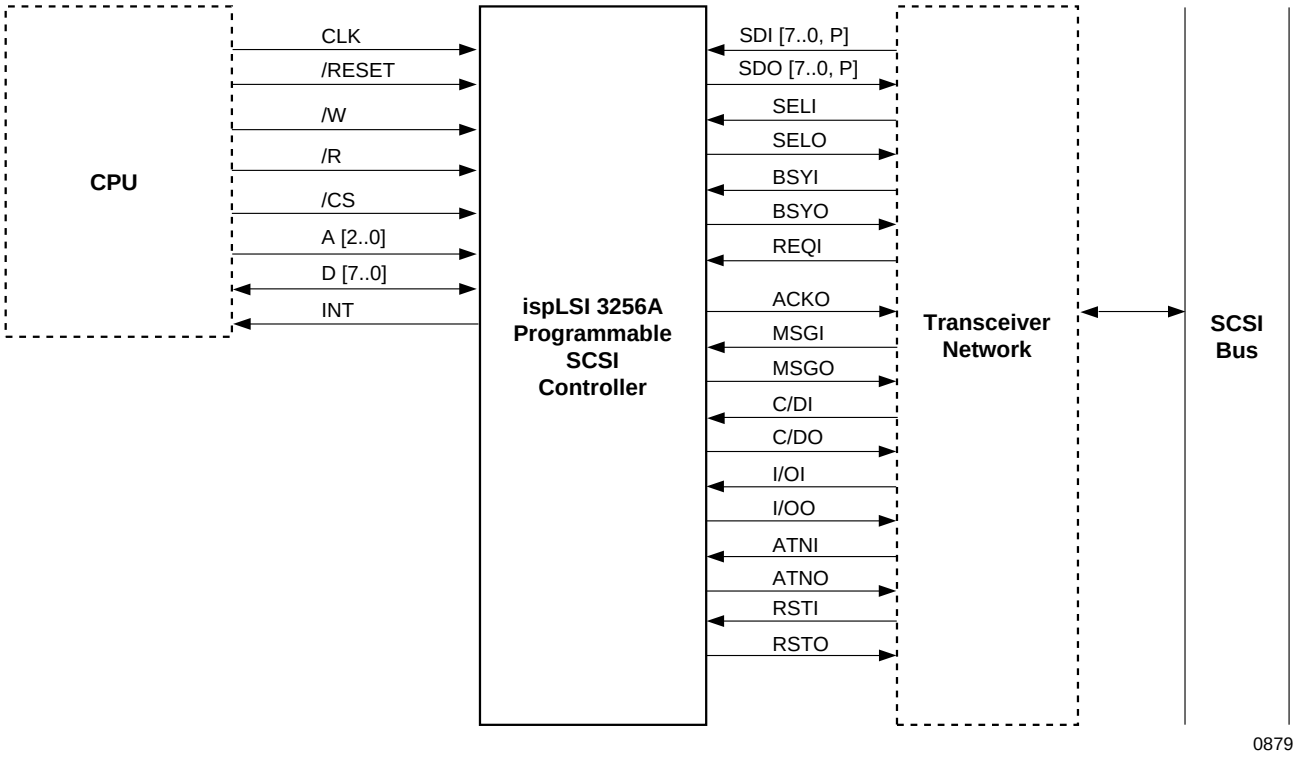
SCSI Interface with the ispLSI® 3256A

Introduction

Today's high-performance computer systems require greater data storage capacity and higher throughput. The SCSI (Small Computer System Interface) bus interface has risen to become the standard in peripheral communications for high-end computer systems. The versatility and flexibility of SCSI allow for higher integration without sacrificing cost and space. This applications note describes the implementation of a Programmable SCSI Controller (PSC) using Lattice Semiconductor's ispLSI 3256A device. Figure 1 shows a block diagram of a PSC application.

SCSI is an intelligent bus interface that provides high-performance data transfers between the host computer and peripheral devices. SCSI allows a maximum of eight devices to be attached to the bus without additional hardware. Control of the SCSI bus is shared through arbitration using a prioritized ID assigned to each SCSI device. When two SCSI devices communicate, one acts as an initiator, and the other as a target. The initiator originates an operation, and the target executes the operation. The ispLSI 3256A PSC design implements a SCSI initiator.

Figure 1. Application Using an ispLSI 3256A Device as Programmable SCSI Controller



SCSI Bus Phases

Communication between the devices is governed by the SCSI bus phases. Figure 2 shows a simple state flow for the different SCSI bus phases. Initial system power-up and all SCSI reset conditions puts the SCSI bus in the Bus Free state. Although optional, almost all SCSI systems support and use the arbitration facility to prevent bus contention. Either the Selection or Reselection phases follow after winning arbitration. The Information Transfer state is really composed of the Command, Data, Status and Message phases. These phases determine the type of data on the bus and in what direction the data travel.

BUS FREE - The Bus Free phase indicates that the SCSI bus is available for use and that no SCSI device is actively using it. SCSI operations normally start and end with the Bus Free phase.

ARBITRATION - The Arbitration phase allows an SCSI device to acquire control of the SCSI bus. Depending on the control signals the device will become either the initiator or target.

SELECTION - During the Selection phase, an initiator selects a target to begin an operation such as a READ or WRITE.

RESELECTION - During the Reselection phase, a target reconnects to an initiator after operation was suspended by the target.

COMMAND - The Command phase allows the target to request instructions from the initiator.

DATA IN - During the Data In phase, data is transferred from the target to the initiator.

DATA OUT - During the Data Out phase, data is transferred from the initiator to the target.

STATUS - The Status phase allows the target to pass status information to the initiator.

MESSAGE IN - During the Message In phase, the target sends message(s) to the initiator.

MESSAGE OUT - During the Message Out phase, the initiator sends message(s) to the target.

SCSI Bus Signals

The SCSI bus phases are determined by the configuration of the control signals.

BSY (BUSY) Signal indicating the SCSI bus is being used by a device.

SEL (SELECT) Signal driven by an initiator to select a target or by a target to reselect an initiator.

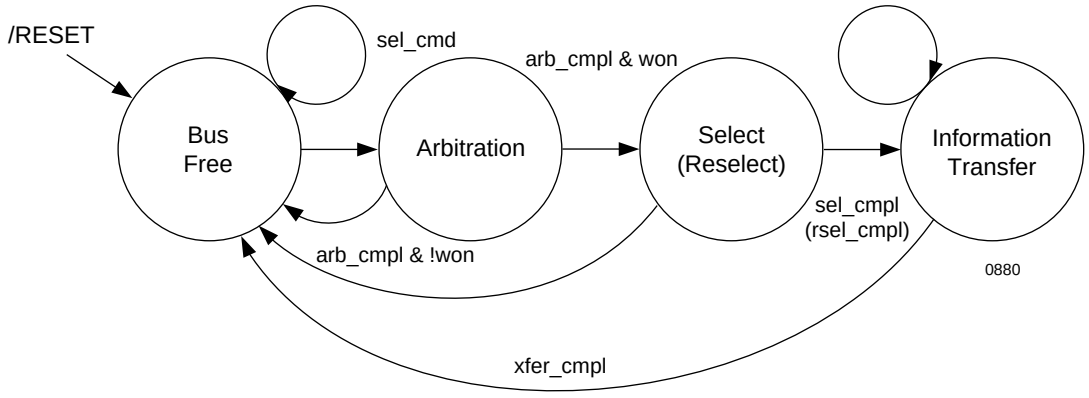
C/D (CONTROL/DATA) Signal driven by a target that indicates the direction of data transfer on the data bus. True signal indicates data flow from the target to the initiator.

MSG (MESSAGE) Signal used by a target during the MESSAGE phase.

REQ (REQUEST) Signal driven by a target to request for data transfer.

ACK (ACKNOWLEDGE) Signal driven by an initiator to acknowledge a data transfer.

Figure 2. SCSI Bus Phase Sequences



ATN (ATTENTION) Signal driven by an initiator to indicate the ATTENTION condition.

RST (RESET) Signal that indicates the RESET condition.

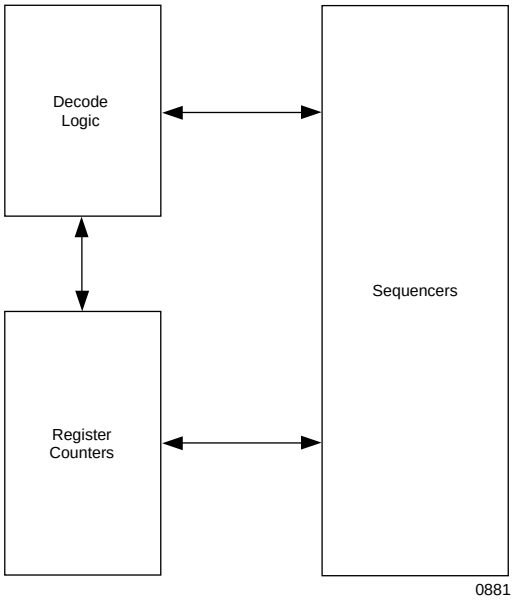
DB(7-0,P) (DATA BUS) Data bus signals that include eight data-bits and a parity-bit. Data parity is odd.

Design Description

A complex programmable logic device such as the ispLSI 3256A is an ideal solution for a Programmable SCSI Controller (PSC) device where flexibility is attained without sacrificing speed or density. The 11,000-PLD gate ispLSI 3256A with 80 MHz operating frequency and 15ns delay, provides optimal performance for this application. In addition, the ispLSI 3256A provides not only In-System Programmability but also reconfigurability without the need to remove the device from the PCB. The following discussion shows an ispLSI 3256A device implementing the role of an initiator with support for arbitration, selection, and reselection sequences.

Figure 3 shows the functional blocks of the ispLSI 3256A design which consists of three main modules: sequencers, decoding logic, registers and counters.

Figure 3. Block Diagram of Programmable SCSI Controller



The sequencers module consist of five state machines which process the SCSI bus data and control the flow of information. The RESEL_SM state machine handles the reselection phase sequences. The SEL_SM state machine processes all control signals and executes the selection of the target device. The ARB_SM state machine supports the arbitration phase. The DDXFER_SM state machine controls the transfer protocol between the initiator and the target. The SEQ_SM state machine is the main sequencer which oversees all other state machines.

The ispLSI 3256A architecture is ideal for building complex state machines. State transitions and conditional branching are supported by the AND-OR arrays and Product-Term Sharing Arrays (PTSA) logic. With up to 20 product-terms and hard-XOR gates, high-speed complex combinatorial logic can be realized. The PSC's state machines require a large number of inputs and many product-terms to implement. With 24 inputs per GLB, the ispLSI 3256A can maintain single delay levels for high fan-in functions.

The registers of the PSC include: CMD_REG, IN_REG, OUT_REG, STAT_REG and INTR_REG. The CMD_REG (address 0) is a write-only register used to store the commands for the PSC. The IN_REG, utilizing the input registers of the ispLSI 3256A, holds all the input signals from the SCSI bus. The OUT_REG stores data to be sent to the SCSI bus. The STAT_REG is a readable register giving status of the PSC and the SCSI operation.

Input Register (IN_REG)

Output Register (OUT_REG)

Command Register (CMD_REG)

SEL_CMD

Status Register (STAT_REG)

Interrupt Register (INTR_REG)

SCSI ID Register (SID_REG)

In addition to the 256 GLB registers, the ispLSI 3256A offers 128 registers/latches in the I/O cells. Besides implementing input latches (as used in the PSC design), the I/O registers/latches can also be used for signal synchronization, double registering for mestability, etc.

There are a number of counters in the PSC used to provide timing delays associated with the SCSI operations. The BSFR_DLY counter provides the necessary delay before arbitration may begin. The BSST_DLY

counter provides the bus settle delay before the PSC may be reselected by the target device. The ARB_DLY counter gives the arbitration delay timing.

Fast, loadable counters can be easily implemented in an ispLSI 3256A device. Wide GLB inputs allow up to 24 signals including counter load inputs and Q feedbacks to drive single logic level flip-flop equations without using additional logic.

The ispLSI 3256A also provides two Global Output Enables (GOEs) which are dedicated input pins driving all of the 127 I/O cells for output or directional operations. In the PSC design, the two GOEs can be used for transfering bidirectional data to the CPU and SCSI buses without requiring internal product-terms or routing resources.

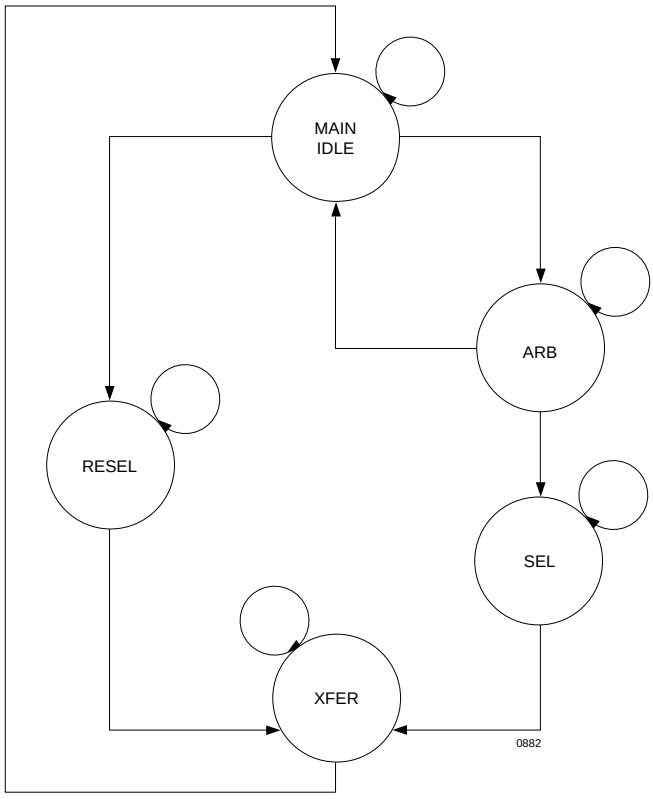
Typical SCSI Operations

A typical SCSI operation can be used to illustrate the functionality of the Programmable SCSI Controller. The first section describes sequences associated with the target selection by the PSC. The second section details the reselection of the PSC by the target device.

PSC Selects Target

INITIALIZATION - For the PSC, a typical SCSI operation starts with the SEQ_SM (main sequencer) state machine in the IDLE state and waits for a SEL command from the CPU. Figure 4 shows a state machine diagram for SEQ_SM. Once the SEL command is received, the SEQ_SM goes into the arbitrate state and remain there until arbitration is complete. Listing 1 details the ABEL implementation of the main sequencer.

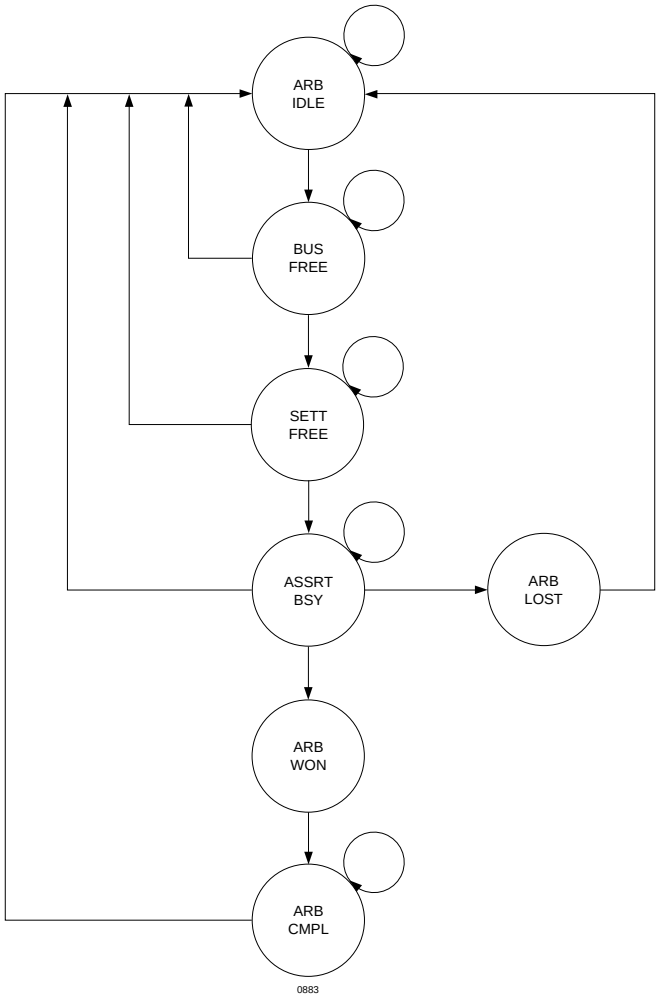
Figure 4. Main Sequencer State Machine Diagram (SEQ_SM)



Listing 1.

```
STATE_DIAGRAM main_sm

STATE main_idle:
  IF sel_cmd==1 THEN arb_st
“ Receive Select command “
    WITH arb_phs.d = 1
  ELSE IF (resel_phs) THEN resel_st
“ Detect Reselection Phase “
  ELSE main_idle;

STATE arb_st:
“ Arbitration Phase “
  IF (arb_cmpl & won) THEN sel_st
“ Won Arbitration “
    WITH sel_phs.d = 1
“  Goto Selection Phase “
  ELSE IF (arb_cmpl & !won) THEN
main_idle   “ Lost Arbitration “
    WITH intr.d = 1;
“  Interrupt CPU “

STATE sel_st:
“ Selection Phase “
  IF (sel_cmpl) THEN xfer_st

    WITH xfer_phs.d = 1
  ELSE sel_st;

STATE resel_st:
“ Reselection Phase “
  IF (resel_cmpl) THEN xfer_st
    WITH xfer_phs.d = 1
  ELSE resel_st;

STATE xfer_st:
“ Data Transfer Phase “
  IF (xfer_cmpl) THEN main_idle
  ELSE xfer_st;

END
```

Figure 5. Arbitration State Machine Diagram (ARB_SM)



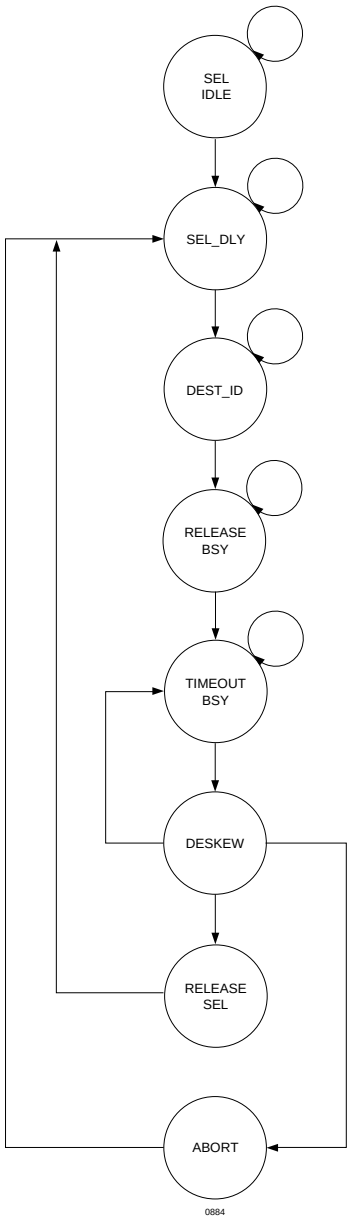
Listing 2.

```
STATE_DIAGRAM arb_sm

STATE arb_idle:
  IF (arb_phs) THEN bus_free
“ Detect start of Arbitration
  ELSE arb_idle;

STATE bus_free:
  IF (!bsyi & !seli) THEN sett_free
“ Detect Bus Free state
    WITH timer.ar = 1
  ELSE IF (arb_phs) THEN bus_free
  ELSE arb_idle;

STATE sett_free:
  IF (!bsyi & !seli & timer.q ==
```

```
sett_free_dly)    " Bus settle & bus free
delays
    THEN assrt_bsy WITH timer.ar = 1
  ELSE IF (!bsyi & !seli & timer.q <
^D11)
    THEN sett_free WITH timer.d =
timer.q + 1
  ELSE arb_idle;

STATE assrt_bsy:
  bsyo.j = 1;
" Assert BSY
  dbo0 = sid==0;                    "
Assert RSC's SCSI ID
  dbo1 = sid==1;
  dbo2 = sid==2;
  dbo3 = sid==3;
  dbo4 = sid==4;
  dbo5 = sid==5;
  dbo6 = sid==6;
  dbo7 = sid==7;
  IF (!seli & won & (timer.q==arb_dly))
      " Won Arbitration
    THEN arb_won WITH timer.ar = 1
  ELSE IF (seli # ((timer.q==arb_dly &
!won)))     " Failed Arbitration
    THEN arb_lost
  ELSE assrt_bsy WITH timer.d = timer.q
+ 1;  " Arbitration delay period

STATE arb_won:
  selo.j = 1;
" Assert SEL
  GOTO arb_cmpl;

STATE arb_cmpl:
  IF (timer.q==sett_clr_dly)
" Wait for bus settle & clear
    THEN arb_idle WITH arb_cmplt = 1
      " End of Arbitration
  ELSE arb_cmpl WITH timer.d = timer.q +
1;

STATE arb_lost:
  arb_cmplt = 1;
" End of Arbitration
  bsyo.k = 1;
" Negate BSY
  arb_fail_int = 1;
" Set Interrupt
  GOTO arb_idle;
END
```

BUS FREE PHASE - Before arbitration can begin, the PSC must detect the Bus Free phase. The ARB_SM state machine must read the BSY and SEL signals false for a bus free delay until the arbitration phase is entered. Figure 5 shows a state machine diagram for ARB_SM and Listing 2 details the ABEL implementation of the arbitration process.

Figure 6. Selection State Machine Diagram (SEL_SM)



Listing 3

```
STATE_DIAGRAM selection

STATE sel_idle_st:
  IF (sel_phs) THEN sel_dly_st
  ELSE sel_idle_st;

STATE sel_dly_st:
  IF (sel_dly_cnt == 12) THEN dest_id_st
  ELSE sel_dly_st
    WITH sel_dly_cnt.D = sel_dly_cnt.Q +
1;

STATE dest_id_st:
  dbo0 = (dest_id==0);
  dbo1 = (dest_id==1);
  dbo2 = (dest_id==2);
  dbo3 = (dest_id==3);
  dbo4 = (dest_id==4);
  dbo5 = (dest_id==5);
  dbo6 = (dest_id==6);
  dbo7 = (dest_id==7);
  IF (id_dly_cnt==4 & atn_cmd) THEN
release_bsy_st WITH atno=1
  ELSE IF (id_dly_cnt==4 & !atn_cmd)
THEN release_bsy_st
  ELSE dest_id_st WITH
id_dly_cnt.D=id_dly_cnt.Q+1;

STATE release_bsy_st:
  bsyo.K = 1;
  IF (bsy_dly_cnt==4) THEN
timeout_bsy_st
  ELSE release_bsy_st WITH bsy_dly_cnt.D
= bsy_dly_cnt.Q + 1;

STATE timeout_bsy_st:
  IF (bsyi & resel_cmd) THEN deskew_st
  ELSE if (!bsyi & timeout_cnt==timeout)
THEN abort_st
  ELSE timeout_bys_st WITH timeout.D =
timeout.Q + 1;

STATE deskew_st:
  GOTO deskew1_st;

STATE deskew1_st:
  GOTO deskew2_st;

STATE deskew2_st:
  GOTO release_sel_st;

STATE release_sel_st:
  selo.K = 1;
  sel_cmplt = 1;
  GOTO sel_idle_st;

STATE abort_st:
  selo.K = 1;
  disc_int.D = 1;
  cmd_reg.re = 1;
  GOTO sel_idle_st;

END
```

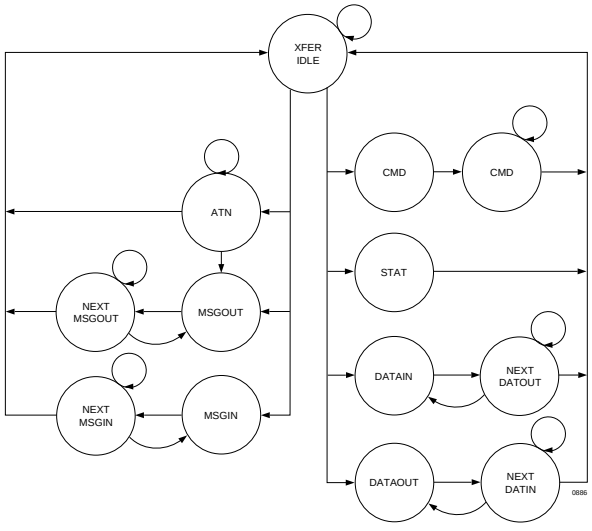


Figure 7. Data Transfer State Machine Diagram (XFER_SM)

ARBITRATION PHASE - In the arbitration phase, one or more devices will try to gain control of the SCSI bus. In state machine ARB_SM shown in Figure 5 and Listing 2, the PSC asserts BSY and drives its SCSI ID bit onto the eight-bit data bus. At the same time, the PSC reads the data bus to determine whether a device with higher priority wants control of the bus. The highest priority device wins control of the bus and continues to assert the BSY and SEL signals. All other devices participating in the arbitration must release BSY and their SCSI ID bit when SEL become active.

SELECTION PHASE - After winning the arbitration, the PSC (acting as initiator) asserts both SEL and BSY signals. Figure 6 shows the Selection state diagram and Listing 3 details the ABEL implementation. In state machine SEL_SM, to select a target, the PSC releases the BSY signal, drives the target's SCSI ID bit and its own ID bit active on the data bus, and de-asserts the I/O signal. The PSC will continue to drive SEL until the target asserts BSY.

ATTENTION CONDITION - The PSC may assert the ATN signal during the Selection phase and while the SEL signal is still asserted, indicating that it wants the target to go to the Message Out phase immediately after the Selection phase.

MESSAGE OUT PHASE - Figure 7 and Listing 4 show the Data Transfer State Machine. During the Message Out phase, the target asserts CD and MSG and de-asserts I/O. The PSC sends the Identify message to indicate which logical unit of the target is to be selected and that the PSC supports the Disconnect/Reselect operation.

COMMAND PHASE - The target starts the Command phase by asserting CD and de-asserting I/O and MSG to indicate the Command phase. The PSC responds by sending the command information to the target.

MESSAGE IN PHASE - When the target has determined that it needs to perform a disconnect operation, it asserts the CD, I/O and MSG signals indicating the Message In phase. The PSC then reads the Disconnect message from the target.

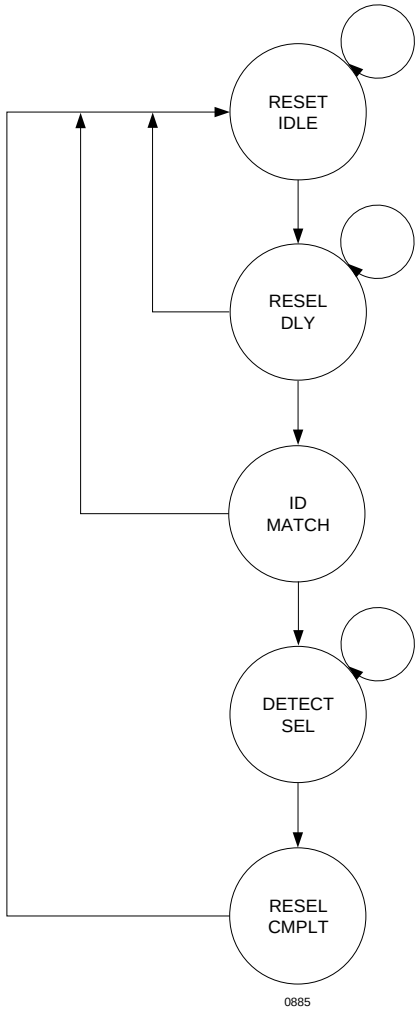
DISCONNECTED STATE - After sending the Disconnect message, the target goes to a disconnected state, suspending operations in the Bus Free phase by de-asserting all control signals.

Listing 4

```
STATE_DIAGRAM xfer_sm

STATE xfer_idle:
  IF (atno == 1) THEN xfer_idle
    WITH atno.k = 1
" Clear ATN "
  ELSE IF (atn_cmd) THEN atn_st
      " Attention requested "
    WITH atno.j = 1
" Assert ATN "
  ELSE IF (reqi & cmd_phs) THEN cmd_st
  ELSE IF (reqi & stat_phs) THEN stat_st
  ELSE IF (reqi & datin_phs) THEN
datin_st
  ELSE IF (reqi & datout_phs) THEN
datout_st
  ELSE IF (reqi & msgout_phs) THEN
msgout_st
  ELSE IF (reqi & msgin_phs) THEN
msgin_st
  ELSE xfer_idle;

STATE atn_st:
  IF (req & msgout_phs) THEN msgout_st
  ELSE IF (req & !msgout_phs)
    THEN func_cmpl_st
  ELSE atn_st;

STATE cmd_st:
  sdout_reg.d = din_reg.q
  xfr_cnt.d = xfr_cnt.q - 1
  ack = 1
  GOTO next_cmd_st;

STATE next_cmd_st:
  IF (req & !cmd_phs) # (xfr_cnt == 0)
    THEN func_cmpl_st
  ELSE IF (req & cmd_phs) THEN
send_cmd_st
  ELSE next_cmd_st;
" Wait for REQ "

STATE stat_st:
  dout_reg.d = sdin_reg.q
  ack = 1
  IF (parity == 0) THEN func_cmpl_st
      " Check SCSI data parity "
    WITH bad_parity = 1
  ELSE func_cmpl_st
```

```
STATE datain_st:
  dout_reg.d = sdin_reg.q
  ack = 1
  IF (parity == 0) THEN func_cmpl_st
    WITH bad_parity = 1
  ELSE next_datin_st;

STATE next_datin_st:
  IF (req & !datain_phs) # (xfr_cnt ==
0)
    THEN func_cmpl_st
  ELSE IF (req & datain_phs) THEN
datain_st
  ELSE next_datin_st;

STATE dataout_st:
  sdout_reg.d = din_reg.q
  xfr_cnt.d = xfr_cnt.q - 1
  ack = 1
  GOTO next_datout_st;

STATE next_datout_st:
  IF (req & !dataout_phs) # (xfr_cnt ==
0)
    THEN func_cmpl_st
  ELSE IF (req & dataout_phs) THEN
dataout_st
  ELSE next_datout_st;

STATE msgout_st:
  sdout_reg.d = din_reg.q
  xfr_cnt.d = xfr_cnt.q - 1
  ack = 1
  GOTO next_msgout;

STATE next_msgout:
  IF (req & !msgout_phs) # (xfr_cnt ==
0)
    THEN func_cmpl_st
  ELSE IF (req & msgout_phs) THEN
msgout_st
  ELSE next_msgout;

STATE msgin_st:
  dout_reg.d = sdin_reg.q
  xfr_cnt.d = xfr_cnt.q - 1
  ack = 1
  IF (parity == 0) THEN func_cmpl_st
    WITH bad_parity = 1
  ELSE next_msgin;

STATE next_msgin:
  IF (req & !msgin_phs) # (xfr_cnt == 0)
```

Figure 8. Reselection State Machine Diagram (RESEL_SM)




```
    THEN func_cmpl_st
  ELSE IF (req & msgin_phs) THEN
msgin_st
  ELSE next_msgin;

END
```

Target Reselects PSC

The target remains in the disconnected state until it is ready to continue with the next SCSI operation. The PSC is also in the disconnected state until the target reselects it.

ARBITRATION PHASE - Before reselecting the PSC, the target goes through the arbitration process to acquire control of the SCSI bus. The target asserts BSY and its SCSI ID bit.

RESELECTION PHASE - The target drives its SCSI ID, and the PSC's ID on to the data bus, and then asserts SEL and I/O and de-asserts BSY. The PSC reads its SCSI ID and the control signals to determine that it has been reselected by the target. Figure 8 and Listing 5 show the Reselection State Machine.

When reselected, the PSC responds to the target by asserting BSY. The target then drives the BSY signal active and releases SEL, thus indicating the end of the Reselection phase. When the PSC detects the SEL signal going inactive, it releases the BSY signal. However, BSY will still be held active because the target is driving it. This transfer of control is necessary because it allows the target to regain control of the BSY signal and control the usage of the bus.

MESSAGE IN PHASE - The target asserts CD, I/O and MSG to indicate the Message In phase and to send a message to the PSC. The target places the message byte on the data bus and asserts the REQ signal to indicate the beginning of data transfer.

DATA IN PHASE - The target begins the Data In phase by asserting I/O and de-asserting CD and MSG. The target then places the first byte of the data on the data bus and starts the transfer protocol. After reading the data byte, the PCK acknowledges the transfer. The target continues to transfer bytes in the same manner until all requested data have been transferred.

STATUS PHASE - To begin the Status phase, the target asserts CD and I/O and de-asserts MSG. The target then places the status information on the data bus and begins the transfer protocol. The PSC reads the status byte and completes the transfer process.

MESSAGE IN PHASE - The target asserts the CD, I/O and MSG signals indicating the Message In phase. The target places the message byte on the data bus and begins the transfer protocol. The PSC reads the message byte and completes the transfer process.

BUS FREE PHASE - After sending the "Command Complete" message, the target releases control of the SCSI bus by de-asserting all control signals. After the PSC and target physically and logically disconnect from the bus, the Bus Free phase begins.

Listing 5

```
STATE_DIAGRAM reselection

STATE resel_idle_st:
  IF (resel_phs) THEN resel_dly_st
  ELSE resel_idle_st;

STATE resel_dly_st:
  IF (timer == bsst_dly & resel_phs)
THEN id_match_st
  ELSE IF (!resel_phs) THEN
resel_idle_st
  ELSE resel_dly_st
    WITH timer.d = timer.q + 1;

STATE id_match_st:
  IF (tar_id_match & psc_id_match &
parity)
    THEN detect_sel_st WITH bsyo.j = 1
  ELSE resel_idle_st;

STATE detect_sel_st:
  IF (!seli) THEN resel_cmp_st WITH
bsyo.k = 1
  ELSE detect_sel_st;

STATE resel_cmp_st:
  resel_cmpl = 1;
  GOTO resel_idle_st;

END
```

Summary

Lattice's ispLSI 3256A is the ideal solution for implementing a Programmable SCSI Controller. The ispLSI 3256A's input registers allows asynchronous signals to be synchronized to the PSC's system clock. The in-system programmability and reconfigurability of the ispLSI 3256A enables different SCSI configurations to be implemented or upgraded without the need to remove the device from the board.

Technical Support Assistance

Hotline:	1-800-LATTICE (Domestic) 1-408-826-6002 (International)
e-mail:	techsupport@latticesemi.com