A[3] * A[2] * /A[1] * STATE[2] * STATE[1] * STATE[0] * /INPPOLL
121:
122: WRITEREG7 = CS * WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[2] * STATE[1] * STATE[0] * /INPPOLL
123:
124: WRITEREG9 = CS * WRITE * /A[4] * A[3] * STATE[2] * STATE[1] * STATE[0] * /INPPOLL

```

### 5.7 RS-232 Register Access Machine

```

1: pal RS232SM;
2: {
3:     PAL Designer RS232 control State Machine program.
4:
5:     This file generates the Next State Equations and Output
6:     equations for the RS232 interface.
7:
8:     Don Soltis      v0.1    2/27/89
9:     Initial equations
10:
11: }
12: input  cs,                {rs232 CS}
13:        Nwrite,            {High true read/Low true write}
14:        reset,            {High True reset signal}
15:        st[3..0],         {Output Terms Fed Back}
16:        ior,
17:        iow,
18:        a[4],
19:        a[1],
20:        busgr;
21:
22: output ior,                {Low true IO Read}
23:        iow,                {Low true IO Write}
24:        st[3..0],          {State Variables}

```



```
25:      dtack,          {Low True Data Transfer Acknowledge}
26:      ReadReg1,ReadReg3,  {read strobes}
27:      WriteReg1,WriteReg3, {write strobes}
28:      Sbusrq,Rbusrq,
29:      oEXTdataEn,
30:      OdioData1,
31:      ls2dio,
32:      uartcs;
33:
34:PAL_TYPE 'ASIC';
35:
36:STATE_VARS st[3], st[2], st[1], st[0];
37:
38:Macro goto;
39:begin
40:  st[3..0] :=
41:end;
42:
43:Macro state;
44:begin
45:  st[3..0]
46:end;
47:
48:Macro startstate;      begin 0 end;
49:Macro s0;              begin 8 end;
50:Macro s1;              begin 4 end;
51:Macro s2;              begin 1 end;
52:Macro s3;              begin 9 end;
53:Macro s4;              begin 13 end;
54:Macro s5;              begin 5 end;
55:Macro s6;              begin 3 end;
56:Macro s7;              begin 11 end;
57:Macro s8;              begin 15 end;
58:Macro s9;              begin 7 end;
59:Macro s10;             begin 12 end;
60:Macro s11;             begin 10 end;
61:Macro s12;             begin 2 end;
62:Macro s13;             begin 14 end;
63:Macro s14;             begin 6 end;
64:
65:macro an8250reg;       begin a[4] end;
66:macro aDio_reg;       begin ^a[4] end;
67:
68:BEGIN
69:case state of
70:
71:startstate: {Async. reset put the circuit here}
72:  begin
73:    if cs and an8250reg then begin
74:      Sbusrq = 1;          {ask for bus if cs}
75:      goto startstate;
76:    end;
77:    if cs and aDio_reg then {register xfer}
78:      goto s0;
79:    if cs and an8250reg and busgr then {start 8250 xfer}
80:      goto s0;           { have bus, will travel }
81:    if ^cs then begin
82:      Rbusrq = 1;        { no longer request }
83:      goto startstate;
84:    end;
85:  end;
86:
87:s0: begin
88:  if ^cs then
89:    begin {Error exit, two cards at same address}
90:      Rbusrq = 1;          {let go of bus}
91:      goto startstate;
92:      ior := 0;
93:      iow := 0;
94:      dtack := 0;
95:    end
96:  else
97:    begin
98:      if an8250reg then begin
99:        UartCs = 1;
100:        if Mwrite = 0 then {write}
101:          oEXTdataEn;
```



```
102:         goto s1;
103:         end
104:     else begin
105:         if Nwrite = 1 then begin
106:             if a[1] = 0 then
107:                 readReg1 = 1
108:             else
109:                 readReg3 = 1;
110:                 {OodioData1 = 1;}
111:             end;
112:             goto s1;
113:         end;
114:     end;
115: end;
116:
117:s1: begin
118:     if ~cs then
119:         begin {Error exit, two cards at same address}
120:             Rbusrq = 1;           {let go of bus}
121:             goto startstate;
122:             ior := 0;
123:             iow := 0;
124:             dtack := 0;
125:         end
126:     else begin
127:         if an8250reg then begin
128:             UartCs = 1;
129:             if Nwrite = 0 then {write}
130:                 oEXTdataEn
131:             else {READ} begin
132:                 OodioData1 = 1;
133:                 ls2dio = 1;
134:             end;
135:             goto s2;
136:         end
137:     else begin
138:         if Nwrite = 1 then begin
139:             if a[1] = 0 then
140:                 readReg1 = 1
141:             else
142:                 readReg3 = 1;
143:                 OodioData1 = 1;
144:             end;
145:             goto s2;
146:         end;
147:     end;
148: end;
149:
150:s2: begin
151:     if ~cs then
152:         begin {Error exit, two cards at same address}
153:             Rbusrq = 1;           {let go of bus}
154:             goto startstate;
155:             ior := 0;
156:             iow := 0;
157:             dtack := 0;
158:         end
159:     else
160:         begin
161:             if an8250reg then begin
162:                 UartCs = 1;
163:                 if Nwrite = 0 then
164:                     oEXTdataEn
165:                 else {READ} begin
166:                     OodioData1 = 1;
167:                     ls2dio = 1;
168:                 end;
169:                 goto s3;
170:             end
171:         else begin
172:             if Nwrite = 1 then begin
173:                 if a[1] = 0 then
174:                     readReg1 = 1
175:                 else
176:                     readReg3 = 1;
177:                     OodioData1 = 1;
178:                 end;
```



```
179:         goto s3;
180:         end;
181:     end;
182: end;
183:
184:s3: begin
185:     if ~cs then
186:     begin {Error exit, two cards at same address}
187:         Rbusrq = 1;           {let go of bus}
188:         goto startstate;
189:         ior := 0;
190:         iow := 0;
191:         dtack := 0;
192:     end
193:     else
194:     begin
195:         if an8250reg then begin
196:             if Nwrite = 0 then {write}
197:             oEXTdataEn
198:             else {READ} begin
199:                 OdioData1 = 1;
200:                 ls2dio = 1;
201:                 ior = 1;
202:             end;
203:             UartCs = 1;
204:             goto s4;
205:         end
206:         else begin
207:             if Nwrite = 1 then begin
208:                 if a[1] = 0 then
209:                     readReg1 = 1
210:                 else
211:                     readReg3 = 1;
212:                     OdioData1 = 1;
213:                 end
214:                 else begin           {write the DIO register}
215:                     if a[1] = 0 then
216:                         WriteReg1 = 1
217:                     else
218:                         WriteReg3 = 1;
219:                     end;
220:                     goto s10;
221:                 end;
222:             end;
223:         end;
224:
225:s4: begin
226:     if ~cs then
227:     begin {Error exit, two cards at same address}
228:         Rbusrq = 1;           {let go of bus}
229:         goto startstate;
230:         ior := 0;
231:         iow := 0;
232:         dtack := 0;
233:     end
234:     else
235:     begin
236:         if an8250reg then begin
237:             if Nwrite = 0 then begin {write}
238:                 iow = 1;
239:                 oEXTdataEn;
240:             end
241:             else {READ} begin
242:                 OdioData1 = 1;
243:                 ls2dio = 1;
244:                 ior = 1;
245:             end;
246:             UartCs = 1;
247:             goto s5;
248:         end
249:         else           {shouldn't be here}
250:             goto startstate;
251:         end;
252:     end;
253:
254:s5: begin
255:     if ~cs then
```



```
256:         begin {Error exit, two cards at same address}
257:           Rbusrq = 1;           {let go of bus}
258:           goto startstate;
259:           ior := 0;
260:           iow := 0;
261:           dtack := 0;
262:         end
263:       else
264:         begin
265:           if an8250reg then begin
266:             if Nwrite = 0 then begin {write}
267:               iow = 1;
268:               oEXTdataEn;
269:             end
270:           else {READ} begin
271:             OodioData1 = 1;
272:             ls2dio = 1;
273:             ior = 1;
274:           end;
275:           UartCs = 1;
276:           goto s6;
277:         end
278:       else
279:         goto startstate;
280:       end;
281:     end;
282:
283:s6: begin
284:   if ~cs then
285:     begin {Error exit, two cards at same address}
286:       Rbusrq = 1;           {let go of bus}
287:       goto startstate;
288:       ior := 0;
289:       iow := 0;
290:       dtack := 0;
291:     end
292:   else begin
293:     if an8250reg then begin
294:       if Nwrite = 0 then begin {write}
295:         iow = 1;
296:         oEXTdataEn;
297:       end
298:     else {READ} begin
299:       OodioData1 = 1;
300:       ls2dio = 1;
301:       ior = 1;
302:     end;
303:     UartCs = 1;
304:     goto s7;
305:   end
306:   else
307:     goto startstate;
308:   end;
309: end;
310:
311:s7: begin
312:   if ~cs then
313:     begin {Error exit, two cards at same address}
314:       Rbusrq = 1;           {let go of bus}
315:       goto startstate;
316:       ior := 0;
317:       iow := 0;
318:       dtack := 0;
319:     end
320:   else begin
321:     if an8250reg then begin
322:       UartCs = 1;
323:       if Nwrite = 0 then begin {write}
324:         iow = 1;
325:         oEXTdataEn;
326:         goto s10;           {finished}
327:       end
328:     else begin           {read}
329:       ior = 1;
330:       OodioData1 = 1;
331:       ls2dio = 1;
332:       goto s8;
```



```
333:         end;
334:         end
335:     else
336:         goto startstate;
337:     end;
338: end;
339:
340:s8: begin
341:     if ~cs then
342:
343:         begin {Error exit, two cards at same address}
344:             Rbusrq = 1;           {let go of bus}
345:             goto startstate;
346:             ior := 0;
347:             iow := 0;
348:             dtack := 0;
349:         end
350:     else begin
351:         if an8250reg then begin
352:             UartCs = 1;
353:             if Nwrite = 0 then begin {write, shouldn't be here}
354:                 goto startstate;
355:             end
356:             else begin           {read}
357:                 ior = 1;
358:                 OdioData1 = 1;
359:                 ls2dio = 1;
360:                 goto s9;
361:             end;
362:         end
363:     else
364:         goto startstate;
365:     end;
366: end;
367:
368:s9: begin
369:     if ~cs then
370:         begin {Error exit, two cards at same address}
371:             Rbusrq = 1;           {let go of bus}
372:             goto startstate;
373:             ior := 0;
374:             iow := 0;
375:             dtack := 0;
376:         end
377:     else
378:         begin
379:             if an8250reg then begin
380:                 UartCs = 1;
381:                 if Nwrite = 0 then begin {write, shouldn't be here}
382:                     goto startstate;
383:                 end
384:                 else begin           {read}
385:                     ior = 1;
386:                     OdioData1 = 1;
387:                     ls2dio = 1;
388:                     goto s10;
389:                 end;
390:             end
391:         else
392:             goto startstate;
393:         end;
394:     end; {case s9}
395:
396:s10: begin
397:     if ~cs then begin
398:         Rbusrq = 1;           {no longer need bus}
399:         goto startstate;
400:     end
401:     else begin {cs still asserted}
402:         dtack = 1;
403:         if an8250reg then begin
404:             UartCs = 1;
405:             if Nwrite = 0 then {write}
406:                 oEXITdataEn
407:             else begin
408:                 OdioData1 = 1;
409:                 ls2dio = 1;
```



```
410:                end;
411:                goto s1;
412:                end
413:                else begin          {internal register}
414:                    if Mwrite = 1 then begin
415:                        if a[1] = 0 then
416:                            readReg1 = 1
417:                        else
418:                            readReg3 = 1;
419:                            OdioData1 = 1;
420:                        end;
421:                    end; {else not an 8250 reg}
422:                goto s10;
423:                end; {else cs}
424:            end; {case s10}
425:
426:s11:                {Unused State}
427:                begin
428:                    goto startstate;
429:                end;
430:s12:                {Unused State}
431:                begin
432:                    goto startstate;
433:                end;
434:s13:                {Unused State}
435:                begin
436:                    goto startstate;
437:                end;
438:s14:                {Unused State}
439:                begin
440:                    goto startstate;
441:                end;
442:end; {end of case statement}
443:END.
```



```

1:<no-name>
2:PAL Designer 2.2 88/05/05 14:27
3:Tue May 2 07:57:01 1989
4:PALASM Description for RS232SM
5:
6:
7:;; PAL Attributes File is
8:;; Using structure info for "ASIC"
9:
10:DTACK = CS * ST[3] * ST[2] * /ST[1] * /ST[0]
11:
12:IOR = CS * ST[2] * ST[0] * A[4] * NWRITE
13: + CS * ST[1] * ST[0] * A[4] * NWRITE
14: + CS * ST[3] * ST[0] * A[4] * NWRITE
15:
16:LOW := CS * /ST[2] * ST[1] * ST[0] * A[4] * /NWRITE
17: + CS * ST[2] * /ST[1] * ST[0] * A[4] * /NWRITE
18:
19:LS2DIO = CS * ST[2] * /ST[1] * A[4] * NWRITE
20: + CS * ST[0] * A[4] * NWRITE
21:
22:ODIODATA1 = CS * ST[2] * /ST[1] * /ST[0] * NWRITE
23: + CS * ST[0] * A[4] * NWRITE
24: + CS * /ST[2] * /ST[1] * ST[0] * NWRITE
25:
26:OEXTDATAEN = CS * ST[3] * /ST[1] * A[4] * /NWRITE
27: + CS * ST[2] * /ST[1] * A[4] * /NWRITE
28: + CS * /ST[2] * ST[0] * A[4] * /NWRITE
29:
30:RBUSRQ = /CS * /ST[1]
31: + /CS * ST[0]
32:
33:READREG1 = CS * ST[2] * /ST[1] * /ST[0] * /A[4] * /A[1] * NWRITE
34: + CS * ST[3] * /ST[2] * /ST[1] * /A[4] * /A[1] * NWRITE
35: + CS * /ST[2] * /ST[1] * ST[0] * /A[4] * /A[1] * NWRITE
36:
37:READREG3 = CS * ST[2] * /ST[1] * /ST[0] * /A[4] * A[1] * NWRITE
38: + CS * ST[3] * /ST[2] * /ST[1] * /A[4] * A[1] * NWRITE
39: + CS * /ST[2] * /ST[1] * ST[0] * /A[4] * A[1] * NWRITE
40:
41:SBUSRQ = CS * /ST[3] * /ST[2] * /ST[1] * /ST[0] * A[4]
42:
43:ST[0] := CS * ST[3] * ST[0] * A[4] * NWRITE
44: + CS * /ST[3] * /ST[2] * ST[0] * A[4]
45: + CS * /ST[1] * ST[0] * A[4]
46: + CS * /ST[3] * /ST[2] * /ST[1] * ST[0]
47: + CS * /ST[3] * ST[2] * /ST[1] * /ST[0]
48:
49:ST[1] := CS * ST[3] * ST[1] * ST[0] * A[4] * NWRITE
50: + CS * /ST[3] * /ST[2] * ST[1] * ST[0] * A[4]
51: + CS * /ST[3] * ST[2] * /ST[1] * ST[0] * A[4]
52:
53:ST[2] := CS * ST[3] * /ST[1] * /ST[0]
54: + CS * ST[3] * /ST[1] * A[4]
55: + CS * ST[2] * ST[1] * ST[0] * A[4] * NWRITE
56: + CS * ST[3] * /ST[2] * ST[0] * A[4]
57: + CS * ST[3] * /ST[2] * /ST[1]
58:
59:ST[3] := CS * ST[3] * ST[2] * /ST[1] * /ST[0]
60: + CS * /ST[3] * ST[1] * ST[0] * A[4] * NWRITE
61: + CS * /ST[2] * ST[0] * A[4]
62: + CS * /ST[2] * /ST[1] * ST[0]
63: + CS * /ST[3] * /ST[2] * /ST[1] * BUSGR
64: + CS * /ST[3] * /ST[2] * /ST[1] * /A[4]
65:
66:UARTCS = CS * ST[3] * /ST[1] * A[4]
67: + CS * ST[2] * /ST[1] * A[4]
68: + CS * ST[0] * A[4]
69:
70:WRITEREG1 = CS * ST[3] * /ST[2] * /ST[1] * ST[0] * /A[4] * /A[1] * /NWRITE
71:
72:WRITEREG3 = CS * ST[3] * /ST[2] * /ST[1] * ST[0] * /A[4] * A[1] * /NWRITE

```

## 5.8 Centronics Register Access Machine

```

1:pal cnDIOsm;
2:
3:{
4:    PAL Designer Centronics DIO register state machine.
5:
6:    This file generates the Next State Equations and Output
7:    equations for the DIO register Centronics state machine.
8:
9:    Don Soltis    v0.1    3/7/89
10:   Initial equations
11:   Don Soltis    v1.1    10/15/89
12:   Increased length of Write to register 1 (soft reset)
13:
14:}
15:
16:input    Cs, write, a[5..0],
17:         busgr, FifoFull, FifoEmpty,
18:         inbound,
19:         state[2..0];
20:
21:output    SldFifo,RldFifo,SunldFifo,RunldFifo,
22:         busrq,
23:         olsDataEn,
24:         oDioDataEn1,
25:         cnCclk,
26:         dtack,
27:         ReadReg1,ReadReg3,ReadReg5,ReadReg7,ReadReg9,ReadReg11,
28:         WriteReg1,WriteReg3,WriteReg7,WriteReg9,
29:         state[2..0];
30:
31:
32:pal_type    'ASIC';
33:
34:{ ONLY the active low outputs need to be specified here for logic reduction }
35:{ use the psn version of pd2ddl for proper generation }
36:pal_pin_order    "SldFifo,"SunldFifo";
37:
38:state_vars    state[2..0];
39:
40:macro idle;    begin 0 end;
41:macro s0;    begin 1 end;
42:macro s1;    begin 5 end;
43:macro s2;    begin 6 end;
44:macro s3;    begin 2 end;
45:
46:macro next_state;    begin state[2..0] end;
47:
48:macro ExternalReg;    begin (a[4..1] = 3) end; {Register 7}
49:
50:macro FifoReg;    begin (a[4..1] = 5) end; {Register 11}
51:
52:procedure ReadDioReg; begin
53:    OdioDataEn1 = 1;
54:    case a[4..1] of
55:        0: begin {Reg 1}
56:            ReadReg1 = 1;
57:        end;
58:        1: begin {Reg 3}
59:            ReadReg3 = 1;
60:        end;
61:        2: begin {Reg 5}
62:            ReadReg5 = 1;
63:        end;
64:        3: begin {Reg 7}
65:            ReadReg7 = 1;
66:        end;
67:        4: begin {Reg 9}
68:            ReadReg9 = 1;
69:        end;
70:        5: begin {Reg 11}
71:            ReadReg11 = 1;

```



```
72:      RunldFifo = 1;
73:      end;
74:      end; {case}
75: end; {Procedure ReadDioReg}
76:
77: procedure WriteDioReg; begin
78:   case a[4..1] of
79:     0: begin {Reg 1}
80:       WriteReg1 = 1;
81:       end;
82:     1: begin {Reg 3}
83:       WriteReg3 = 1;
84:       end;
85:     3: begin {Reg 7}
86:       WriteReg7 = 1;
87:       end;
88:     4: begin {Reg 9}
89:       WriteReg9 = 1;
90:       end;
91:     5: begin {Reg 11}
92:       if ~inbound then SldFifo = 1;
93:       end;
94:   end; {case}
95: end; {Procedure WriteDioReg}
96:
97: {*****
98: * Main
99: *****}
100: begin
101:
102: case state[2..0] of
103:
104:   idle: begin
105:     if Cs then begin
106:       {*****}
107:       { Write }
108:       {*****}
109:     if write then begin
110:       if ~ExternalReg then begin
111:         next_state := s0;
112:         end
113:       else begin
114:         busrq = 1;
115:         next_state := s0;
116:         end;
117:       end {if write}
118:       {*****}
119:       { Read }
120:       {*****}
121:     else begin {read}
122:       if inbound*(FifoReg) then SunldFifo = 1;
123:       next_state := s0;
124:       end; {else read}
125:     end {if cs}
126:   else begin
127:     next_state := idle;
128:     end; {else not cs}
129:   end; {state idle}
130:
131:   s0: begin
132:     if Cs then begin
133:       {*****}
134:       { Write }
135:       {*****}
136:     if write then begin
137:       if ~ExternalReg then begin
138:         next_state := s1;
139:         end
140:       else begin
141:         busrq = 1;
142:         if busgr = 0 then next_state := s0
143:         else begin
144:           OlsDataEn = 1;
145:           next_state := s1;
146:           end;
147:         end;
148:       end {if write}
```



```
149:         {*****}
150:         { Read }
151:         {*****}
152:     else begin {read}
153:         ReadDioReg;
154:         next_state := s1;
155:     end; {else read}
156: end {if cs}
157: else begin
158:     RldFifo = 1;
159:     RunldFifo = 1;
160:     next_state := idle;
161: end; {else not cs}
162: end; {state s0}
163:
164: s1: begin
165:     if Cs then begin
166:         {*****}
167:         { Write }
168:         {*****}
169:         if write then begin
170:             if ~ExternalReg then begin
171:                 if( a[4..1] = 0 ) then WriteReg1 = 1;
172:                 next_state := s2;
173:             end
174:             else begin
175:                 busrq = 1;
176:                 cnCclk = 1;           {Load control into external latch}
177:                 OlsDataEn = 1;
178:                 next_state := s2;
179:             end;
180:         end {if write}
181:         {*****}
182:         { Read }
183:         {*****}
184:     else begin {read}
185:         ReadDioReg;
186:         next_state := s2;
187:     end; {else read}
188: end {if cs}
189: else begin
190:     RldFifo = 1;
191:     RunldFifo = 1;
192:     next_state := idle;
193: end; {else not cs}
194: end; {state s1}
195:
196: s2: begin
197:     if Cs then begin
198:         {*****}
199:         { Write }
200:         {*****}
201:         if write then begin
202:             WriteDioReg;
203:             if ~ExternalReg then begin
204:                 dtack = 1;
205:                 next_state := s3;
206:             end
207:             else begin
208:                 busrq = 1;
209:                 dtack = 1;
210:                 OlsDataEn = 1;
211:                 next_state := s3;
212:             end;
213:         end {if write}
214:         {*****}
215:         { Read }
216:         {*****}
217:     else begin {read}
218:         ReadDioReg;
219:         dtack = 1;
220:         next_state := s3;
221:     end; {else read}
222: end {if cs}
223: else begin           {Shouldn't happen}
224:     RldFifo = 1;
225:     RunldFifo = 1;
```



```
226:         next_state := idle;
227:         end; {else not cs}
228:     end; {state s2}
229:
230: s3: begin
231:     if Cs then begin
232:         {*****}
233:         { Write }
234:         {*****}
235:         if write then begin
236:             if ~ExternalReg then begin
237:                 RldFifo = 1;
238:                 dtack = 1;
239:                 next_state := s3;         {goto idle when cs goes away}
240:             end
241:             else begin
242:                 busrq = 0;
243:                 dtack = 1;
244:                 next_state := s3;         {goto idle when cs goes away}
245:             end;
246:             end {if write}
247:             {*****}
248:             { Read }
249:             {*****}
250:             else begin {read}
251:                 ReadDioReg;
252:                 dtack = 1;
253:                 next_state := s3;
254:                 end; {else read}
255:             end {if cs}
256:             else begin
257:                 next_state := idle;
258:                 end; {else not cs}
259:             end; {state s3}
260:         end;
261:     end; {Case state of}
262: end.
```

```

1:<no-name>
2:PAL Designer 2.2 88/05/05 14:27
3:Sun Oct 15 14:57:29 1989
4:PALASM Description for CNDIOSM
5:/SLDFIFO
6:/SUNLDFIFO
7:
8;;; PAL Attributes File is
9;;; Using structure info for "ASIC"
10:
11:BUSRQ = CS * WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[2] * STATE[1] * /STATE[0]
12: + CS * WRITE * /A[4] * /A[3] * A[2] * A[1] * /STATE[1] * STATE[0]
13: + CS * WRITE * /A[4] * /A[3] * A[2] * A[1] * /STATE[2] * /STATE[1]
14:
15:CNCCLK = CS * WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[2] * /STATE[1] * STATE[0]
16:
17:DTACK = CS * STATE[1] * /STATE[0]
18:
19:ODIODATAEN1 = CS * /WRITE * STATE[1] * /STATE[0]
20: + CS * /WRITE * /STATE[1] * STATE[0]
21:
22:OLSDATAEN = CS * WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[2] * STATE[1] * /STATE[0]
23: + CS * WRITE * /A[4] * /A[3] * A[2] * A[1] * BUSGR * /STATE[1] * STATE[0]
24: + CS * WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[2] * /STATE[1] * STATE[0]
25:
26:READREG1 = CS * /WRITE * /A[4] * /A[3] * /A[2] * /A[1] * STATE[1] * /STATE[0]
27: + CS * /WRITE * /A[4] * /A[3] * /A[2] * /A[1] * /STATE[1] * STATE[0]
28:
29:READREG11 = CS * /WRITE * /A[4] * A[3] * /A[2] * A[1] * STATE[1] * /STATE[0]
30: + CS * /WRITE * /A[4] * A[3] * /A[2] * A[1] * /STATE[1] * STATE[0]
31:
32:READREG3 = CS * /WRITE * /A[4] * /A[3] * /A[2] * A[1] * STATE[1] * /STATE[0]
33: + CS * /WRITE * /A[4] * /A[3] * /A[2] * A[1] * /STATE[1] * STATE[0]
34:
35:READREG5 = CS * /WRITE * /A[4] * /A[3] * A[2] * /A[1] * STATE[1] * /STATE[0]
36: + CS * /WRITE * /A[4] * /A[3] * A[2] * /A[1] * /STATE[1] * STATE[0]
37:
38:READREG7 = CS * /WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[1] * /STATE[0]
39: + CS * /WRITE * /A[4] * /A[3] * A[2] * A[1] * /STATE[1] * STATE[0]
40:
41:READREG9 = CS * /WRITE * /A[4] * A[3] * /A[2] * /A[1] * STATE[1] * /STATE[0]
42: + CS * /WRITE * /A[4] * A[3] * /A[2] * /A[1] * /STATE[1] * STATE[0]
43:
44:RLDFIFO = CS * WRITE * A[4] * /STATE[2] * STATE[1] * /STATE[0]
45: + CS * WRITE * A[3] * /STATE[2] * STATE[1] * /STATE[0]
46: + CS * WRITE * /A[2] * /STATE[2] * STATE[1] * /STATE[0]
47: + CS * WRITE * /A[1] * /STATE[2] * STATE[1] * /STATE[0]
48: + /CS * STATE[2] * STATE[1] * /STATE[0]
49: + /CS * /STATE[1] * STATE[0]
50:
51:RUNLDFIFO = CS * /WRITE * /A[4] * A[3] * /A[2] * A[1] * STATE[1] * /STATE[0]
52: + /CS * STATE[2] * STATE[1] * /STATE[0]
53: + /WRITE * /A[4] * A[3] * /A[2] * A[1] * /STATE[1] * STATE[0]
54: + /CS * /STATE[1] * STATE[0]
55:
56:/SLDFIFO = /CS
57: + /WRITE
58: + A[4]
59: + /A[3]
60: + A[2]
61: + /A[1]
62: + /STATE[2]
63: + /STATE[1]
64: + STATE[0]
65: + INBOUND
66:
67:STATE[0] := CS * /STATE[2] * /STATE[1]
68:
69:STATE[1] := CS * STATE[1] * /STATE[0]
70: + CS * STATE[2] * /STATE[1] * STATE[0]
71:
72:STATE[2] := CS * BUSGR * /STATE[1] * STATE[0]
73: + CS * A[4] * /STATE[1] * STATE[0]
74: + CS * A[3] * /STATE[1] * STATE[0]
75: + CS * /A[2] * /STATE[1] * STATE[0]
76: + CS * /A[1] * /STATE[1] * STATE[0]
77: + CS * /WRITE * /STATE[1] * STATE[0]

```

```

78: + CS * STATE[2] * /STATE[1] * STATE[0]
79:
80:/SUNLDFIFO = /CS
81: + WRITE
82: + A[4]
83: + /A[3]
84: + A[2]
85: + /A[1]
86: + STATE[2]
87: + STATE[1]
88: + STATE[0]
89: + /INBOUND
90:
91:WRITEREG1 = CS * WRITE * /A[4] * /A[3] * /A[2] * /A[1] * STATE[2] * /STATE[1] * STATE[0]
92: + CS * WRITE * /A[4] * /A[3] * /A[2] * /A[1] * STATE[2] * STATE[1] * /STATE[0]
93:
94:WRITEREG3 = CS * WRITE * /A[4] * /A[3] * /A[2] * A[1] * STATE[2] * STATE[1] * /STATE[0]
95:
96:WRITEREG7 = CS * WRITE * /A[4] * /A[3] * A[2] * A[1] * STATE[2] * STATE[1] * /STATE[0]
97:
98:WRITEREG9 = CS * WRITE * /A[4] * A[3] * /A[2] * /A[1] * STATE[2] * STATE[1] * /STATE[0]

```

### 5.9 Centronics FIFO-CENTRONICS (back end) Machine

```

1:pal cnBEsm;
2:
3:{
4:    PAL Designer back end (Centronics-FIFO) Centronics state machine.
5:
6:    This file generates the Next State Equations and Output
7:    equations for the BE Centronics state machine.
8:
9:    Don Soltis    v0.1    3/7/89
10:        Initial equations
11:
12:}
13:
14:
15:input    FifoEmpty,
16:        FifoFull,
17:        inbound,
18:        iStrobe,
19:        iBusy,
20:        iAck,
21:        busgr,
22:        slow,
23:        timeout,
24:        outbound,
25:        state[2..0];
26:
27:output    SldFifo,RldFifo,SunldFifo,RunldFifo,
28:        oStrobe,oBusy,
29:        busrq,
30:        olsDataEn,
31:        startTimer,
32:        cnDclk,
33:        Southbound,
34:        Routbound,
35:        state[2..0];
36:
37:
38:pal_type    'ASIC';
39:
40:{ ONLY the active low outputs need to be specified here for logic reduction }
41:{ use the psn version of pd2ddl for proper generation }
42:pal_pin_order    "SldFifo,"SunldFifo,"Southbound;
43:
44:state_vars    state[2..0];
45:
46:macro idle;    begin 0 end;
47:macro s0;    begin 4 end;
48:macro s1;    begin 6 end;
49:macro s2;    begin 5 end;
50:macro s3;    begin 1 end;

```



```
51:macro s4;                begin 3 end;
52:macro s5;                begin 2 end;
53:
54:macro next_state;        begin state[2..0] end;
55:
56:{*****}
57: * Main
58: {*****}
59:begin
60:
61:case state[2..0] of
62:
63:  idle: begin
64:    {*****}
65:    { OUTBOUND }
66:    {*****}
67:    if ~inbound then begin
68:      if ~FifoEmpty*~iBusy then begin
69:        busrq = 1;
70:        Southbound;
71:        next_state := s0; {wait to get bus}
72:        end {if fifo not empty}
73:      else {fifo is empty, or Centronics is Busy} begin
74:        next_state := idle;
75:        end;
76:      end {if outbound}
77:      {*****}
78:      { INBOUND }
79:      {*****}
80:    else begin {inbound}
81:      oBusy = 1;
82:      if ~FifoFull*~Slow then begin
83:        Routbound;
84:        next_state := s0;
85:        end
86:      else begin
87:        {*** SLOW MODE, wait for fifo empty ***}
88:        if FifoEmpty*Slow then begin
89:          Routbound;
90:          next_state := s0;
91:          end
92:        else
93:          next_state := idle;
94:        end;
95:      end; {else inbound}
96:    end; {state idle}
97:
98:  s0: begin
99:    {*****}
100:   { OUTBOUND }
101:   {*****}
102:   if ~inbound*outbound then begin
103:     busrq = 1;
104:     if ~busgr then begin
105:       next_state := s0; {Wait for bus}
106:       end
107:     else begin {have bus}
108:       OlsDataEn = 1;
109:       next_state := s1;
110:       end;
111:     end {if outbound}
112:     {*****}
113:     { INBOUND }
114:     {*****}
115:   else if inbound*~outbound then begin
116:     oBusy = 0;
117:     if iStrobe = 0 then next_state := s0
118:     else begin
119:       busrq = 1;
120:       next_state := s1;
121:       end; {else iStrobe = 1}
122:     end {else inbound}
123:   else
124:     next_state := idle;
125:   end; {state s0}
126:
127:  s1: begin
```





```
128:      {*****}
129:      { OUTBOUND }
130:      {*****}
131:      if ~inbound*outbound then begin
132:          busrq = 1;
133:          if ~busgr then begin {SHOULDN'T be here}
134:              next_state := s0;
135:          end
136:          else begin {have bus}
137:              OlsDataEn = 1;
138:              cnDclk = 1;          {Externally latch data}
139:              startTimer = 1;     {start up 1microsecond timer}
140:              SunldFifo = 1;     {Clock fifo beginning of s2}
141:              next_state := s2;
142:          end;
143:          end {if outbound}
144:      {*****}
145:      { INBOUND }
146:      {*****}
147:      else if inbound*~outbound then begin
148:          busrq = 1;
149:          if ~busgr then next_state := s1
150:          else next_state := s2;
151:          end {else inbound}
152:      else
153:          next_state := idle;
154:      end; {state s1}
155:
156:      s2: begin
157:          {*****}
158:          { OUTBOUND }
159:          {*****}
160:          if ~inbound*outbound then begin
161:              busrq = 0;          {all done}
162:              if ~timeout then begin
163:                  next_state := s2;          {Wait for 1microsec data setup}
164:              end
165:              else begin {Got timeout, advance}
166:                  RunldFifo = 1;          {advance FIFO}
167:                  startTimer = 1;        {Start up timer again, min strobe width}
168:                  oStrobe = 1;
169:                  next_state := s3;
170:              end;
171:              end {if outbound}
172:          {*****}
173:          { INBOUND }
174:          {*****}
175:          else if inbound*~outbound then begin
176:              busrq = 1;
177:              cnDclk = 1;          { enable data in }
178:              {SldFifo = 1;}      {ldFifo start of s3 }
179:              next_state := s3;
180:          end {else inbound}
181:          else
182:              next_state := idle;
183:          end; {state s2}
184:
185:      s3: begin
186:          {*****}
187:          { OUTBOUND }
188:          {*****}
189:          if ~inbound*outbound then begin
190:              if timeout and iBusy then begin          {done with strobe}
191:                  next_state := s4;
192:              end
193:              else begin
194:                  oStrobe = 1;
195:                  next_state := s3;          {Wait for busy}
196:              end;
197:              end {if outbound}
198:          {*****}
199:          { INBOUND }
200:          {*****}
201:          else if inbound*~outbound then begin
202:              busrq = 1;
203:              cnDclk = 1;
204:              SldFifo = 1;
```



```
205:         oBusy = 1;                               { The sooner the better ? }
206:         next_state := s4;
207:         end {else inbound}
208:     else
209:         next_state := idle;
210:     end; {state s3}
211:
212: s4: begin
213:     {*****}
214:     { OUTBOUND }
215:     {*****}
216:     if ~inbound*outbound then begin
217:         if ~Slow then begin
218:             if iAck then next_state := s5           {Wait for Ack}
219:             else next_state := s4;
220:             end
221:         else begin
222:             next_state := idle;                     {ignore Ack}
223:             end; {else Slow}
224:         end {if outbound}
225:         {*****}
226:         { INBOUND }
227:         {*****}
228:     else if inbound*~outbound then begin
229:         RldFifo = 1;
230:         busrq = 0;           {Don't need it anymore}
231:         oBusy = 1;          {Keep busy asserted until iStrobe goes away}
232:         next_state = s5;
233:         end {else inbound}
234:     else
235:         next_state := idle;
236:     end; {state s4}
237:
238: s5: begin
239:     {*****}
240:     { OUTBOUND }
241:     {*****}
242:     if ~inbound*outbound then begin
243:         if ~iAck then next_state := idle           {Wait for Ack to go away}
244:         else next_state := s5;
245:         end {if outbound}
246:         {*****}
247:         { INBOUND }
248:         {*****}
249:     else if inbound*~outbound then begin
250:         oBusy = 1;
251:         if iStrobe = 1 then next_state := s5
252:         else next_state := idle;
253:         end {else inbound}
254:     else
255:         next_state := idle;
256:     end; {state s5}
257:
258: end; {Case state}
259: end.
```

```
1:<no-name>
2:PAL Designer 2.2 88/05/05 14:27
3:Fri May 12 22:01:05 1989
4:PALASM Description for CNBESM
5:/SLDFIFO
6:/SUNLDFIFO /SOUTBOUND
7:
8;;; PAL Attributes File is
9;;; Using structure info for "ASIC"
10:
11:BUSRQ = /STATE[1] * STATE[0] * INBOUND * /OUTBOUND
12: + STATE[2] * /STATE[0] * INBOUND * ISTROBE * /OUTBOUND
13: + STATE[2] * STATE[1] * /STATE[0] * INBOUND * /OUTBOUND
14: + STATE[2] * /STATE[0] * /INBOUND * OUTBOUND
15: + /IBUSY * /STATE[2] * /STATE[1] * /STATE[0] * /FIFOEMPTY * /INBOUND
16:
17:CNDCLK = /STATE[1] * STATE[0] * INBOUND * /OUTBOUND
18: + BUSGR * STATE[2] * STATE[1] * /STATE[0] * /INBOUND * OUTBOUND
19:
20:OBUYS = /STATE[2] * INBOUND * /OUTBOUND
21: + /STATE[2] * /STATE[1] * /STATE[0] * INBOUND
22:
23:OLSDATAEN = BUSGR * STATE[2] * /STATE[0] * /INBOUND * OUTBOUND
24:
25:OSTROBE = /IBUSY * /STATE[2] * /STATE[1] * STATE[0] * /INBOUND * OUTBOUND
26: + /STATE[2] * /STATE[1] * STATE[0] * /INBOUND * /TIMEOUT * OUTBOUND
27: + STATE[2] * /STATE[1] * STATE[0] * /INBOUND * TIMEOUT * OUTBOUND
28:
29:RLDFIFO = /STATE[2] * STATE[1] * STATE[0] * INBOUND * /OUTBOUND
30:
31:ROUTBOUND = SLOW * /STATE[2] * /STATE[1] * /STATE[0] * FIFOEMPTY * INBOUND
32: + /SLOW * /STATE[2] * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND
33:
34:RUNLDFIFO = STATE[2] * /STATE[1] * STATE[0] * /INBOUND * TIMEOUT * OUTBOUND
35:
36:/SLDFIFO = STATE[2]
37: + STATE[1]
38: + /STATE[0]
39: + /INBOUND
40: + OUTBOUND
41:
42:/SOUTBOUND = IBUSY
43: + STATE[2]
44: + STATE[1]
45: + STATE[0]
46: + FIFOEMPTY
47: + INBOUND
48:
49:STARTTIMER = STATE[2] * /STATE[1] * STATE[0] * /INBOUND * TIMEOUT * OUTBOUND
50: + BUSGR * STATE[2] * STATE[1] * /STATE[0] * /INBOUND * OUTBOUND
51:
52:STATE[0] := /IACK * /SLOW * /STATE[2] * STATE[0] * /INBOUND * OUTBOUND
53: + /STATE[1] * STATE[0] * INBOUND * /OUTBOUND
54: + /STATE[1] * STATE[0] * /INBOUND * OUTBOUND
55: + BUSGR * STATE[2] * STATE[1] * /STATE[0] * INBOUND * /OUTBOUND
56: + BUSGR * STATE[2] * STATE[1] * /STATE[0] * /INBOUND * OUTBOUND
57:
58:STATE[1] := /STATE[2] * STATE[1] * INBOUND * ISTROBE * /OUTBOUND
59: + IACK * /STATE[2] * STATE[1] * /STATE[0] * /INBOUND * OUTBOUND
60: + /STATE[2] * STATE[0] * INBOUND * /OUTBOUND
61: + /SLOW * /STATE[2] * STATE[1] * STATE[0] * /INBOUND * OUTBOUND
62: + IBUSY * /STATE[2] * /STATE[1] * STATE[0] * /INBOUND * TIMEOUT * OUTBOUND
63: + /BUSGR * STATE[2] * STATE[1] * /STATE[0] * INBOUND * /OUTBOUND
64: + STATE[2] * /STATE[1] * /STATE[0] * INBOUND * ISTROBE * /OUTBOUND
65: + BUSGR * STATE[2] * /STATE[1] * /STATE[0] * /INBOUND * OUTBOUND
66:
67:STATE[2] := STATE[2] * /STATE[1] * /INBOUND * /TIMEOUT * OUTBOUND
68: + STATE[2] * /STATE[0] * INBOUND * /OUTBOUND
69: + STATE[2] * /STATE[0] * /INBOUND * OUTBOUND
70: + SLOW * /STATE[2] * /STATE[1] * /STATE[0] * FIFOEMPTY * INBOUND
71: + /SLOW * /STATE[2] * /STATE[1] * /STATE[0] * /FIFOFULL * INBOUND
72: + /IBUSY * /STATE[2] * /STATE[1] * /STATE[0] * /FIFOEMPTY * /INBOUND
73:
74:/SUNLDFIFO = /BUSGR
75: + /STATE[2]
76: + /STATE[1]
77: + STATE[0]
```



78: + INBOUND  
79: + /OUTBOUND

### 5.10 Centronics DIO-FIFO DMA Machine

```

1:pal cnDMAasm;
2:
3:{
4:    PAL Designer Centronics DIO-DMA State Machine program.
5:
6:    This file generates the Next State Equations and Output
7:    equations for the DIO-DMA half of the Centronics interface.
8:
9:    Don Soltis      v0.1    3/8/89
10:        Initial equations
11:
12:}
13:
14:input          DMAen,FifoFull,FifoEmpty,inbound,
15:              done,rst,
16:              Udmack,          {Unsynced dmack}
17:              dmack,dmrq,dtacki,
18:              state[2..0];
19:
20:output        state[2..0],
21:              SldFifo,          { set ldFifo }
22:              RldFifo,          { reset ldFifo }
23:              SunldFifo,        { set unldFifo }
24:              RunldFifo,        { reset unldFifo }
25:              Sdmrq,            { set DMA request }
26:              Rdmrq,            { reset DMA request }
27:              oDataEn,          { Output data Enable }
28:              dmrdr;
29:
30:pal_type      'ASIC';
31:pal_pin_order ~SldFifo,~SunldFifo,~Sdmrq;
32:
33:state_vars    state[2..0];
34:
35:macro idle;   begin 0 end;
36:
37:macro inb0;   begin 1 end; {Inbound state 0}
38:macro inb1;   begin 3 end; {Inbound state 1}
39:macro inb2;   begin 2 end; {Inbound state 2}
40:
41:macro outb0;  begin 4 end; {outbound state 0}
42:macro outb1;  begin 5 end; {outbound state 1}
43:macro outb2;  begin 6 end; {outbound state 2}
44:
45:macro next_state; begin state[2..0] end;
46:
47:begin
48:
49:if rst then begin { Reset }
50:
51:  state[2..0] := idle;
52:  Rdmrq      := 1;
53:  RldFifo    := 1;
54:  RunldFifo  := 1;
55:
56:  end;
57:
58:case state[2..0] of
59:
60:  idle:      {idle states}
61:    begin
62:
63:      if DMAen then begin { Make sure DMA is enabled }
64:
65:        {*****}
66:        { start inbound if Fifo not empty }
67:        {*****}
68:        if( inbound AND ~FifoEmpty ) then begin
69:          SunldFifo := 1;

```



```
70:         Sdmrq      := 1;
71:         next_state := inb0;
72:         end
73:
74:         {*****}
75:         { start outbound if Fifo is not Full }
76:         {*****}
77:         else if( ~inbound AND ~FifoFull ) then begin
78:             Sdmrq      := 1;
79:             next_state := outb0;
80:             end
81:         else
82:             next_state := idle;
83:         end {if DMAen}
84:     end {if DMAen}
85:     else
86:         next_state := idle;
87:     end; {in idle0}
88:
89: inb0: begin
90:     if DMAen then begin
91:         if dmack then
92:             next_state := inb1
93:         else
94:             next_state := inb0; { wait for dmack }
95:         end
96:     else begin
97:         next_state := idle;
98:         Rdmrq := 1;      {don't dmack}
99:         end;
100:
101:     end; {case inb0}
102:
103: inb1: begin
104:     if FifoEmpty then
105:         Rdmrq := 1;      {no more DMA requests}
106:     end;
107:
108:     oDataEn := 1;
109:     RunldFifo := 1;
110:     next_state := inb2;
111:     end; {case inb1}
112:
113: inb2: begin
114:     oDataEn := 1;
115:     if Udmack then dmardy := 1; {must release dmardy 0-50ns after dmack}
116:     if dmack then begin
117:         next_state := inb2
118:     end
119:     else begin
120:         if ~dmack then begin
121:             next_state := idle
122:         end
123:         else begin
124:             SunldFifo := 1;      { Another transfer }
125:             next_state := inb0;
126:         end;
127:     end;
128:     end; {case inb2}
129:
130: outb0: begin
131:     if DMAen then begin
132:         if dmack then begin
133:             Rdmrq := 1; {unrequest until later}
134:             if dtacki then begin
135:                 next_state := outb1
136:             end
137:             else
138:                 next_state := outb0;
139:             end
140:         else
141:             next_state := outb0; { wait for dmack }
142:         end
143:     end
144:     else begin
```



```
147:     next_state := idle;
148:     Rdmrq := 1;      {don't dmrq}
149:     end;
150:
151:     end; {case outb0}
152:
153:     outb1: begin
154:         dardy := 1;          {data hold 85 ns after this}
155:         SldFifo := 1;       {Actually occurs in next state}
156:         next_state := outb2;
157:     end; {case outb1}
158:
159:     outb2: begin
160:         if Udmack then dardy := 1; {must release dardy 0-50ns after dmack}
161:         RldFifo := 1;
162:         if dmack then
163:             next_state := outb2
164:         else begin
165:             if FifoFull then
166:                 next_state := idle
167:             else begin
168:                 Sdmrq := 1;
169:                 next_state := outb0;
170:             end;
171:         end;
172:
173:     end; {case outb2}
174:
175: end; { case }
176: end. {END}
```



# HEWLETT PACKARD

```
1:<no-name>
2:PAL Designer 2.2 88/05/05 14:27
3:Wed Mar 8 09:48:13 1989
4:PALASH Description for CNDMASH
5:/SLDFIFO
6:/SUNLDFIFO /SDMRQ
7:
8;;; PAL Attributes File is
9;;; Using structure info for "ASIC"
10:
11:DMRDY := STATE[1] * /STATE[0] * UDMACK
12: + STATE[2] * /STATE[1] * STATE[0]
13:
14:ODATAEN := /STATE[2] * STATE[1]
15:
16:RDMRQ := DMACK * STATE[2] * /STATE[1] * /STATE[0]
17: + /DMAEN * STATE[2] * /STATE[1] * /STATE[0]
18: + /STATE[2] * STATE[1] * STATE[0] * FIFOEMPTY
19: + /DMAEN * /STATE[2] * /STATE[1] * STATE[0]
20: + RST
21:
22:RLDFIFO := STATE[2] * STATE[1] * /STATE[0]
23: + RST
24:
25:RUNLDFIFO := /STATE[2] * STATE[1] * STATE[0]
26: + RST
27:
28:/SDMRQ := /DMAEN * /STATE[1]
29: + /STATE[2] * /STATE[1]
30: + STATE[1] * FIFOFULL
31: + DMACK * STATE[1]
32: + /STATE[2] * STATE[1]
33: + STATE[0]
34: + /STATE[1] * FIFOEMPTY * INBOUND
35: + FIFOFULL * /INBOUND
36:
37:/SLDFIFO := /STATE[2]
38: + STATE[1]
39: + /STATE[0]
40:
41:STATE[0] := DMAEN * DMACK * STATE[2] * /STATE[1] * /STATE[0] * DTACKI
42: + /DMACK * DMRQ * /STATE[2] * STATE[1] * /STATE[0]
43: + DMAEN * /STATE[2] * /STATE[1] * /FIFOEMPTY * INBOUND
44: + DMAEN * /STATE[2] * /STATE[1] * STATE[0]
45:
46:STATE[1] := DMACK * STATE[1] * /STATE[0]
47: + STATE[2] * /STATE[1] * STATE[0]
48: + DMAEN * DMACK * /STATE[2] * STATE[0]
49: + /STATE[2] * STATE[1] * STATE[0]
50:
51:STATE[2] := STATE[2] * STATE[1] * /STATE[0] * /FIFOFULL
52: + DMACK * STATE[2] * STATE[1] * /STATE[0]
53: + DMAEN * STATE[2] * /STATE[1]
54: + STATE[2] * /STATE[1] * STATE[0]
55: + DMAEN * /STATE[1] * /STATE[0] * /FIFOFULL * /INBOUND
56:
57:/SUNLDFIFO := STATE[0]
58: + /DMAEN * /STATE[1]
59: + STATE[2]
60: + DMACK * STATE[1]
61: + /DMRQ * STATE[1]
62: + /STATE[1] * FIFOEMPTY
63: + /STATE[1] * /INBOUND
```