Lattice™ Semiconductor Corporation

PCI Bus Implementation

Introduction

The Peripheral Component Interconnect (PCI) local bus was designed as a high-bandwidth bus that provides a data path between the CPU and multiple high-performance peripherals. Proposed as a total system solution, PCI provides interconnects to networks, disk drives, video and other high speed peripherals. Processor independence allows the PCI bus to be optimized for I/O functions and enables concurrent operation of the local bus with the processor/memory subsystem. A 32-bit synchronous bus that provides data throughput of 132 Mbytes/sec, the PCI bus is expandable up to a 64-bit data path which doubles the throughput. On account of its futuristic processor independent orientation, PCI allows manufacturers to significantly trim development costs by not having to completely redesign every product cycle.

This ties in elegantly with Lattice Semiconductor's ispLSI® (In-System Programmable™ Large Scale Integration) family, designed to implement high-integration functions, such as controllers, while delivering superior performance and the flexibility of In-System Programmability™ (ISP™). The basic PCI-compliant Master/Target state machines can be implemented in the ispLSI device, while the remaining glue logic can be modeled around a given peripheral/processor. The options become enormous, when one has the ability to change the functionality of devices already soldered on the board. ISP continues to emerge as the design methodology of choice by providing reconfigurable systems with diagnostic capabilities, field upgradability and simplification of manufacturing flow.

PCI flexibility brings with it new design challenges for the system designer. This application note presents a Master/Target-PCI interface design implemented in an ispLSI device. The attached source code contains the basic PCI- compliant state machines and is intended to be used as a guideline on which a PCI bridge design for a specific interface can be based. The benefits of ispLSI as applied to the PCI bus, and AC/DC and timing specifications are reviewed.

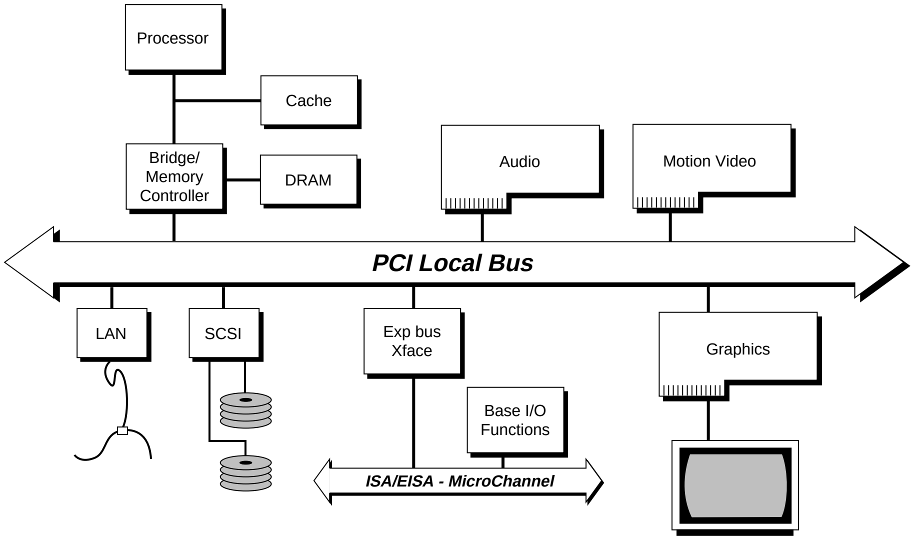
PCI/Lattice ispLSI Interface

The following section presents the PCI interface based on the PCI Local Bus Specifications, Revision 2.1. A concise overview of the PCI bus and ispLSI architecture and the relevant electrical and timing characteristics are discussed. The Lattice Semiconductor Data Book and the PCI Specification should be consulted to obtain more detailed information.

PCI Overview

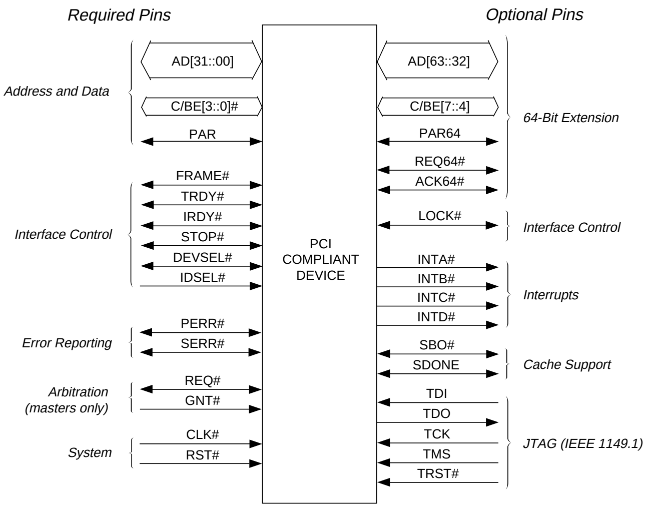
The PCI bus is a non-proprietary local bus solution, providing increased performance for Graphical User In-

Figure 1. PCI System Block Diagram



terfaces and other high bandwidth functions such as SCSI, full motion video, LANs etc.. The PCI component and the add-in card interface is processor independent, enabling an efficient transition to future processor generations and use with multiple processor architectures. Processor independence allows the bus to be optimized for I/O functions, enabling concurrent operation of the bus with the processor/memory subsystem. Figure 1 shows a typical PCI system.

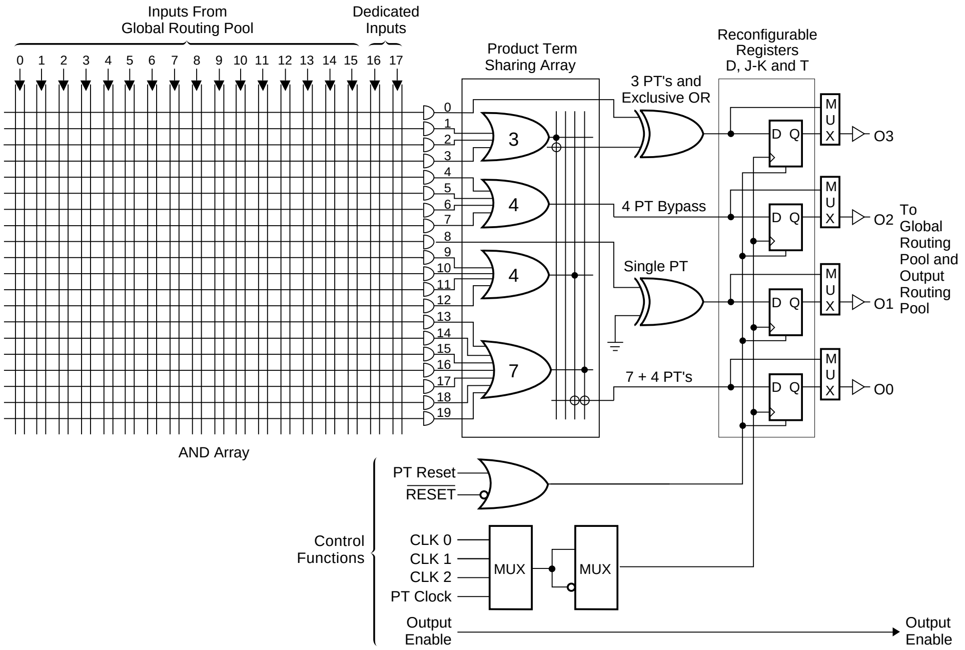
Figure 2. PCI Pin List



The processor/memory subsystem is connected to PCI through a bridge, which provides a low latency path for the agent to directly access the PCI devices mapped onto the processor address space. The PCI specifications defines both a Master and Target bridge implementation. Both can be implemented in one device, however each has to have an independent controller state machine. Figure 2 shows the pins required on a PCI controller in order to handle addressing, arbitration, interface control and other system functions. A minimum of 47 pins are needed for a Target only device and 49 pins for a Master.

The PCI interface consists of two different types of buses and control signals which govern the timing of data transfer on the address/data bus by the insertion of wait states. The larger of the two buses is the multiplexed Address/Data (AD) bus. The transfer of data onto the AD bus is not required to be the full width of the bus. The width of the data transfer is indicated by control informations present at the time of the bus transaction. The second bus is the Command/Byte Enable (C/BE) bus. The C/BE bus contains information about the activity that is to occur (i.e. read/write and memory or I/O access) during the address phase of the bus transaction, and contains the byte enables during the data phase of the bus transaction. Byte lane swapping is not allowed on the PCI bus since all devices must connect to 32 address/data bits. Furthermore, automatic bus sizing is not supported and the byte enables determine which bytes carry meaningful data. The PCI bus interface requires that every active member connected to the PCI bus be

Figure 3. Mixed Mode Generic Logic Block



synchronized to a system clock. This allows information to be transferred between the active agents with wait states, inserted by Master or Target, to match the timing requirements of either party involved in bus activity. The wait states are inserted through the use of the signals IRDY and TRDY. The signal FRAME indicates that a Master is currently active on the bus and that all other bus Masters are not to become active on the bus until the current activity is completed.

Lattice ispLSI Architectural Overview

Lattice's ispLSI high-density PLDs are ideally suited to high-speed controller, state machine intensive applications. This section provides a broad overview of the architecture. Relevant features will be discussed in further detail as they relate to this application note. In addition to in-system reprogrammability, characteristics such as wide input gating (18 input/20 product terms per register), hardware XOR gates on each register, low skew (less than 2 ns), input clamping capability and high speed make the ispLSI device ideal for complex state machine implementation. The ispLSI devices contain programmable logic, registers, I/O pins, multiple clocks, a Global Routing Pool and Output Routing Pool. The basic unit of logic is the Generic Logic Block (GLB). Figure 3 shows a simplified logic diagram of the ispLSI GLB.

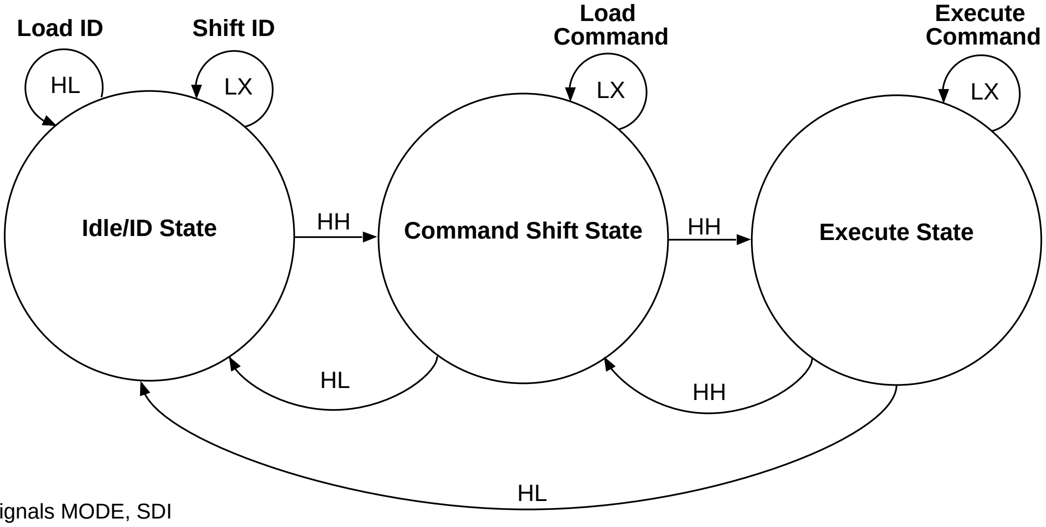
The Lattice ispLSI devices are programmable, in-circuit, on a powered board. This simplifies the design flow by eliminating the time consuming simulation process. The design can be tested in the final system by downloading the JEDEC file directly into the part. This is especially useful in surface mount environments where the parts cannot be removed from the board for programming. Test points are brought out to unused I/O pins during the debug cycle, and eliminated for standard operation. A designer can complete the design in steps by creating smaller modules of the design, testing them as stand-alone circuits, and then combining them once they are all working correctly. In addition to being a design tool, in-system programming also offers production advantages. Field service upgrades can be performed by simply reprogramming the boards, and options added by programming them into the logic. If several boards are similar in function, but have different logic, a single printed circuit board can be designed, and the specific function programmed into the logic just before the board is shipped. This reduces both production and inventory costs.

The only requirements of the system are that it must have a stable 5-volt power supply and a connection point for the ispDOWNLOAD™ cable. The standard interface used on the ispLSI prototype boards is a common eight-pin telephone connector. This connector is selected because it is small, reliable and inexpensive. Five pins on the ispLSI 1032 device are dedicated to programming when the part is used in the ISP mode. They are:

$\overline{\text{ispEN}}$	In-System Programming Enable
MODE	ISP Mode Control
SCLK	Shift Clock
SDI	Serial Data In
SDO	Serial Data Out

The algorithm used to program the part is straightforward. The MODE, SCLK and SDI pins are used to control a state machine internal to the ispLSI device. The device

Figure 4. ISP Programming State Machine



NOTE: Control Signals MODE, SDI

is controlled by serially shifting in a series of commands and data streams. The state diagram for that operation is shown in Figure 4.

PCI Electrical Specifications

The PCI specification provides for both 5V and 3.3V signaling environments, but all components in a PCI design must use the same signaling environment. The PCI bus is a CMOS bus, i.e., steady state currents are minimal (after transients have died out), with most of the current spent on pull-up resistors. PCI is based on reflective wave signaling, rather than incident wave, which implies that the bus drivers have to switch the bus halfway to the required high or low voltage. The fact that the bus is unterminated, causes the reflected wave at the unterminated end of the transmission line to add to the incident wave to achieve the required voltage level. (See Figure 5). The bus driver is actually in the middle of its switching range during this propagation time, which lasts up to 10ns, or one-third the bus cycle frequency of 33MHz. The PCI bus drivers are specified in terms of the AC switching characteristics or V/I curves. Figure 6 shows the V/I curves of the PCI bus under a 5V signaling environment.

The PCI specification dictates that pins used for extended data path (64-bit) such as high order AD lines, C/BE lines and PAR64 (64-bit extension parity, see Figure 2) have pullups in order to prevent oscillation or high power drain through the input buffer. Some signals have to be pulled up in order to have stable values when no agent is driving the bus. In addition, the inputs are required to be clamped to ground. According to the PCI Local Bus specification, clamps to 5V are optional, but may be needed to protect 3.3V devices. When using dual power rails, parasitic diodes exist from one supply to another. These diode paths can become forward biased, if one of the power rails goes out of specification for an instant. The diode clamps to the power rail and to the output devices must be able to withstand short circuit current until the drivers can be tristated.

It should be noted that PCI-compliant devices that directly drive the bus have extremely high output drive capability (greater than 48mA). This high drive is required to overcome incident wave effects that may occur within the design and not so much from a DC drive perspective. Hence, the ispLSI devices may be used in conjuction with external buffers (GAL®16VP8 or GAL20VP8) or with

Figure 5. Measurement of Tprop (From PCI Specification)

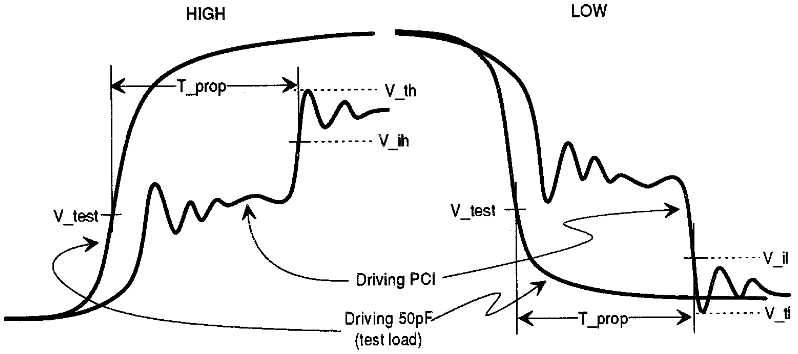


Figure 6. V/I Curve for 5V Signaling (From PCI Specification)

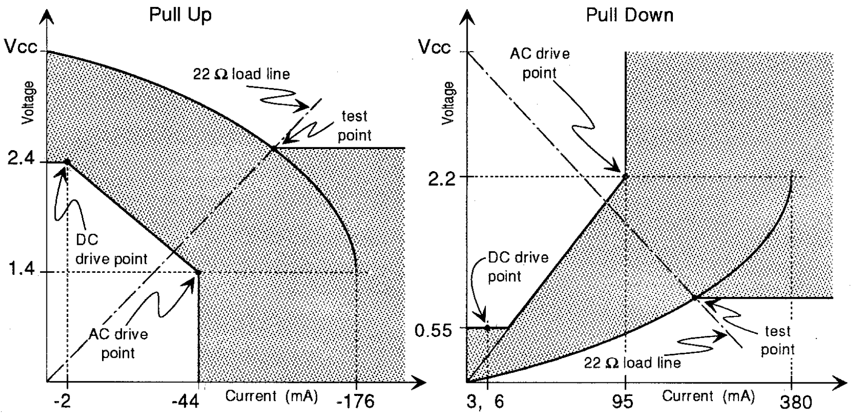


Figure 7a. ispLSI Input Clamp Characteristics

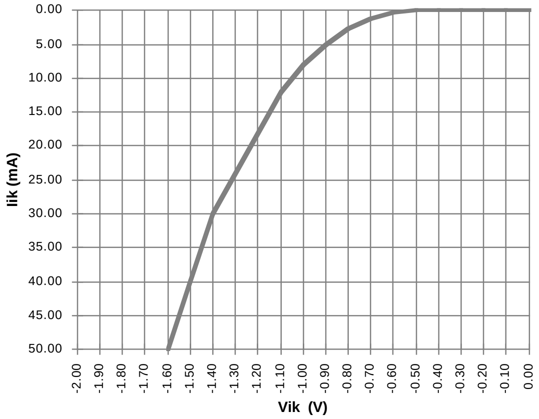


Figure 7b. ispLSI Vol vs. Iol

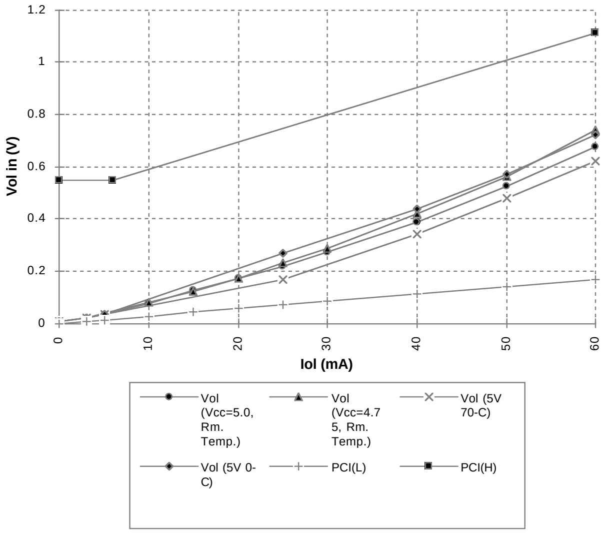
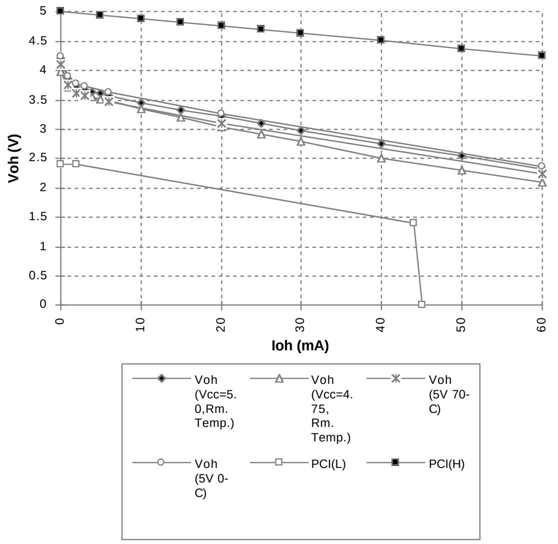


Figure 7c. ispLSI Voh vs. Ioh



series termination applied. In many cases, the loading conditions are such that no external buffering or termination is needed. This must be determined by the system designer.

Lattice ispLSI Electrical Specifications

The Lattice ispLSI families have programmable pull-up resistors that may be used instead of the external resistors, saving real estate. The ispLSI devices have an input clamp that turns on at approximately -1.7v, -18mA (see Figure 7). These clamps exist on each of the dedicated inputs and I/Os. In addition, the ispLSI devices are capable of operating under conditions of "excessive" overshoot or undershoot. Figure 8 depicts the results when a 16-volt peak-to-peak pulse is injected into the input or I/O pin.

Finally, with respect to input capacitance, the PCI specification stipulates that the input capacitance should not exceed 10 pF for an input pin and 12 pF for the clock and I/O pin. The ispLSI devices have input capacitance of eight pF on input pins and 10 pF on I/O and clock pins.

PCI Timing Requirements

The PCI specification provides strict timing requirements in terms of setup time (7ns minimum). The Lattice ispLSI 1032-80 device has a minimum set up time of 7ns on the inputs.

Please refer to the PCI specifications and the Lattice Semiconductor Data Book for detailed specifications of the PCI bus and Lattice ispLSI devices.

Controller Logic Implementation

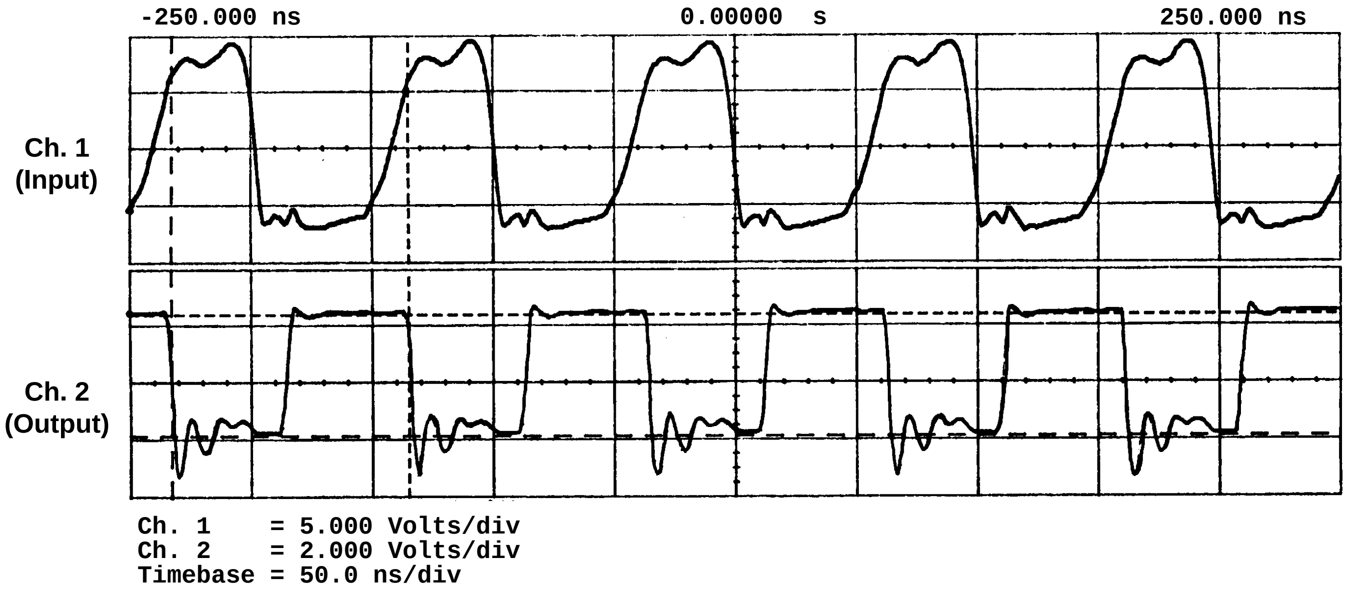
This section describes the implementation of the Master and Target state machines. Simulation waveforms are provided for the read cycle in Appendix A. The equations are for illustrative purposes only, and may have to be modified to support the actual design requirements. Lattice is not responsible for conflicts between the design and the specification. The PCI protocol has priority if any conflict arises in the equations.

Master State Machine

The PCI Master performs the following functions:

1. Data reads and writes on the PCI bus along with address stepping
2. Initiate a time-out if cycle is not decoded by any target (no subtractive decoding)
3. Initiate a PCI bus latency time-out
4. Responds to the system reset
5. Generate parity error

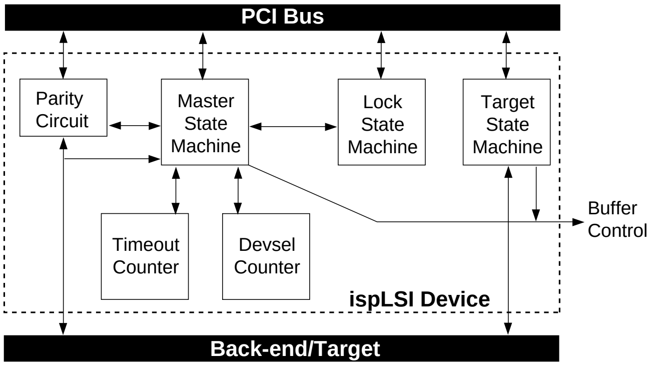
Figure 8. ispLSI Overshoot/Undershoot Characteristics



6. Can address memory or I/O space
7. PCI bus locked cycles

The Master state machine supports several options as specified in the PCI protocol. The bus interface consists of two parts. First, the Master Sequencer state machine which actually performs the bus operation. The second part, the backend (processor), initiates the transaction and provides the address, data, command, byte enables and the length of the transfer. It is responsible for the address if the transaction is retried. The backend can request a locked transfer or terminate a transfer. Each state of the sequencer machine will be discussed, with viable options. There are seven valid states of the sequencer machine:

Figure 9. Controller Block Diagram



IDLE is when the Master waits for a request from the backend to do a bus operation. The only possible option in this state is the ‘step’ option. This is extremely useful in stepping through a bus cycle, in the initial prototype stages of the product cycle. It can be removed if address stepping is not desired.

ADDR state is reached when the transaction is initiated by the processor. It is used to drive the address onto the bus, in this implementation it enables the address buffers and drives the commands on the bus.

DATA state is reached unconditionally from the ADDR state and the data is transferred in this state.

DATA1 state is reached from the DATA state only if more than one data phase is needed. This state is needed for the parity generation. The parity for the address lines needs to be generated in the clock after the address phase. Similarly, the data parity is generated in the next clock.

TURN_AR is where the Master deserts signals in preparation of tri-stating them. If back-to-back transitions are not required the path to the ADDR state may be removed. A turnaround cycle is required on all signals that may be driven by more than one agent in order to avoid contention when one agent stops driving the bus and another starts driving it.

S_TAR is reached when the current Target requests the Master to stop the transaction.

DR_BUS is used when the PCI bus has been granted to the current Master and the Master either is not prepared to start a transaction (for address stepping) or has none pending.

The following is the state diagram for the Master sequencer state machine. The transitions to various states will be discussed in greater detail following the state machine.

The attached equations (Appendix A) listing should be used as reference along with the Master Sequencer State Machine diagram in order to interpret the following state machine logic description.

The machine is in IDLE state when there are no requests for a bus transfer. On a processor PCI transaction request (generated by decoder, included in design), and the PCI bus grant from the external arbiter, the state machine transitions to the ADDR state. The PCI specification requires that there be only one central arbiter in the PCI system. This design assumes that the arbiter is implemented off board. If the processor is using address stepping, then the transition is to the DR_BUS state from the IDLE state.

Once in ADDR state, on the next clock the DATA state is reached unconditionally. In the ADDR state the appropriate command bus signals are driven. These define the PCI bus command, for example, 0010 specifies an I/O read cycle. These are generated from the processor read/write, IO/memory and data/code signals, which are used by the i486 to define the processor cycle. FRAME, which signals the start of a PCI cycle, is generated in the ADDR state and is held active through the DATA state till the Target/processor asserts a cycle complete signal.

In the DATA state, data is transferred from the Master to Target in case of a write, or from the Target to Master in case of a read. Wait states can be added by the Target by asserting TRDY or by Master by deasserting IRDY. In case of a read cycle, a turnaround cycle is required between the ADDR and DATA phases in order to avoid contention when one agent stops driving the signal and another agent starts driving. The turnaround wait state is asserted by the Target. (See PCI read cycle timing diagram, Figure 11 and Appendix A.). The DATA and DATA1 state are identical, the DATA1 state is needed for parity purposes.

In case of fast back-to-back processor cycles, the machine remains in the DATA1 state. A flag SA is used to determine if the current PCI cycle is going to the same Target as the previous cycle. Flag L_CYC is set when the current cycle is a write and the previous cycle was also a write. These flags determine the presence of fast back to back cycles. The state machine transitions to TURN_AR state if the cycles are not back-to-back, in preparation for completing the cycle and tri-stating the bus signals. If the Target asserts a STOP (stop current cycle), the machine transitions from the DATA1 state to the S_TAR state

The DR_BUS state is needed only if address stepping is used. In this design, transitions to this state are used for the Master to park on the PCI bus, while the processor is stepping though a cycle.

Figure 10. Master Sequencer State Machine

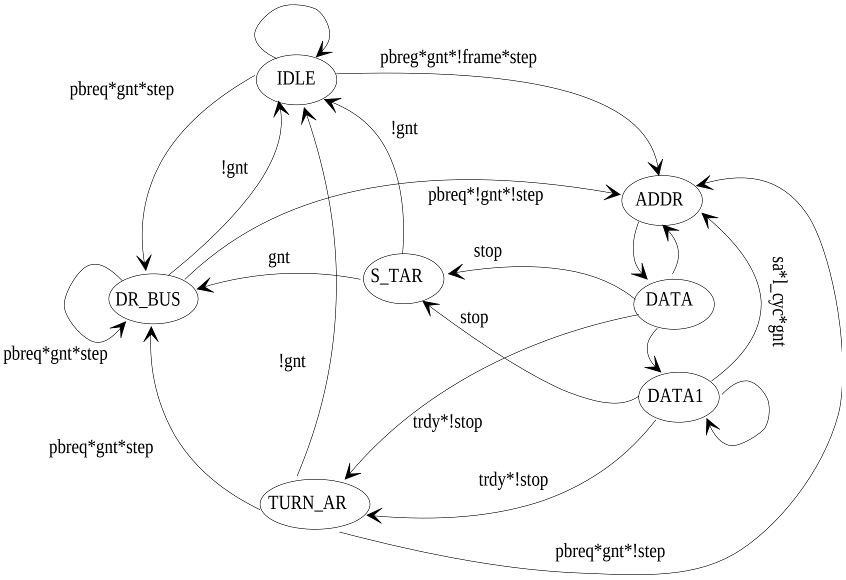


Figure 11. PCI Bus Read Cycle (From PCI Spec.)

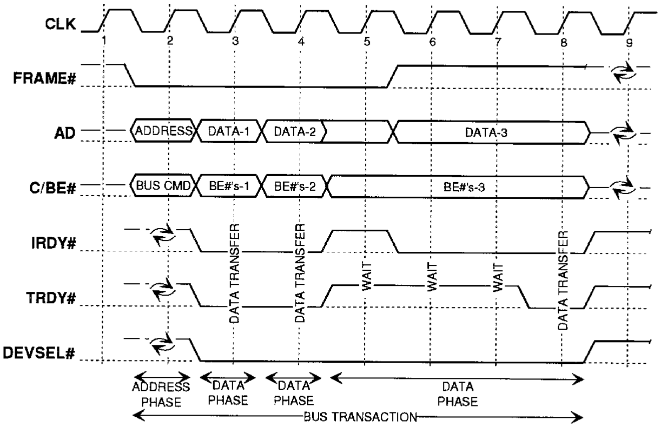
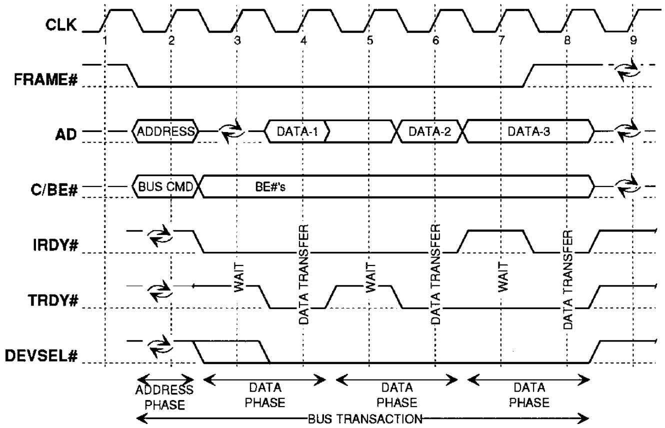


Figure 12. PCI Bus Write Cycle (From PCI Spec.)

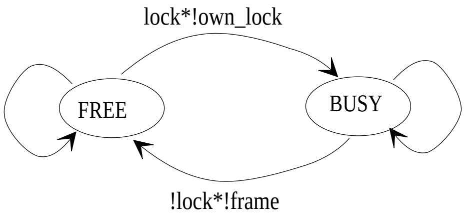


Finally, in the S_TAR state, if grant is valid, the machine transitions to DR_BUS state.

PCI provides an access mechanism which allows non-exclusive access to processors in the face of an exclusive access. This is referred to resource lock. This mechanism is based on locking only the PCI resource to which the original locked access was Targeted. The LOCK signal indicates that an exclusive lock is underway. The Master state machine controls the master lock mechanism. It has only two states, BUSY and FREE. The FREE state implies that the bus is not locked by any Master or the current Master has it locked. If another Master owns the lock, the state transitions to BUSY and stays there till LOCK and FRAME are deasserted. The LOCK state machine has not been simulated, since resource locks are not implemented in the on board Target, however, the equations are as per the PCI protocol.

The Devsel state machine is used to control the time-out. DEVSEL is driven by the Target of the current transaction. DEVSEL must be driven within three clocks following the address phase. That is, a Target must issue a DEVSEL before any response. If there is no subtractive decoding in the system, then the Devsel state machine will reach state SIX and time-out will be generated, signifying that no Target decoded the address. This will enable the Master to terminate the transaction.

Figure 13. Master Lock State Machine



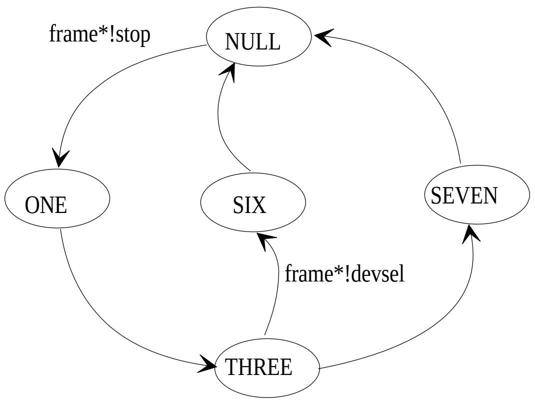
In addition to the above state machines, the Master Controller has a five-bit counter, which runs on a 1 MHz clock. This counter is used to generate the MAS_TO signal. This provides a 32 microseconds latency for all PCI transactions. Latency is defined as the time from when FRAME is asserted to TRDY being asserted. Typical latencies are relatively short, however worst case latencies may be quite long and unpredictable. For example, latency to a standard expansion adapter (ISA/EISA) through a bridge is often a function of the adapter behavior, not PCI behavior. The length of the latency time-out can be modified In-System as desired for a low latency system.

Target State Machine

The target located on the PCI bus performs the following functions:

1. Decodes the PCI bus cycle and provide data during a read

Figure 14. Devsel State Machine



2. Generates parity on PCI bus
3. Generates target abort to terminate a bus cycle
4. Inserts wait state during a read cycle between address and data phase

The PCI specification requires that the Target state machine be independent of the Master state machine. The Target interface has a backend that is responsible for determining when a transaction is terminated. The location of the Target in the Master backend address space can be changed In-System. Furthermore, subtractive decoding can be introduced if desired. This will make sure that the DEVSEL time-out is never asserted. The backend can also implement a resource lock. In this design, resource locks are not included in the target and zero wait state address decoding is assumed. The protocol for the target is fairly simplistic. The Master asserts the address, on a read cycle, if the target has an address hit, it initiates its internal state machine and either supplies the data or asserts an abort signal. Following is the Target state machine state description:

TGT_IDLE: In this state the machine is waiting for a decode to the target, i.e., the on-board decoder sees a bus cycle directed to the target. The machine transitions to TGT_DATA on HIT. This path can be removed if the Target cannot do single cycle decodes. If STOP is asserted by the Target, the machine transitions to BACKOFF. The machine goes to state B_BUSY when it sees FRAME asserted on the bus, but the HIT signal is still invalid.

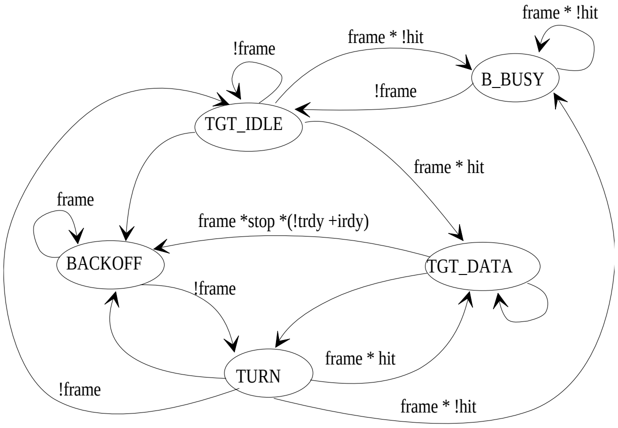
B_BUSY: The Target waits for the current transaction to complete and the bus to return to idle. This state is useful for devices that do slow address decode or perform subtractive decode. In this design , both these are not supported, hence there is no transition to the TGT_DATA and BACKOFF states.

TGT_DATA: The Target transfers data in this state. The machine transitions to BACKOFF if FRAME and STOP are asserted. In case of read cycle, the target asserts a wait state after the address is driven on the bus by the processor. This wait state is asserted by delaying the assertion of TRDY.

BACKOFF: The target goes to this state after it asserts STOP and waits for the Master to desert FRAME.

TURN: This state is reached when the transaction is completed. In preparation for the bus signals to be tri-stated.

Figure 15. Target State Machine



In addition to the above state machine the Target also contains a trivial command bus state machine. This machine is responsible for storing the command bus information during the address phase of the bus cycle. This is required since the command bus carries the byte enables during the data phase and the cycle type information is lost.

Parity

PCI-compliant devices are required to implement parity control. PCI bus has two signals, PAR and PERR that driven by the Master or Target. PAR is used to drive an even parity, covering AD3..AD0 and C/BE3..C/BE0, during address and data phases. To ensure the correct bus operation is performed, the four command lines are included in the parity calculation. In this design, parity generation is supported. The i486 processor drives DP0..DP3 lines which contain the parity bits for the four bytes of the processor bus. These bits and the data/address lines are used to generate PAR. The Lattice ispLSI device has a hardware eight-input XOR that can be used for this purpose. The Master drives the PAR onto the PCI bus during a write cycle. The Target is responsible for driving the PERR signal during the write cycle, if it has a parity error. During a read cycle, the Master generates the PERR based on the PCHK signal provided by the i486 processor. The Master also generates the PAR signal based on the state of PAR which is asserted by the Target in the read cycle. The PAR signal is generated by the Target on a read cycle. This design does not incorporate this feature, however it can be implemented quite nicely in an additional ispLSI device, since all the AD lines and the local processor lines are needed for generating the PAR bit.

Design Options/Enhancements

The PCI bridge can be designed to include various options. Some of them are discussed here.

1. Cache Support

A system may have some cacheable memory located on the PCI bus. The PCI specification allows the bridge to implement a standard interface which supports a snooping cache coherency mechanism. The support for cache is optimized for simple, entry-level systems and assumes a flat address space. PCI provides support for both Write-through and Write-back caches. The ispLSI devices provide an efficient implementation of a programmable cache controller on account of their in In-System Programmability, which makes the design flexible to support various cache schemes.

2. 64-Bit Data Bus

PCI provides a 64-bit extension to the data bus for agents with a 64-bit data bus. This requires 39 additional pins: REQ64, ACK64, PAR64, C/BE4..C/BE7 and AD32..AD63. Basically the 64-bit bus works the same way as the 32-bit bus. In this design, the data lines are not driven by the ispLSI device, which actually drives the control signals to enable the external buffers. This would make the expansion to 64-bit mode simple. The internal logic can be modified to support the additional control signals. REQ64 is the pin used by the Master to request a 64 bit-transfer. This is an extremely attractive option for 64-bit processors such as Pentium. When implementing this option, one has to be careful since Double Word swapping is allowed on the 64-bit data bus.

3. 64-Bit Addressing

PCI supports addressing beyond 4GB by defining a mechanism to transfer a 64-bit address from the Master to Target. A 64-bit address can be provided in one clock if the 64-bit address/data bus is being used. The Dual Address Cycle mode can be used, for 32-bit systems where the address is transferred in two clocks. This option cannot support address stepping on account of the two clock address transfer.

4. Slow Decoding Targets/Subtractive Decoding

This design assumes that the target can decode the PCI bus address with no wait state. For slower Targets, additional transitions can be added into the Target state machine, namely, transition from B_BUSY state to TGT_DATA and BACKOFF can be added. In addition, the path from IDLE to TGT_DATA can be removed if the Target cannot do single cycle decoding. Additional logic will depend on the specific Target implementation.

Other design options would be to include interrupt generation or even implement the entire interrupt controller in the master interface for PCI as well as local interrupts. Target Resource lock is another viable option. In a resource lock, exclusivity of an access is guaranteed by the target of the access, not by excluding all other accesses. This allows future processors to hold a hardware lock across several accesses without interfering with non-exclusive accesses such as video.

Summary

With the popularity of the non-proprietary, high-performance and extremely flexible local bus, it is not surprising that designers are looking to programmable logic to meet the challenges offered by a PCI interface design. The Lattice In-System Programmable device family is ideally suited to such complex state machine-intensive applications. While the sample design in this application note is specific enough to cover the required PCI protocol, it is adaptable and can be molded around any given peripheral or processor. In fact, it can even be reconfigured in the system from one peripheral to another, as long as the hardware interface is not too rigid. Additional features can always be added either in more ispLSI devices or discrete logic on account of the modular layout of the design.

The source file for the design is included in the following pages. This design is implemented using ABEL 4.1.3 software with Lattice pDS+™ ABEL Fitter. pLSI® Property Statements provide the user direct control over hardware specific features of the ispLSI and pLSI devices. The simulations were carried out using Viewlogic ViewSim software. Alternatively, the design can be implemented quite easily using the Lattice pDS+ Development System.

References

1. PCI Local Bus Specification, Rev. 2.1.
2. Lattice Semiconductor Data Book, 1996.
3. Lattice PDS+ Software, User Manual.
4. ABEL Design Software, User Manual.

Design Equations and Simulation Waveform

```
module pci_master
title  'pci bus master interface for i486 cpu';

"NOTES:
"this design assumes that there is no cacheable memory located
"as a target on the pci bus
"This design is a guideline for implementing PCI bus bridge
"for a 486 cpu interface.  This design does not implement a 100%
"PCI compatible bridge, however, the basic state machine is
"implemented and provides a baseline to build a complete PCI
"master interface

"****************************************************************
"****************************************************************
"       plsi properties
"****************************************************************
"****************************************************************

plsi property 'timing_sim pci_mast';
plsi property 'strong_route extended';
plsi property 'try 4';
plsi property 'max_delay 1';

"****************************************************************
"****************************************************************
declarations
"****************************************************************
"****************************************************************

pci_master device  'isp1032j09';

"****************************************************************
"       inputs
"****************************************************************
"inputs for processor interface
pa0 pin;                             "processor address lines
pa1 pin;                             "processor address lines
pa2 pin;                             "processor address lines
pa3 pin;                             "processor address lines
pa4 pin;                             "processor address lines
pa5 pin;                             "processor address lines
pa6 pin;                             "processor address lines
pa7 pin;                             "processor address lines
pa8 pin;                             "processor address lines
pa9 pin;                             "processor address lines
pa10 pin;                            "processor address lines
pa11 pin;                            "processor address lines
pa12 pin;                            "processor address lines
pa13 pin;                            "processor address lines
pa14 pin;                            "processor address lines
pa15 pin;                            "processor address lines
pa16 pin;                            "processor address lines
pa17 pin;                            "processor address lines
pa18 pin;                            "processor address lines
pa19 pin;                            "processor address lines
pa20 pin;                            "processor address lines
pa21 pin;                            "processor address lines
pa22 pin;                            "processor address lines
pa23 pin;                            "processor address lines
pa24 pin;                            "processor address lines
pa25 pin;                            "processor address lines
pa26 pin;                            "processor address lines
```

```
pa27 pin;                            "processor address lines
pa28 pin;                            "processor address lines
pa29 pin;                            "processor address lines
pa30 pin;                            "processor address lines
pa31 pin;                            "processor address lines
pbe0 pin;                            "processor byte enables
pbe1 pin;                            "processor byte enables
pbe2 pin;                            "processor byte enables
pbe3 pin;                            "processor byte enables
!plock pin;                          "processor lock pin
!pdata pin;                          "processor C/D  pin
!piom pin;                           "processor IO/m pin
!pbreq pin;                          "processor bus request
!pread pin;                          "processor read/write
dp0,dp1,dp2,dp3 pin;    "processor parity pins

step pin;                            "stepping input for debugging
cclk pin;                            "clock - 1mhz
pclk pin;                             "clock for timeout counter

"master input pins from pci
!gnt pin;                            "from bus arbiter
!trdy pin;                           "from target
!stop pin;                           "from target
!devsel pin;                         "indicates tgt has been selected
ready pin;                           "indicates ready to transfer
comp pin;

par pin;    "bidirectional parity pin

"slave inputs
term pin;   "slave wants to terminate the bus cycle
tar_dly pin;

"****************************************************************
"       outputs
"****************************************************************
"master output pins and bi directionals

cbe0 pin;
cbe1 pin;
cbe2 pin;
cbe3 pin;
data_en pin;                        " enables the data buffers on the pCI bus

!frame pin;
!lock pin;
!req pin;
!irdy pin;
addr_en pin;
mas_abort pin;    "transaction aborted by master due to timeout

"The following is the  output enable for the external buffers
ad_oe pin;

"****************************************************************
"       nodes
"****************************************************************
mas_to node;            "internal timer has expired
pci node;         " cpu access is on pci bus from built in address decoder
dev_to node;            "devsel timeout on pci bus,ie, DEVSEL was not asserted
sa node ;         "last cycle to same tgt as current
L_cyc node;       "last cyc was a write, bit set in register
Own_lock node;          "master owns lock
```

```
tgt_abort node;         "target aborts access
tgt1 node;
tgt1r node istype 'buffer,reg_d';   "used to store tgt access info.
ldt pin;
preadr node istype 'buffer,reg_d';    "used to store write/read cycle bit

"target related nodes
hit node;
cmdr3,cmdr2,cmdr1,cmdr0  node;

"****************************************************************
"       Other Definitions                                       *
"****************************************************************
"defn. of all bus cycles
int_ack                 = [0,0,0,0];
spec_cyc                =[0,0,0,1];
io_read           = [0,0,1,0];
io_write                = [0,0,1,1];
res1        = [0,1,0,0];    "RESERVED
res2        = [0,1,0,1];    "RESERVED
mem_read          = [0,1,1,0];
mem_write         = [0,1,1,1];
res3        = [1,0,0,0];    "RESERVED
res4        = [1,0,0,1];    "RESERVED
config_read       = [1,0,1,0];
config_write            = [1,0,1,1];
mem_wr_mult       = [1,1,0,0];
dual_add_cyc            = [1,1,0,1];
"for 64 bit addressing only- not supported by this design
mem_read_line           = [1,1,1,0];
mem_wr_inval            = [1,1,1,1];

cmd  = [cbe3..cbe0];    " for convenient definition of cbeX lines used in ist phase
of bus cycle
cmdr = [cmdr3..cmdr0];        "storage for command bus
pbex = [pbe3..pbe0];    "processor byte enables

"***** MASTER MACHINE DEFN. ***************************
"master lock machine
lreg node;
lreg istype 'buffer,reg_d';

"state definitions for lock machine
free = 0;
busy = 1;

"devsel timer machine
d2,d1,d0 node;
dreg = [d2..d0];
dreg istype 'buffer,reg_d';

"state defn. for devsel state machine
null  =     [0,0,0] ;
one   =     [0,0,1] ;
three =     [0,1,1] ;
seven =     [1,1,1] ;
six   =     [1,1,0] ;

"master sequencer machine
s0,s1,s2 node;
sreg = [s2..s0];
sreg istype 'buffer,reg_d';

"state defn. for master sequencer machine
```

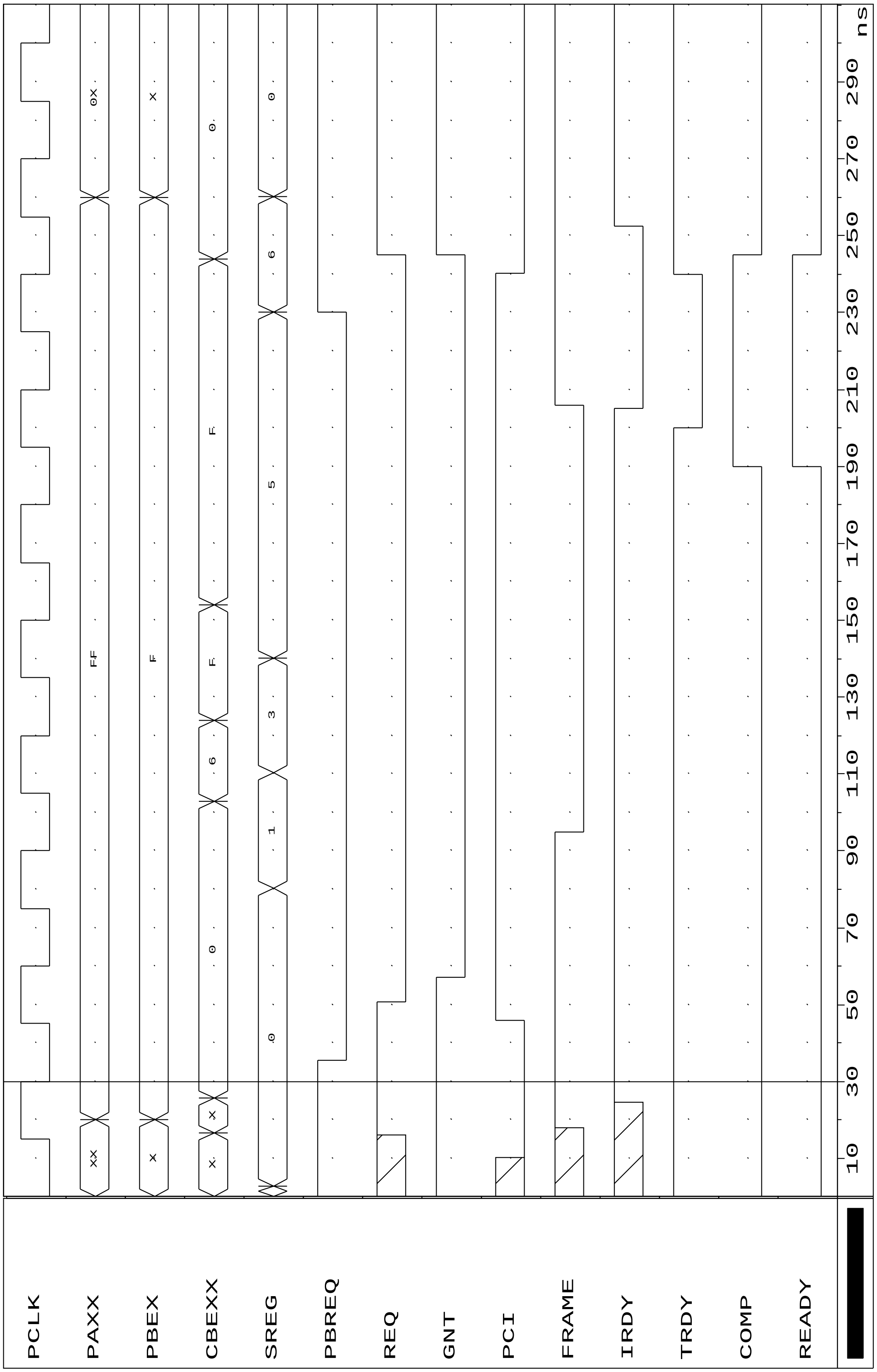
```
idle  =     [0,0,0] ;
addr  =     [0,0,1] ;
data  =     [0,1,1];
data1 =     [1,0,1];   " for parity purpose
dr_bus      =     [1,1,1];
turn_ar     =     [1,1,0];
s_tar           =       [1,0,0];

"counter defn.
q0,q4,q3,q2,q1 node;
count = [q4..q0];
count istype 'buffer,reg_d';

"****** TARGET MACHINE DEFN.************************************
t2,t1,t0 node;
treg = [t2..t0];
treg istype 'reg_d,buffer';

tgt_idle    = [0,0,0];
backoff     = [0,0,1];
b_busy   = [0,1,0];
tgt_data = [0,1,1];
turn  = [1,0,1];

"state machine to clock comand bus for target
c1,c0 node;
creg = [c1,c0];
creg istype 'reg_d, buffer';

no_ack      =     [0,0];
strobe      =     [0,1];
hold  =     [1,1];

"****************************************************************
"       State machines
*
"****************************************************************
"state diagram for sequencer machine
state_diagram sreg;

state idle:                          "idle state on the bus
if (pbreq & gnt & !frame & !irdy & !step)       "cpu has a pci bus request and ad-
dress strobe
     then       addr;

   else if ((!pbreq & gnt) # (pbreq & gnt & step)) & (!frame & !irdy)      "park on
bus if stepping
         then        dr_bus;

      else idle;

state addr:                          "master starts a transaction
     goto data;                            "goto data state on next clock

state data:                    "master transfers data  first data phase
if (frame) # ((!frame & !irdy & !trdy.pin & !stop.pin & !dev_to) & !((cmd==spec_cyc)
& comp))
   then   data1;

   else if (!frame & !step & trdy.pin & !stop.pin & !(cmd==spec_cyc) & sa & L_cyc &
pbreq & gnt)
        then addr;            " only if fast back to back cycles are supported

       else if (!frame & trdy.pin & !stop.pin & !(sa & L_cyc & pbreq & gnt) #
(!(cmd==spec_cyc) & comp))
                then turn_ar;       "turnaround state if no back-back cycles

                else if (!frame & stop.pin # !frame & dev_to & !trdy.pin)
                        then s_tar;

state data1:
if (frame) # ((!frame & !irdy & !trdy.pin & !stop & !dev_to) & !((cmd==spec_cyc) &
comp))
   then   data1;

    if (!frame & !step & trdy.pin & !stop.pin & !(cmd==spec_cyc) & sa & L_cyc &
pbreq & gnt)
        then addr;            " only if fast back to back cycles are supported

       else if (!frame & trdy.pin & !stop.pin & !(sa & L_cyc & pbreq & gnt) #
(!(cmd==spec_cyc) & comp))
                then turn_ar;       "turnaround state if no back-back cycles

                else if (!frame & stop.pin # !frame & dev_to & !trdy.pin)
                        then s_tar;

state turn_ar:                 " state for houskeeping purposes
if (pbreq & gnt & !step)
   then addr;

   else if (!pbreq & gnt # pbreq & gnt & step)
         then dr_bus;

         else if (!gnt)
              then idle;

          else turn_ar;

state s_tar:                               " turnaround state when stop is asserted
if (gnt)
   then dr_bus;

   else if (!gnt)
        then idle;

     else s_tar;

state dr_bus:                    "bus parked or address stepping is used
if (pbreq & gnt & step # !pbreq & gnt)
   then dr_bus;

   else if (pbreq & !gnt & !step)
        then addr;

        else if (!gnt)
             then idle;

          else dr_bus;

"************** end of master sequencer state machine****************************

"state diagram for LOCK machine
state_diagram lreg;

state free:                       "bus is locked by current master
if (!lock # lock & Own_lock)
   then free;
```

```
  else if (lock & !Own_lock)
       then busy;

state busy:                        " some other master has the bus locked
if (!lock & !frame)
   then free;

   else if (lock # frame)
        then busy;

"************** end of master lock state machine****************************


state_diagram dreg;

state null:                   "machine is waiting for frame to be asserted
if (frame & !stop)
   then one;
      else null;

state one:
   goto three;

state three:
   goto seven;

state seven:
if (!devsel & frame)
   then six;
   else null;

state six:
  goto null;
"************** end of devsel state machine**********************

" ************ target state machine *****************************
state_diagram treg;

state tgt_idle:               "target state machine is idle
if (!frame.pin)
   then tgt_idle;

   else if (frame.pin & !hit)
        then b_busy;

       else if (frame.pin & hit & (!term # term & ready))
             then tgt_data;

            else if (frame.pin & hit & term & !ready)
                then backoff;

                else tgt_idle;

state b_busy:
if ((frame.pin # irdy.pin) & !hit)
    then b_busy;

    else if (!frame.pin)
        then tgt_idle;

        else b_busy;

state tgt_data:
if (frame.pin & stop & trdy & !irdy.pin # frame.pin & !stop # !frame.pin & !trdy &
!stop)
    then tgt_data;

   else if (frame.pin & stop & (!trdy # irdy.pin))
     then backoff;

       else if (!frame.pin & (stop # trdy))
            then turn;

          else tgt_data;

state backoff:
if (frame.pin)
   then backoff;

   else if (!frame.pin)
      then turn;

state turn:
if (!frame.pin)
    then tgt_idle;

    else if (frame.pin & !hit)
      then b_busy;

       else if ( frame.pin & hit & (!term # term & ready))
        then  tgt_data;

         else if (frame.pin & hit & (term & !ready))
          then backoff;
"************* end of target state machine**********************

" cmd bus store state machine
state_diagram creg;

state no_ack:
if (!frame.pin )
     then no_ack;

    else if (frame & hit)
      then strobe;

state strobe:
goto hold;

state hold:
if (frame.pin)
    then hold;

     else if (!frame.pin)
         then no_ack;
"*****************************************************************
equations
"*****************************************************************
lreg.c = pclk;
dreg.c = pclk;
sreg.c = pclk;
treg.c = pclk;
creg.c=  pclk;

count.c = cclk;
count.re = trdy & gnt;               "reset counter when trdy is generated by master

"5 bit counter initiated by asserting frame,runs on a 1Mhz clock
```

```
"will generate mas_to signal at end of count
q0.d = q0 $ frame;
q1.d = (q0 & frame) $ q1;
q2.d = (q0 & q1 & frame) $ q2;
q3.d = (q0 & q1 & q2 & frame) $ q3;
q4.d = (q0 & q1 &  q3 & q3 & frame) $ q4;
mas_to = (q1 & q2 & q3 & q4 & q0 & frame );   "master timed out


pci = (pa31 & pa30 & pa29 & pa28 & pbreq);    " decoded pci address space f0000000-
ffffffff

Own_lock = !lock & !frame & !irdy & pbreq & gnt & plock # Own_lock & (frame # lock);

frame = (sreg==addr) # ((sreg==data)#(sreg==data1) & !dev_to & ((!comp & (!mas_to #
gnt) & !stop.pin) # !ready));

lock = !((Own_lock & (sreg==addr)) # tgt_abort # dev_to #
(((sreg==data)#(sreg==data1)) & stop.pin & !trdy.pin & !ldt)
        # (Own_lock & !plock & comp & ((sreg==data)#(sreg==data1)) & !frame &
trdy.pin));

req =  (pbreq & !plock # pbreq & plock & (lreg==free)) & !(sreg==s_tar);

irdy = ((sreg==data)#(sreg==data1)) & (ready # dev_to);

dev_to = (dreg==six);

mas_abort = mas_to;

cmd =    ( int_ack & ((sreg==addr) & pread  & piom  & pdata)
        # io_read & ((sreg==addr) & pread  & piom  & !pdata)
        # io_write &((sreg==addr) & !pread & piom  & !pdata)
        # mem_read & ((sreg==addr) & pread  & !piom & !pdata)
        # mem_write & ((sreg==addr) & !pread & !piom & !pdata)
        # spec_cyc & ((sreg==addr) & !pread & piom  & pdata)
        # pbex & (sreg==data)
        # pbex & (sreg==data1)
        # pbex & ((sreg==dr_bus) & step & pbreq));

addr_en = (sreg==addr);

data_en = (sreg==data)#(sreg==data1)#(sreg==dr_bus);

"preadr is used to store the write/read access
preadr.d = pread & gnt;

preadr.clk = pclk;

preadr.ar = (!gnt & !pci);

L_cyc = !pread & preadr.q;

tgt_abort = (stop.pin & !devsel.pin & ((sreg==data)#(sreg==data1)) & !frame & irdy);

"the following equations assume only 1 target device.  The access to the device
"is stored for back to back transfers. this can be expanded to include more devices

tgt1 = (pbreq & pa31 & pa30 & pa29 & pa28 & pa27 & pa26 & pa25 & pa24 ); "FF000000-
FFFFFFFF

tgt1r.d = tgt1;

tgt1r.ar = (!gnt & !pci);  "reset the register when there is a non-pci access
```

```
tgt1r.c = pclk;

sa = tgt1r.q & tgt1;

"****************************** output enables*****************************

cmd.oe = (sreg==addr) # (sreg==data) # (sreg==dr_bus) # (sreg==data1);

lock.oe = Own_lock & ((sreg==data)#(sreg==data1)) # (lock.oe & (frame # lock));

ad_oe = (sreg==addr) # (sreg==dr_bus) & step & pbreq;

"irdy needs to be asserted when addr or data are the previous states
irdy.oe = (sreg==addr)
          #((sreg==idle) & pbreq & gnt & !frame & !irdy & pci & !step)   "asserted
when addr is next state
          #((sreg==turn_ar) & pbreq & gnt & !step)           " asserted when addr is
next state
          #((sreg==data) & !frame & !step & trdy.pin & !stop.pin & !(cmd==spec_cyc)
& sa & L_cyc & pbreq & gnt)
          #((sreg==dr_bus) & pbreq & !gnt & !step)           "asserted when addr is
next state
          #(((sreg==data) & frame) # ((!frame & !irdy & !trdy.pin & !stop.pin &
!dev_to) & !((cmd==spec_cyc) & comp)))
          #(((sreg==data1) & frame) # ((!frame & !irdy & !trdy.pin & !stop.pin &
!dev_to) & !((cmd==spec_cyc) & comp)));


"****************************************************************************
"   Parity logic
"****************************************************************************
par = ((dp0 $ dp1) $ (dp2 $ dp3));
"        # (from slave par circuit);

par.oe = (sreg==data) & (cmd==io_write) # (cmd==mem_write)   "for address parity
        # (sreg==data1) & (cmd==io_write) # (cmd==mem_write)   "for data parity
        # (treg==tgt_data) & trdy & ((cmdr==io_read) # (cmdr==mem_read));  " for
slave driven par

"******************************** target equations***********************
trdy = !(ready & !tgt_abort & (treg==tgt_data)
        & (((cmdr==io_write) # (cmdr==mem_write))
           # ((cmdr==io_read) # (cmdr==mem_read) & tar_dly)));

stop = !((treg==backoff) # (treg==tgt_data) & (tgt_abort # term)
        & (((cmdr==io_write) # (cmdr==mem_write))
           # ((cmdr==io_read) # (cmdr==mem_read) & tar_dly)));

devsel = !((treg==backoff ) # (treg==tgt_data)  & !tgt_abort);

"perr = (from parity circuit)

trdy.oe =    (treg==backoff) # (treg==tgt_data) #  (treg==turn);

stop.oe =  (treg==backoff) # (treg==tgt_data) #  (treg==turn);

devsel.oe = (treg==backoff) # (treg==tgt_data) #  (treg==turn);

"hit = (decode of  PCI address lines );

cmdr = cmd & (creg==strobe);        "store the command bus info for use

END;
```

PCI Bus Read Cycle (Simulation)



pDS+ Fitter Report

```
********************************************
*                                          *
*        Lattice pDS+ Fitter Report        *
*                                          *
********************************************

Copyright (c) Lattice Semiconductor Corp. 1992.  All Rights Reserved.

      Design Name:  pci_mast
             File:  pci_mast.doc
        Date/Time:  Mon Apr 25 12:13:33 1994
  Targeted Device:  pLSI1032-90LJ84
 Software Version:  DPM 1.60  12/8/93

Fitter Parameters Used
----------------------

       AVG_GLB_IN:  16
           EFFORT:  4
 IGNORE_FIXED_PIN:  OFF
        MAX_DELAY:  1
       MAX_GLB_IN:  16
       PARAM_FILE:  (null)
             PART:  pLSI1032-90LJ84
       TIMING_SIM:  pci_mast
              TRY:  4
       FAST_ROUTE:  OFF
     STRONG_ROUTE:  EXTENDED

Process Status
--------------

     Design Analysis:  complete
  Logic Partitioning:  complete
     Place and Route:  complete
          Post Route:  complete
Fuse File Generation:  complete
Merging TMV in JEDEC:  incomplete


*********************************
*                               *
*       Post-Route Report       *
*                               *
*********************************

Design Name:            pci_mast
Targeted Device:  pLSI1032-90LJ84
Date/Time:        Mon Apr 25 13:09:06 1994

Software Version: 1.00.35

All strategy results:
Strategy 4 - Estimated No. of GLBs : 19
Strategy 4 - Estimated No. of GLB Levels: 3

Final Selected Strategy 4 - Estimated No. of GLBs : 19
Strategy 4 - Estimated No. of GLB Levels: 3

Partitioning:

Total number of GLBs : 31
Total number of Product Terms used : 193
Average number of Product Terms :   6.2
Total number of nets created : 119
Average number of Inputs per GLB :   9.6
Average number of Outputs per GLB :   2.2
Number of I/Os Generated : 46
Number of Dedicated Inputs Generated : 4
Type of Clocks Generated : 2 System Clocks
                         : 0 I/O Clocks
                         : 0 Product Term Clocks
```

Post Route Pin Report

```
Post-Route Pin Report
--------------------------

Pin Number     Signal Name         Fixed           Pin Type

    1              GND              Yes                Gnd
    3        mas_abort               No             Output
    5             pa29               No              Input
    6            frame               No             Output
    7            irdy-               No              Input
    9          devsel-               No              Input
   10             irdy               No               Bidi
   11              par               No             Output
   14             comp               No              Input
   16             pa28               No              Input
   20             pclk              Yes              Clock
   21              VCC              Yes                Vcc
   22              GND              Yes                Gnd
   25             pbe2               No              Input
   26              hit               No              Input
   27             cbe3               No               Bidi
   28             cbe2               No               Bidi
   29              req               No             Output
   30          addr_en               No             Output
   31             term               No              Input
   32          tar_dly               No              Input
   33           frame-               No              Input
   34             pa31               No              Input
   35           devsel               No               Bidi
   37             pa27               No              Input
   38              dp0               No              Input
   39          data_en               No             Output
   40             stop               No               Bidi
   41             pa26               No              Input
   42             pbe3               No              Input
   43              GND              Yes                Gnd
   44             pbe1               No              Input
   45             cbe0               No               Bidi
   46              gnt               No              Input
   47            pdata               No              Input
   48             cbe1               No               Bidi
   49            trdy-               No              Input
   50              dp3               No              Input
   51              dp2               No              Input
   52              dp1               No              Input
   54            pread               No              Input
   55            plock               No              Input
   56            ad_oe               No             Output
   64              GND              Yes                Gnd
   65              VCC              Yes                Vcc
   66             cclk              Yes              Clock
   68            pbreq               No              Input
   69             pbe0               No              Input
   70            ready               No              Input
   71             piom               No              Input
   72             pa24               No              Input
   77            #lock               No               Bidi
   78             step               No              Input
   79             pa30               No              Input
   81            stop-               No              Input
   82             pa25               No              Input
   83             trdy               No               Bidi
   84              ldt               No              Input
```

Lattice Semiconductor Corporation (LSC) products are made under one or more of the following U.S. and international patents: 4,761,768 US, 4,766,569 US, 4,833,646 US, 4,852,044 US, 4,855,954 US, 4,879,688 US, 4,887,239 US, 4,896,296 US, 5,130,574 US, 5,138,198 US, 5,162,679 US, 5,191,243 US, 5,204,556 US, 5,231,315 US, 5,231,316 US, 5,237,218 US, 5,245,226 US, 5,251,169 US, 5,272,666 US, 5,281,906 US, 5,295,095 US, 5,329,179 US, 5,331,590 US, 5,336,951 US, 5,353,246 US, 5,357,156 US, 5,359,573 US, 5,394,033 US, 5,394,037 US, 5,404,055 US, 5,418,390 US, 5,493,205 US, 0194091 EP, 0196771B1 EP, 0267271 EP, 0196771 UK, 0194091 GB, 0196771 WG, P3686070.0-08 WG. LSC does not represent that products described herein are free from patent infringement or from any third-party right.

The specifications and information herein are subject to change without notice. Lattice Semiconductor Corporation (LSC) reserves the right to discontinue any product or service without notice and assumes no obligation to correct any errors contained herein or to advise any user of this document of any correction if such be made. LSC recommends its customers obtain the latest version of the relevant information to establish, before ordering, that the information being relied upon is current.

LSC warrants performance of its products to current and applicable specifications in accordance with LSC’s standard warranty. Testing and other quality control procedures are performed to the extent LSC deems necessary. Specific testing of all parameters of each product is not necessarily performed, unless mandated by government requirements.

LSC assumes no liability for applications assistance, customer’s product design, software performance, or infringements of patents or services arising from the use of the products and services described herein.

LSC products are not authorized for use in life-support applications, devices or systems. Inclusion of LSC products in such applications is prohibited.

LATTICE SEMICONDUCTOR CORPORATION
5555 Northeast Moore Court
Hillsboro, Oregon 97124 U.S.A.
Tel.: (503) 681-0118
FAX: (503) 681-3037
http://www.latticesemi.com

November 1996