

PCI 9080/860 AN

MPC860 PowerQUICC™ to PCI bus Application Note

Features

- Complete Application Note for designing a PCI adapter or embedded system based on the Motorola MPC860 PowerQUICC including:
 - Detailed Design Description
 - OrCad Schematics
 - Verilog HDL Source Code
 - PLD Compiler and Timing Report files
 - Timing Diagrams
- Superior PCI performance based on PCI 9080 bus master interface chip which supports:
 - PCI burst, master, DMA and slave cycles
 - PCI configuration cycles
 - Asynchronous PCI/ MPC860 operation
 - Zero wait state infinite burst data transfers
 - I₂O™ Messaging Unit
 - MPC860 RETRY function support
 - Support for MPC860's IDMA channel
- Combined with PLX's I2OSDK®, provides a powerful tool for developing an MPC860-based I₂O IOP

General Description

This application note describes how to interface the Motorola MPC860 PowerQUICC CPU to the PCI bus for high performance data transfers using the PLX PCI 9080 "PCI to Local Bus Bridge" IC. The information can be used to build either a PCI adapter or embedded system.

The PCI 9080 has Direct Master, DMA and Direct Slave data transfer capabilities. Use of the PCI 9080's dual DMA channels minimize processor overhead while overcoming performance limitations associated with IDMA and SDMA of the MPC860. Direct Master mode allows a device (MPC860) on the Local Bus to perform memory, I/O, and configuration cycles to the PCI bus in addition to supporting RETRY functions of the MPC860, backing off the local bus, and allowing for high performance throughput. The Direct Slave gives a master device on the PCI bus the ability to access memory or I/O on the Local Bus. The PCI 9080 allows the Local Bus to run asynchronously to the PCI bus through the use of bi-directional FIFOs. In this design example, the PCI Bus runs at 33 MHz while the Local Bus is clocked at 40 MHz.

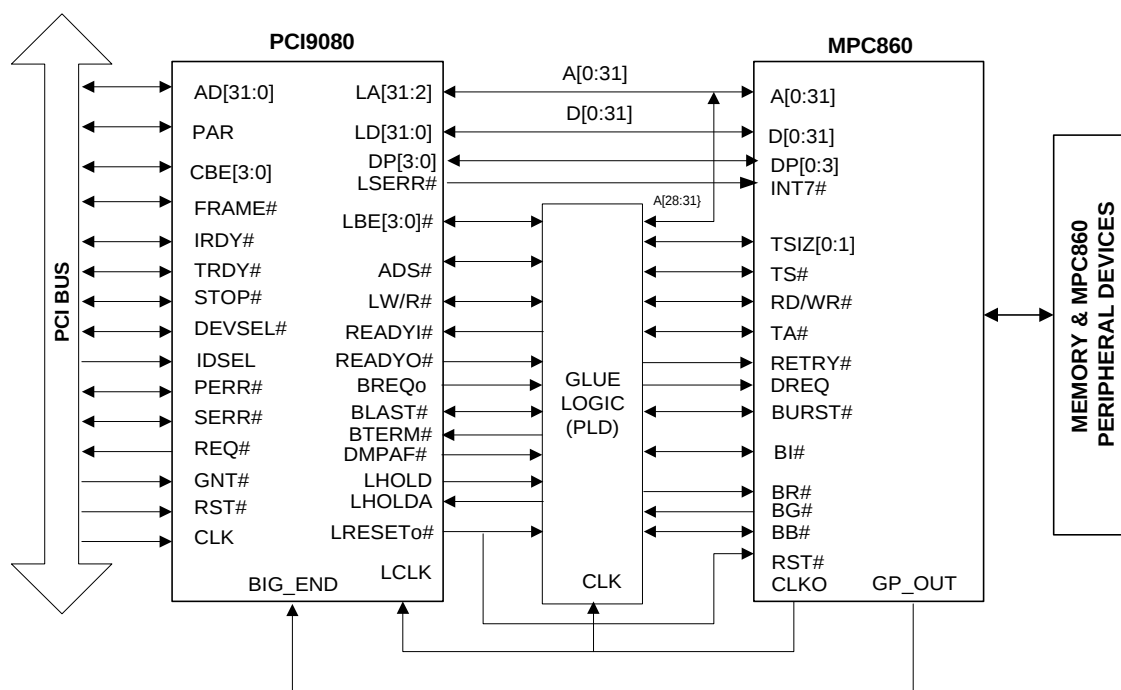


Figure 1. PCI to MPC860 Interface. MPC860 subsystem, including memory, is shown at right.

Table of Contents:

1	INTRODUCTION	3
2	LOGIC SOURCE CODE AND SCHEMATICS	3
3	DESIGN METHODOLOGY	3
4	CIRCUIT OPERATION	3
4.1	CONFIGURATION	3
4.2	DIRECT MASTER TRANSFERS	4
4.3	DIRECT SLAVE TRANSFERS	4
4.4	POWERQUICC MPC850/860 DMA MODES	5
4.5	PCI 9080 AND MPC850/860 DATA TRANSFERS	5
4.5.1	Configuration Registers Access	6
4.5.2	Direct Master (DM) Operation	6
4.5.3	Direct Slave (DS) Operation	6
4.5.4	PCI 9080 DMA Operation	6
4.5.5	MPC850/860 DMA Operation (IDMA / SDMA)	7
4.6	FEATURES UPDATED IN THE APPLICATIONS NOTE	7
5	OPERATING SPEED ANALYSIS.....	7
5.1	CRITICAL PATH	7
5.2	PLD SELECTION	8
5.3	ADDITIONAL SPEED AND LOGIC COMBINATIONS	8
6	ASSUMPTIONS.....	8
	VERILOG HDL SOURCE CODE FOR GLUE LOGIC.....	10
	PLSI2032 COMPILER REPORT FILE	16
	PLSI2032 TIMING REPORT FILE.....	37
APPENDIX:		
OrCAD SCHEMATICS		
TIMING DIAGRAMS		

1 INTRODUCTION

This application note describes how to interface the Motorola MPC860 CPU to the PCI bus using the PLX PCI 9080 "PCI to Local Bus Bridge" IC. The information can be used to build either a PCI adapter or embedded system. The design described in this note consists of an MPC860 and associated memory connected to a PLX Technology PCI 9080 chip through "glue logic". The glue logic is implemented in a Lattice 44 pin CPLD. With the source code provided, this glue logic can be customized or transferred to other programmable logic or ASIC devices. Figure 1, is a block diagram showing connections between the PCI 9080, PLD and the MPC860 subsystem which includes the CPU and the memory.

The PCI 9080 has both Direct Master (DM), DMA and Direct Slave (DS) transfer capabilities. The Direct Master mode is the mode by which the MPC860 transfers data to and from the PCI bus. This design was optimized to allow the fastest possible burst data transfers between the MPC860 and the PCI bus. The Direct Master mode allows a device (MPC860) on the Local Bus to perform memory, I/O and configuration cycles to the PCI bus. Configuration cycles would be used only if the MPC860 is the PCI system controller in an embedded system. In this application, the MPC860 may perform Type 0 and 1 configuration cycles.

The PCI 9080 also has a powerful two channel chaining DMA controller providing an alternative to the IDMA and SDMA controllers of the MPC860. Use of the PCI 9080 DMA controller is described in detail in section 4.5.4 of this document. Use of IDMA and SDMA controllers of the MPC860 are also described in detail in section 4.5.5 of this document.

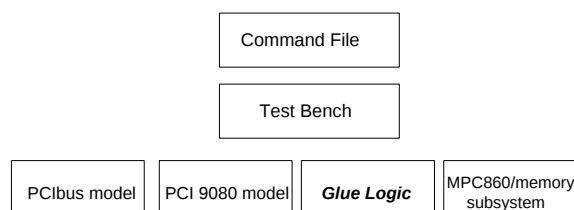
The Direct Slave mode gives a master device on the PCI bus the ability to access the MPC860 configuration registers or memory on the Local Bus. This design allows burst or single cycle direct slave transfers.

The PCI 9080 allows the Local Bus to run asynchronously to the PCI bus through the use of bi-directional FIFOs. In this application, the PCI Bus runs at 33 MHz while the Local Bus can be clocked at 25 MHz, 33 MHz, or as high as 40 MHz.

2 LOGIC SOURCE CODE AND SCHEMATICS

The appendices of the application note contain detailed logic source code and schematics. Except for the timing diagrams, this information is also available in source code from the PLX Technology web page (<http://www.plxtech.com>) including:

1. OrCAD Schematics of a design containing the MPC860, memory, PCI 9080 and the Lattice pLSI2032-44LJ programmable logic device. **Designers should note that on the MPC860, A0 is the most-significant bit and on the 9080 A31 is the most-significant bit.**
2. Verilog HDL source code of the glue logic. The Verilog design is organized as follows. The command file, test bench and model source code are not included in the application note. Contact PLX for information on these modules.



3. Lattice PLD compiler and timing report files. The compiler report shows the reduced Boolean equations for the glue logic.
4. Timing diagrams.

3 DESIGN METHODOLOGY

Verilog HDL was used for PLD design entry. The Verilog code was targeted to a Lattice pLSI2032-44LJ CPLD using Synopsys with Lattice libraries, to generate an EDIF netlist. Device mapping from EDIF netlist was achieved using the Lattice ispDS+ Development System. Verilog compilation/simulation was performed using the Cadence Verilog XL Simulator, and Undertow Verilog waveform viewer.

4 CIRCUIT OPERATION

4.1 Configuration

The PCI 9080 can be configured through its PCI or Local Bus interface or an EEPROM. See the PCI 9080 data sheet for a detailed description of the PCI

configuration cycle and general chip configuration options.

For this application note, ADMODE is tied HIGH, and S[2:0] = b001 to place Local Bus configuration space at address 0x20000000.

A general-purpose output from the MPC860 is used to control Endian mode, via the PCI 9080 BIGEND# input. Use of the BIGEND# input is application specific but in general, Little Endian mode should be used for 32-bit data as is the case when configuring the PCI 9080 from the Local side. Big Endian should normally be used for byte-oriented data. **Note that on the MPC860, A0 is the most-significant bit and on the 9080 A31 is the most-significant bit.**

4.2 Direct Master Transfers

In Direct Master Mode, the PCI 9080 translates the MPC860's Local bus master cycles into PCI bus master cycles. Through PCI to Local Bus memory-mapping, the MPC860 can perform PCI memory, I/O, and configuration cycles via the PCI 9080.

The PCI 9080 enables efficient use of the PCI bus by translating MPC860 cycles into PCI bursts. See the PCI 9080 data sheet for information on Direct Master performance.

In a Direct Master operation the MPC860 arbitrates for the Local bus via its internal arbiter. When the PLD sees BB# active it performs the following:

- TS# is buffered and then used to drive PCI9080.ADS#.
- PCI9080.READYO# is buffered and then used to drive TA#.
- RD_WRN is inverted and then used to drive PCI9080.WR_RDN.
- BI# is asserted if the MPC860 starts a non-16-byte-aligned burst.
- Generates PCI9080.LBE# from RD_WRN, TSIZE, A[30:31].
- Generates PCI9080.BLAST# from TS#, BURST#, A[28:29], PCI9080.READYO#.

4.3 Direct Slave Transfers

In Direct Slave Mode, the PCI 9080 acts as a target (slave) on the PCI bus and master on the Local bus. This mode provides a PCI master access to Local bus slave devices (typically memory).

In a Direct Slave transfer, the PLD performs the following:

- Converts PCI 9080 bus request (LHOLD) handshake into MPC860 bus request handshake.
- ADS# is delayed by 2 clock cycles and then used to drive TS# (see below).
- WR_RDN is inverted/buffered and then used to drive MPC860.RD_WRN.
- Generates BTERM# from LHOLD, BLAST#, MPC860.TA#, MPC860.A[28:29], MPC860.BI#.
- Generates MPC860.BURST# from LHOLD, ADS#, BLAST#, BTERM#, MPC860.TA#.
- Generates MPC860.TS, MPC860.TSIZE, MPC860.A[30:31] from ADS#, WR_RDN, LBE#.

The PLD processes the PCI 9080 Local Bus request as follows, where each step is synchronous with the Local Bus clock rising edge (LCLK[^]):

- PCI 9080 asserts its LHOLD output which is an input to the PLD.
- PLD asserts MPC860.BR# (External Bus Request input to internal arbiter).
- The MPC860 internal arbiter asserts BG# (External Bus Grant).
- With MPC860.BG# asserted and the MPC860 BB negated, the PLD will assert PCI9080.LHOLD and MPC860.BB#.
- PLD negates MPC860.BR#.

It takes 2 clock cycles to detect BLAST from the PCI 9080 and determine if the transfer is a burst and thus generate MPC860.BURST#. Also, since BURST must be present along with TS, it is necessary to delay TS by 2 clock cycles. Thus, single-cycle DS transfers will take 4 clock cycles. A 4-Dword burst will take 7 clocks.

The PLD breaks a Direct Slave (DS) unaligned transfer into 2 cycles when necessary for compatibility with the MPC860. This process is implemented by the CYC_CNT state machine. First, the WR_RDN and LBE lines are encoded into XFER_TYPE = READ_CYC, ONE_CYC, or TWO_CYC according to the following rules:

- DS reads become Dword reads regardless of the LBE pattern (READ_CYC).
- DS single-byte and Dword writes (ONE_CYC).

- DS upper/lower 16-bit writes (ONE_CYC).
- Other DS writes (TWO_CYC).

The CYC_CNT state machine generates RDY_GATE and internal versions of TS, TSIZE, and A[30:31] (TS_R, TSIZE_R, A3031_R). RDY_GATE is used to hold-off the generation of PCI9080.READYI when a two-cycle transfer is indicated.

The CYC_CNT state machine flow is as follows:

state 0 (CYC_CNT=0):

TS_R is delayed (2 clock periods) version of ADS#;

Generate TSIZE_R and A3031_R as needed for first (or only) cycle;

if (XFER_TYPE=TWO_CYC):

negate RDY_GATE,

wait for TA active,

proceed to state 1;

else stay in state 0.

state 1 (CYC_CNT=1):

Pulse TS_R for one clock cycle (TS_DIS used for this purpose);

Generate TSIZE_R and A3031_R as needed for second cycle;

Enable READYI by asserting RDY_GATE; wait for TA active; return to state 0.

Note that in this design the PCI 9080 BREQ input is not used. This input, when used, allows the MPC860, through the arbiter circuit, to force the PCI 9080 to release the local bus. When BREQ is not used, the PCI 9080 will hold the local bus until it either completes the transfer or its internal local bus latency timer expires.

4.4 PowerQuicc MPC850/860 DMA Modes

The PowerQuicc processor contains two physical DMA controllers. Using these two controllers and the internal RISC engine, allows emulation of up to 16 serial DMA channels (SDMA), or two Independent DMA channels (IDMA).

The SDMA channels are used to communicate with the internal serial devices (located in the Communications Processor Module) and internal or external memory. (The PowerQuicc contains 5,120 bytes of static Dual-Port RAM that is used by the RISC processor, CPU, serial channels, and for DMA buffer descriptors.) The SDMA acts as an internal

bus master when data is transferred between one of the serial devices and the internal dual-port RAM. For transfers between the serial devices and external memory, the SDMA channel arbitrates for the external system bus and performs the data transfer (which may be byte, half-word, word, or burst of 4 Long words).

The IDMA channels provides general-purpose DMA functionally through the SDMA channels. They support memory-to-memory or peripheral-to-memory transfers (where a peripheral is a device on the external system bus).

IDMA differs from the SDMA mode because it uses two dedicated control signals per channel -- DMA Request (DREQx) and DMA Acknowledge (SDACKx). These signals are used by the peripheral for Demand Mode DMA operation (DREQ is asserted by the peripheral to request data transfer, and SDACK asserted by the MPC860 to acknowledge the transfer). Memory-to-memory transfers are supported if a timer is connected to DREQ and SDACK to control the transfer pace (DREQ and SDACK are connected to the input/output pins of a timer to control the data transfers between memory devices).

IDMA can also support peripheral-to-peripheral (one external Demand Mode DMA device to another) by using both IDMA channels to handle data transfer (called dual-address mode). This mode uses the internal RAM for temporary storage and can perform data-size translation (i.e., 32-bit device to an 8-bit device) in multiple cycles

Note: The MPC860 can only burst 16 bytes (four long words) of data during one transaction.

Refer to the MPC850 or MPC860 User's Manual for more information.

4.5 PCI 9080 and MPC850/860 Data Transfers

The PCI 9080 communicates with the MPC850/860 in five ways:

- Configuration Registers Access
- Direct Master Operation
- Direct Slave Operation
- DMA Operation
- IDMA Operation

Note: The following discussion assumes the MPC860's internal arbiter is enabled, therefore, only

allowing one external Local Bus Master (the PCI 9080).

4.5.1 Configuration Registers Access

The Local Bus can access all the internal configuration registers of the PCI 9080 through the Configuration Chip Select pin (**CS9080#**).

The PCI 9080 internal registers can also be accessed from the PCI Bus by a memory cycle that matches the base address specified in the PCI Base Address 0 configuration register, or through a PCI I/O cycle (using PCI Base Address 1).

4.5.2 Direct Master (DM) Operation

The MPC860 transfers data to/from the PCI Bus through a 64-byte DM Read FIFO (16 by 32-bits) and a 128-byte DM Write FIFO (32 by 32-bits) of the PCI 9080. The MPC860 can also transfer data using its SDMA or IDMA channels. All transfer must be a single or burst of 4 long word read/write memory transactions to the PCI 9080 (and PCI Bus).

Note: MPC850/860 SDMA or IDMA accesses to the PCI 9080 appear as Direct Master operations. Refer to the IDMA section for more information.

Transactions are initiated by the MPC860 (a Local Bus Master) when the memory address on the Local Bus matches the memory space decoded for Direct Master operations. Upon a Local Bus Read, the PCI 9080 becomes a PCI Bus Master; arbitrates for the PCI Bus; and reads data from the PCI Target into the DM Read FIFO. When the data is on the Local Bus, PCI 9080 asserts **LREADYo#** which is translated to the Transfer Acknowledge (**TA#**) on the MPC 850/860 Bus to indicate data available.

Upon a Local Bus Write, data is written to the DM Write FIFO by the Local Bus Master (MPC850/860 CPU write or the SDMA/IDMA). When enough data is in the FIFO, the PCI 9080 becomes a PCI Bus Master; arbitrates for the PCI Bus; and writes the data to the PCI Target device.

During Direct Master transactions, the PCI 9080 will assert the **BREQo#** signal to indicate possible DeadLock. The **BREQo#** signal and the **DMPAF#** (Direct Master Programmable Almost Full Flag) are used to generate the MPC850/860 **RETRY#** signal (for further information, please refer to section 4.6 of this document).

- Any time the PCI 9080 detects a “deadlock” situation it asserts the **RETRY#** signal to the Local Master (i.e., MPC860), implying that the

bus should be relinquished to allow the Direct Slave operation (from the PCI Bus) to proceed.

- The PCI 9080 asserts the **RETRY#** signal whenever the Direct Master Write FIFO is full, implying the Local Master can relinquish the bus, and finish the write operation at a later time.

Note 1: MPC860 cycles can be a single or a burst of 4 long words only.

A single cycle from the MPC860 will cause a single cycle PCI transaction. A burst cycle from the MPC860 will generate a burst cycle PCI transaction. Bursts are limited to 16 bytes (four long words) in the MPC860.

4.5.3 Direct Slave (DS) Operation

The PCI 9080 (PCI Bus) transfers data to/from the Local Bus Slave device (i.e., MPC860) through a 64-byte DS Read FIFO (16 by 32-bits) and a 128-byte DS Write FIFO (32 by 32-bits).

Transactions are initiated by a PCI Bus Master addressing the memory space decoded for the Local Bus. Upon a PCI Read, the PCI 9080 becomes a Local Bus Master; arbitrates for the Local Bus; and reads data into the DS Read FIFO. After enough data is in the FIFO, the PCI 9080 transfers the data to the PCI Bus (it may tell the host to “retry” if data is not immediately available from the Local Bus).

Upon a PCI Write, data is read into the FIFO. When enough data is available, the PCI 9080 becomes a Local Bus Master; arbitrates for the Local Bus; and writes the data to the Local Bus device (DRAM or SRAM connected to the memory controller of the MPC860).

A single cycle (non-burst) from the PCI Bus will generate a single cycle MPC860 transaction. A burst cycle from the PCI bus will generate a MPC860 burst transaction if the address is quad Lword aligned and the FIFO has at least four long words and all the PCI Byte Enables are asserted. Otherwise, single cycle transactions occur to the MPC860.

4.5.4 PCI 9080 DMA Operation

The PCI 9080 supports two independent DMA channels capable of transferring data between the Local Bus and PCI Bus. Each channel consists of a DMA controller with programmable FIFOs. Both channels support chaining and non-chaining DMA transfers, along with the End of Transfer (**EOT#**) pin. Both DMA channels support Demand Mode DMA using its own **DREQx** and **DACKx** signals. (Demand Mode DMA and the MPC860's IDMA channel are

mutually exclusive modes and cannot be used simultaneously. See IDMA section below.)

When the PCI 9080's DMA mode is enabled, it becomes a Bus Master for both the PCI Bus and the Local Bus. Direct Master and Direct Slave transactions can occur during DMA mode operations (the PCI or Local Buses are released when a DM/DS transaction is pending).

4.5.5 MPC850/860 DMA Operation (IDMA / SDMA)

The PCI 9080 supports the MPC860 Independent DMA mode using the **SDREQ0860#** signal and operating in Direct Master (DM) mode (for further information, please refer to section 4.6 of this document). After programming the MPC860's IDMA channel, the PC 9080 uses Direct Master mode to transfer data between the PCI Bus and the MPC860's internal dual-port RAM (or external memory). The data count is controlled by the IDMA byte counter and throttled by the **SDREQ860#** signal. When the PCI 9080's FIFO is almost full (DMPAF#), the **SDREQ860#** signal is negated to the MPC860, indicating that it should inhibit transferring any more data (the FIFO threshold count in the PCI 9080 must be set to a value that is at least 5 Lwords below the FIFO's full capacity -- 27 Lwords).

After the IDMA has transferred all the required bytes (MPC860 byte counter decrements to zero), it will generate an interrupt to the MPC860, which in-turn executes code to disable the IDMA channel (the **SDREQ860#** input signal may still be asserted by the PCI 9080). The **SDACKx** signal from the MPC860 is not used by the PCI 9080 (no connection).

See the Direct Master (DM) section for more information about data transfers.

4.6 Features Updated in This Applications Note

This applications note has been updated to support Local System Error (LSERR_ to TEA#), Backoff and Retry mechanisms (BREQo to RETRY), and MPC 860 IDMA functions. In order to support these functions, the following equations must be added to the Verilog glue logic. Please refer to section entitled VERILOG HDL SOURCE CODE FOR GLUE LOGIC for further information:

```
wire    TEA_ = LSERR_ ;
wire    SDREQ0860 = ~(DMPAF_);
```

```
wire    BACKOFF = ((BB_860 & BREQO) | (~RD_WRN & ~DMPAF_ & ~TS_L));
```

```
always @(posedge CLK)                //FF
    BACKOFF_R <= BACKOFF;
```

```
assign #('localhold) RETRY_ = ~(BACKOFF & ~BACKOFF_R); //OR gate
assign #('localhold) ADS_L_ = ((BB_860 & ~BREQO) & DMPAF_) ? TS_L : OPEN;
```

5 OPERATING SPEED ANALYSIS

5.1 Critical Path

The critical combinatorial paths which determine maximum MPC860 clock rate are:

1. PCI9080.READYO# to MPC860.TA#
2. MPC860.TA# to PCI9080.READYI#
3. MPC860.TS# to PCI9080.ADS#

Calculations:

Glue Logic Tpd <= Tcyc - clock skew - wire delay -

Max ((READYO:Tco + TA:Tsu),(TA:Tco + READYI:Tsu),(TS:Tco + ADS:Tsu))

Using:

READYO:Tco = 13.5 nsec (50 Pf load)

READYI:Tsu = 7.0 nsec

ADS:Tsu = 7.0 nsec

clock skew = 1.0 nsec

wire delay = 0.5 nsec

For a 25 MHz MPC860 with a 25 MHz local bus:

TS:Tco = 19.0 nsec (50 Pf load)

TA:Tco = 11.0 nsec (50 Pf load)

TA:Tsu = 9.0 nsec

Glue Logic Tpd <= 40 - 1 - 0.5 - Max((13.5 + 9),(11 + 7),(19 + 7)) = **12.5 nsec**

For a 40 MHz MPC860 with a 40 MHz local bus:

TS:Tco = 13.0 nsec (50 Pf load)

TA:Tco = 11.0 nsec (50 Pf load)

TA:Tsu = 7.0 nsec

Glue Logic Tpd $\leq 25 - 1 - 0.5 - \text{Max}((13.5 + 7), (11 + 7), (13 + 7)) = \underline{\underline{3.0 \text{ nsec}}}$

For 40 MHz MPC860 with a 33 MHz local bus:

Glue Logic Tpd $\leq 30 - 1 - 0.5 - \text{Max}((13.5 + 7), (11 + 7), (13 + 7)) = \underline{\underline{8.0 \text{ nsec}}}$

The above calculations for Glue Logic Tpd will actually improve if Tco for READY0, TS, and TA is derated to account for loading of less than 50 Pf.

5.2 PLD Selection

The pLSI2032 comes in several speed grades with the following Tpd:

Speed (MHz)	Tpd (ns)	Tpd(ns)*
180	7.5	5
150	8.0	5.5
135	10.0	7.5
110	13.0	10.0
80	18.5	15.0

* using 4PT Bypass & ORP Bypass

We will use the lower Tpd given above. These numbers are attainable as shown in the Timing Report by specifying a property file when compiling the PLD. (See appendix).

Given the max Tpd values calculated above, and the faster Tpd for a given pLSI2032, the PLD speed selection is as follows:

MPC860(MHz)	Local Bus (MHz)	pLSI2032(MHz)
25	25	110
40	33	135

For example, to achieve 33 MHz operation of the local bus, a 40 MHz MPC860 should be used along with a 180 MHz pLSI 2032.

5.3 Additional Speed and Logic Combinations

The source code provided can be combined with other logic or targeted to other devices.

For example to achieve 40 MHz local bus operation, the critical path signals described previously should be implemented in a fast PLD or logic device. As seen in the max Tpd numbers shown previously, a 3 nsec Tpd is required for several of the critical path signals to achieve 40 MHz local bus operation. At the time of the writing of this application note, this speed grade is not available in the pLSI2032. Therefore, the design could be partitioned to implement the critical path signals in a small PLD and the rest of the logic in an FPGA or CPLD.

6 ASSUMPTIONS

This application note is based on the following assumptions:

- Access to local memory is via the MPC860 Memory Controller only. Since the Memory Controller always provides TA# (which becomes PCI9080.READYI#), there is no need for PCI 9080 wait states. If wait states are inserted, the logic must be changed to accommodate.

- The MPC860 is configured to use Internal Arbiter.

- The BDIP signal is not used or generated.

- All MPC860-initiated bursts will be 16-byte-aligned; otherwise the BI# input is asserted by external glue logic, preventing the burst.

- The MPC860 Memory Controller programming dictates that a burst be a fixed number of data cycles, and since the MPC860 core always bursts in 16-byte lengths, the Memory Controller must be programmed for 16-byte bursts. This plus the fact that the PCI9080 breaks-up bursts at 16-byte boundaries, necessitates that all bursts of greater than 16-bytes be 16-byte-aligned. Restricting bursts as such, satisfies a DRAM requirement as well; bursts from the PCI9080 (DS) targeted to DRAM memory, become page-mode accesses which by starting on a 16-byte boundary, avoid crossing a page boundary.

The above requirements are satisfied as follows:

1. User S/W must insure that PCI9080 initiated bursts be 16-byte-aligned and multiples of 16 bytes.
2. Either
 - a) the PCI9080 BTERM input is disabled which forces breaks in the burst at 16-byte boundaries; or

b) the PLD BTERM output is enabled (BTERM_EN input asserted).

It should be noted that this 16-byte burst length restriction is based on use of the MPC860 Memory Controller as stated above. The user can choose to

provide an application-specific memory controller and thus avoid this restriction.

- The AT[0:3] lines are ignored by the PCI 9080 and external logic. These lines are used only for diagnostic/protection purposes.

VERILOG HDL SOURCE CODE FOR GLUE LOGIC

Note: Please refer to section 4.6 of this document for additional verilog source code required to support NMI, RETRY, and SDREQo860 for IDMA features.

```

////////////////////////////////////////////////////////////////
//
// Glue Logic - PCI9080 to MPC860 I/F
//
// 6/12/97 : Bob Cannataro
//          PLX Technology
//
////////////////////////////////////////////////////////////////

// Notes
//
// For Direct Slave Transfers, BTERM_EN input active (HIGH) will prevent any
// bursts from crossing a 16-byte boundary, via generation of BTERM. If BTERM_EN
// is inactive (LOW), the PCI9080 with BTERM disabled, will also prevent a burst
// from crossing a 16-byte boundary.
//
// For Direct Master Transfers, the PCI9080 is considered a 32-bit port, only,
// and bursts are always 16-byte aligned.
//
// This logic assumes that any Local Bus targets will be generate READYI_L, and
// will break if the PCI9080 is programmed to generate wait states.
//
// BDIP_L is not generated due to complexity.
// low-active signals are indicated via the suffix "_L"

`define localhold 5

module MPC860_GLUE (RST_L, CLK, ADS_L, LBE_L, WR_RDN, READYI_L, READYO_L,
BLAST_L, BTERM_EN, BTERM_L, LHOLD, LHOLDA, TS_L, A28, A29, A30, A31, TSIZE, RD_WRN,
TA_L, BI_L, BURST_L, BR_L, BG_L, BB_L, LHOLDA_FB, TS_R, RDY_GATE);

//PCI9080 I/F

input          CLK;
input          LHOLD, READYO_L;
output         LHOLDA, READYI_L, BTERM_L;
inout          ADS_L, WR_RDN, BLAST_L;
inout [3:0]    LBE_L;

//MPC860 I/F

input          BG_L;
input          A28, A29;
inout          A30, A31;
inout [0:1]    TSIZE;
inout          TS_L, RD_WRN, TA_L, BURST_L, BB_L, BI_L;
output         BR_L;

input          RST_L;
input          BTERM_EN, LHOLDA_FB;          // LHOLDA_FB is tied to ISP2032 OE pin
                                              // to provide second OE control
output         TS_R, RDY_GATE;              // for use by PAL16V8 external bypass

reg            BR;

```

```

reg      LHOLDA;      //indicates 9080 has the bus
reg      TS_R;        // internally generated TS
reg      TS_DIS;
reg      BI_R;
reg      BLAST_R;
reg      BTERM_R;
reg      BURST_R;
reg      ADS_PL;
reg      RDY_GATE;
reg      CYC_CNT;
reg      [1:0] XFER_TYPE;
reg      [0:1] A3031_R;
reg      [0:1] TSIZE_R;

//Convert inputs to positive logic;

wire      RST          = ~RST_L;
wire      ADS          = ~ADS_L;
wire      TS           = ~TS_L;
wire      TA           = ~TA_L;
wire      READY0       = ~READY0_L;
wire      BLAST        = ~BLAST_L;
wire      BURST        = ~BURST_L;
wire      BI           = ~BI_L;
wire      BG           = ~BG_L;
wire      BB           = ~BB_L;
wire      [3:0] LBE     = ~LBE_L;

//Convert outputs to positive logic;

wire      BR_L         = ~BR;

//Text Macros implemented as wires

wire      WRITE = WR_RDN;
wire      READ  = ~WR_RDN;

wire      BB_860 = BB & ~LHOLDA;

wire      READYI   = TA & LHOLDA & RDY_GATE;
wire      READYI_L = ~READYI;

parameter  LOW  = 1'b0,
           HIGH = 1'b1,
           OPEN = 1'bz;

/*----- MPC860 is the Local Bus Master (Direct Master)-----*/

assign #(`localhold) ADS_L   = BB_860?   TS_L           : OPEN;
assign #(`localhold) WR_RDN  = BB_860?   ~RD_WRN        : OPEN;
assign #(`localhold) BLAST_L = BB_860?   ~BLAST_R        : OPEN;
assign #(`localhold) TA_L    = BB_860?   ~READY0         : OPEN;
assign #(`localhold) BI_L    = BB_860?   ~BI_R           : OPEN;

wire [3:0] LBE_GEN = ({READ,TSIZE,A30,A31} == 5'b0_01_00)? 4'b1000 :
    ({READ,TSIZE,A30,A31} == 5'b0_01_01)? 4'b0100 :
    ({READ,TSIZE,A30,A31} == 5'b0_01_10)? 4'b0010 :
    ({READ,TSIZE,A30,A31} == 5'b0_01_11)? 4'b0001 :
    ({READ,TSIZE,A30,A31} == 5'b0_10_00)? 4'b1100 :
    ({READ,TSIZE,A30,A31} == 5'b0_10_10)? 4'b0011 : 4'b1111;

```

```

assign #(`localhold) LBE_L    = BB_860?  ~LBE_GEN      : 4'bz;

always @(posedge CLK)
  if (RST)
    begin
      BI_R    <= 0;
      BLAST_R <= 0;
    end
  else
    begin
      BI_R    <= TS & BURST & ({A28,A29,A30,A31} != 4'b0);
      casez ({BB_860,TS,BURST})
        3'b0?? : BLAST_R <= 0;
        3'b100 : BLAST_R <= BLAST_R & ~READY0;
        3'b101 : BLAST_R <= ({A28,A29} == 2'b10) | BLAST_R & ~READY0;
        3'b110 : BLAST_R <= 1;
        3'b111 : BLAST_R <= 0;
      endcase
    end

/*----- PCI9080 is the Local Bus Master (Direct Slave)-----*/

parameter [1:0] NO_CYC    = 0;    // Encodings for CYC_CNT State Machine below
parameter [1:0] ONE_CYC   = 1;
parameter [1:0] TWO_CYC   = 2;
parameter [1:0] READ_CYC  = 3;

assign #(`localhold) BB_L      = LHOLDA_FB?  LOW      : OPEN;
assign #(`localhold) RD_WRN    = LHOLDA_FB?  ~WR_RDN   : OPEN;
assign #(`localhold) TS_L      = LHOLDA_FB?  ~TS_R     : OPEN;
assign #(`localhold) BTERM_L   = LHOLDA_FB?  ~BTERM_R  : OPEN;
assign #(`localhold) BURST_L   = LHOLDA_FB?  ~BURST_R  : OPEN;
assign #(`localhold) {A30,A31} = LHOLDA_FB?  A3031_R   : 2'bzz;
assign #(`localhold) TSIZE     = LHOLDA_FB?  TSIZE_R   : 2'bzz;

always @(posedge CLK)          // Local Bus Arbitration
  if (RST)
    begin
      BR      <= 0;
      LHOLDA  <= 0;
    end
  else
    begin
      BR <= LHOLD & ~LHOLDA;
      LHOLDA <= BG & ~BB | LHOLDA & LHOLD;
    end
end

always @(posedge CLK) ADS_PL <= ADS;

always @(posedge CLK)          // process to generate BURST_L and BTERM_L
  if (RST | ~LHOLDA)
    begin
      BURST_R <= 0;
      BTERM_R <= 0;
    end
  else
    begin
      BURST_R <= ADS_PL & ~BLAST | BURST_R & !((BLAST | BTERM_R) & TA);
      BTERM_R <= (BTERM_EN & TA & ~BLAST & ({A28,A29} == 2'b10) | BI) | BTERM_R
& !TA;
    end
end

```

```

always @(posedge CLK)          // encoding used by CYC_CNT state machine
  if (RST | ~LHOLDA)
    XFER_TYPE <= NO_CYC;
  else
    if (ADS)
      if (READ)
        XFER_TYPE <= READ_CYC;
      else
        case (LBE)
          4'b0000 : XFER_TYPE <= NO_CYC;
          4'b0001 : XFER_TYPE <= ONE_CYC;
          4'b0010 : XFER_TYPE <= ONE_CYC;
          4'b0011 : XFER_TYPE <= ONE_CYC;
          4'b0100 : XFER_TYPE <= ONE_CYC;
          4'b0101 : XFER_TYPE <= TWO_CYC;
          4'b0110 : XFER_TYPE <= TWO_CYC;
          4'b0111 : XFER_TYPE <= TWO_CYC;
          4'b1000 : XFER_TYPE <= ONE_CYC;
          4'b1001 : XFER_TYPE <= TWO_CYC;
          4'b1010 : XFER_TYPE <= TWO_CYC;
          4'b1011 : XFER_TYPE <= TWO_CYC;
          4'b1100 : XFER_TYPE <= ONE_CYC;
          4'b1101 : XFER_TYPE <= TWO_CYC;
          4'b1110 : XFER_TYPE <= TWO_CYC;
          4'b1111 : XFER_TYPE <= ONE_CYC;
          default : XFER_TYPE <= 2'bxx;
        endcase
    else
      XFER_TYPE <= XFER_TYPE;

always @(posedge CLK)          // CYC_CNT state machine to handle un-aligned transfers
  if (RST | ~LHOLDA)
    begin
      CYC_CNT <= 0;
      RDY_GATE <= 1;
      TS_R <= 0;
      TS_DIS <= 0;
      TSIZE_R <= 0;
      A3031_R <= 0;
    end
  else
    case (CYC_CNT)
      0 : case (XFER_TYPE)
          READ_CYC : begin
            TS_DIS <= 0;
            TS_R <= ADS_PL;
            CYC_CNT <= 0;
            RDY_GATE <= 1;
            TSIZE_R <= 2'b00;
            A3031_R <= 2'b00;
          end
          ONE_CYC : begin
            TS_DIS <= 0;
            TS_R <= ADS_PL;
            CYC_CNT <= 0;
            RDY_GATE <= 1;
            case (LBE)
              4'b0001 : begin
                TSIZE_R <= 2'b01;
                A3031_R <= 2'b11;
              end
            endcase
          end
        endcase
    endcase

```

```

        end
        4'b0010 : begin
            TSIZE_R <= 2'b01;
            A3031_R <= 2'b10;
        end
        4'b0011 : begin
            TSIZE_R <= 2'b10;
            A3031_R <= 2'b10;
        end
        4'b0100 : begin
            TSIZE_R <= 2'b01;
            A3031_R <= 2'b01;
        end
        4'b1000 : begin
            TSIZE_R <= 2'b01;
            A3031_R <= 2'b00;
        end
        4'b1100 : begin
            TSIZE_R <= 2'b10;
            A3031_R <= 2'b00;
        end
        4'b1111 : begin
            TSIZE_R <= 2'b00;
            A3031_R <= 2'b00;
        end
    endcase
end
TWO_CYC : begin
    TS_DIS <= 0;
    TS_R <= ADS_PL;
    if (TA) CYC_CNT <= 1;
    RDY_GATE <= 0;
    case (LBE)
        4'b0101 : begin
            TSIZE_R <= 2'b01;
            A3031_R <= 2'b01;
        end
        4'b0110 : begin
            TSIZE_R <= 2'b01;
            A3031_R <= 2'b01;
        end
        4'b0111 : begin
            TSIZE_R <= 2'b01;
            A3031_R <= 2'b01;
        end
        4'b1001 : begin
            TSIZE_R <= 2'b01;
            A3031_R <= 2'b00;
        end
        4'b1010 : begin
            TSIZE_R <= 2'b01;
            A3031_R <= 2'b00;
        end
        4'b1011 : begin
            TSIZE_R <= 2'b01;
            A3031_R <= 2'b00;
        end
        4'b1101 : begin
            TSIZE_R <= 2'b10;
            A3031_R <= 2'b00;
        end
    end
end

```

```

        4'b1110 : begin
            TSIZE_R <= 2'b10;
            A3031_R <= 2'b00;
        end
    endcase
end
NO_CYC : begin
    TS_DIS <= 0;
    TS_R <= 0;
    CYC_CNT <= 0;
    TSIZE_R <= 0;
    A3031_R <= 0;
    RDY_GATE <= 0;
end
endcase
1 : case (XFER_TYPE)
    TWO_CYC : begin
        TS_DIS <= 1;
        TS_R <= ~TS_DIS;
        if (TA) CYC_CNT <= 0;
        RDY_GATE <= 1;
        case (LBE)
            4'b0101 : begin
                TSIZE_R <= 2'b01;
                A3031_R <= 2'b11;
            end
            4'b0110 : begin
                TSIZE_R <= 2'b01;
                A3031_R <= 2'b10;
            end
            4'b0111 : begin
                TSIZE_R <= 2'b10;
                A3031_R <= 2'b10;
            end
            4'b1001 : begin
                TSIZE_R <= 2'b01;
                A3031_R <= 2'b11;
            end
            4'b1010 : begin
                TSIZE_R <= 2'b01;
                A3031_R <= 2'b10;
            end
            4'b1011 : begin
                TSIZE_R <= 2'b10;
                A3031_R <= 2'b10;
            end
            4'b1101 : begin
                TSIZE_R <= 2'b01;
                A3031_R <= 2'b11;
            end
            4'b1110 : begin
                TSIZE_R <= 2'b01;
                A3031_R <= 2'b10;
            end
        endcase
    end
endcase
endcase
endcase
endmodule

```

pLSI2032 COMPILER REPORT FILE

(is)pLSI Development System Plus Release 5.0.6, Jan 27 1997 10:33:53

Design Parameters

```

EFFORT:                2
IGNORE_FIXED_PIN:      ON
MAX_GLB_IN:            16
MAX_GLB_OUT:           4
OUTPUT_FORM:           SIM
PARAM_FILE:            _MPC860_
PIN_FILE:              MPC860_glue.xpn
STRATEGY:              DELAY
TIMING_ANALYZER:       OFF
USE_GLOBAL_RESET:      OFF
  
```

Design Specification

```

Design:                MPC860_glue
Part:                  ispLSI2032-135LJ44
  
```

```

ISP:                   OFF
ISP_EXCEPT_Y2:       OFF
PULLUP:                ON
SECURITY:              OFF
Y1_AS_RESET:           OFF
SLOWSLEW:              OFF
  
```

```

Number of Critical Pins:  4
Number of Free Pins:      32
Number of Locked Pins:    0
  
```

Input Pins

Pin Name	Pin Attribute
A28	PULLUP
A29	PULLUP
BG_L	PULLUP
BTERM_EN	PULLUP
CLK	PULLUP
LHOLD	PULLUP
LHOLDA_FB	PULLUP
READY0_L	PULLUP
RST_L	PULLUP

Output Pins

Pin Name	Pin Attribute
BR_L	PULLUP
BTERM_L	PULLUP
LHOLDA	PULLUP
RDY_GATE	PULLUP
READYI_L	CRIT, PULLUP
TS_R	PULLUP

Bidirectional Pins

Pin Name	Pin Attribute
A30	PULLUP
A31	PULLUP
ADS_L	CRIT, PULLUP
BB_L	PULLUP
BI_L	PULLUP
BLAST_L	PULLUP
BURST_L	PULLUP
LBE_L[0]	PULLUP
LBE_L[1]	PULLUP
LBE_L[2]	PULLUP
LBE_L[3]	PULLUP
RD_WRN	PULLUP
TA_L	CRIT, PULLUP
TSIZE[0]	PULLUP
TSIZE[1]	PULLUP
TS_L	PULLUP
WR_RDN	CRIT, PULLUP

Critical Paths

Path Name	Starting Net	Ending Net
PATH1		U1263_X00
PATH2		U1274_X00
PATH3		READYI_L
PATH4		U1273_X00

Pre-Route Design Statistics

Number of Macrocells: 28
 Number of GLBs: 8
 Number of I/Os: 30
 Number of Nets: 52

Number of Free Inputs: 7
 Number of Free Outputs: 5
 Number of Free Three-States: 1
 Number of Free Bidi's: 17

Number of Locked Input IOCs: 0
 Number of Locked DIs: 0
 Number of Locked Outputs: 0
 Number of Locked Three-States: 0

Number of Locked Bidi's: 0
 Number of CRIT Outputs: 4
 Number of Global OEs: 1
 Number of External Clocks: 1

GLB Utilization (Out of 8): 100%
 I/O Utilization (Out of 34): 88%
 Net Utilization (Out of 66): 78%

Nets with Fanout of 1: 22
 Nets with Fanout of 2: 15
 Nets with Fanout of 3: 5
 Nets with Fanout of 4: 5
 Nets with Fanout of 5: 3
 Nets with Fanout of 6: 1
 Nets with Fanout of 8: 1

Average Fanout per Net: 2.23

GLBs with 4 Input(s): 1
 GLBs with 7 Input(s): 1
 GLBs with 11 Input(s): 1
 GLBs with 12 Input(s): 1
 GLBs with 13 Input(s): 1
 GLBs with 14 Input(s): 2
 GLBs with 18 Input(s): 1

Average Inputs per GLB: 11.63

GLBs with 2 Output(s): 1
 GLBs with 3 Output(s): 2
 GLBs with 4 Output(s): 5

Average Outputs per GLB: 3.50

Output Enable Nets: 2

Net Name	Net Fanout
N3547	9
LHOLDA_FBX	9

Number of GLB Registers: 17
 Number of IOC Registers: 0

Post-Route Design Implementation

 Number of Macrocells: 28
 Number of GLBs: 8
 Number of IOCs: 29
 Number of DIs: 1
 Number of GLB Levels: 2

GLB glb00, A5

```

11 Input(s)
    (glb00.01, A3031_R[0], I17), (glb03.00, CYC_CNT, I15),
    (glb06.03, LHOLDAX, I8), (glb00.00, RDY_GATEX, I16), (RST_L.0,
    RST_LX, I14), (glb01.01, XFER_TYPE[0], I6), (glb03.03,
    XFER_TYPE[1]_part1, I12), (LBE_L[1].0, _DEF_356, I4),
    (LBE_L[3].0, _DEF_357, I13), (LBE_L[2].0, _DEF_367, I10),
    (LBE_L[0].0, _DEF_371, I7)
2 Output(s)
    (RDY_GATEX, 00), (A3031_R[0], 01)
19 Product Term(s)

```

Output RDY_GATEX

```

6 Input(s)
    RST_LX, XFER_TYPE[1]_part1, CYC_CNT, XFER_TYPE[0], RDY_GATEX,
    LHOLDAX
3 Fanout(s)
    glb06.I4, glb00.I16, RDY_GATE.IR
4 Product Term(s)
1 GLB Level(s)

RDY_GATEX.D = (LHOLDAX & RST_LX & !XFER_TYPE[0] & !CYC_CNT
    # CYC_CNT & LHOLDAX & RST_LX & XFER_TYPE[0] & !RDY_GATEX
    # CYC_CNT & LHOLDAX & RST_LX & !RDY_GATEX
    & !XFER_TYPE[1]_part1)
$ VCC
RDY_GATEX.C = CLKX

```

Output A3031_R[0]

```

10 Input(s)
    A3031_R[0], RST_LX, _DEF_357, _DEF_356, XFER_TYPE[1]_part1,
    _DEF_371, _DEF_367, CYC_CNT, XFER_TYPE[0], LHOLDAX
2 Fanout(s)
    glb00.I17, A30.IR
16 Product Term(s)
1 GLB Level(s)

A3031_R[0].D = (CYC_CNT & LHOLDAX & RST_LX & XFER_TYPE[1]_part1
    & _DEF_371 & !_DEF_367 & !XFER_TYPE[0] & !_DEF_356
    # CYC_CNT & LHOLDAX & RST_LX & XFER_TYPE[1]_part1 & _DEF_357
    & !_DEF_367 & !XFER_TYPE[0] & !_DEF_371
    # CYC_CNT & LHOLDAX & RST_LX & XFER_TYPE[1]_part1 & _DEF_356
    & !_DEF_357 & !XFER_TYPE[0]
    # A3031_R[0] & LHOLDAX & RST_LX & XFER_TYPE[1]_part1
    & _DEF_357 & _DEF_367 & !XFER_TYPE[0] & !_DEF_371
    # CYC_CNT & LHOLDAX & RST_LX & XFER_TYPE[1]_part1 & _DEF_367
    & !_DEF_357 & !XFER_TYPE[0]
    # A3031_R[0] & LHOLDAX & RST_LX & XFER_TYPE[1]_part1
    & !_DEF_357 & !_DEF_367 & !_DEF_356 & !_DEF_371
    # LHOLDAX & RST_LX & XFER_TYPE[0] & _DEF_357 & _DEF_367
    & !XFER_TYPE[1]_part1 & !CYC_CNT & !A3031_R[0]
    # A3031_R[0] & LHOLDAX & RST_LX & XFER_TYPE[0]
    & !XFER_TYPE[1]_part1 & !_DEF_371
    # A3031_R[0] & LHOLDAX & RST_LX & XFER_TYPE[1]_part1
    & _DEF_357 & _DEF_367 & !XFER_TYPE[0] & !_DEF_356
    # A3031_R[0] & LHOLDAX & RST_LX & XFER_TYPE[0]

```

```

& XFER_TYPE[1]_part1 & _DEF_356 & _DEF_371
# A3031_R[0] & LHOLDAX & RST_LX & XFER_TYPE[0]
& !XFER_TYPE[1]_part1 & !_DEF_356
# A3031_R[0] & CYC_CNT & LHOLDAX & RST_LX)
$ (A3031_R[0] & LHOLDAX & RST_LX & XFER_TYPE[1]_part1
& _DEF_356 & _DEF_371 & !CYC_CNT
# CYC_CNT & LHOLDAX & RST_LX & XFER_TYPE[1]_part1 & _DEF_356
& _DEF_371 & !_DEF_357 & !XFER_TYPE[0] & !A3031_R[0]
# LHOLDAX & RST_LX & XFER_TYPE[0] & _DEF_356 & _DEF_357
& _DEF_367 & _DEF_371 & !XFER_TYPE[1]_part1 & !CYC_CNT
# A3031_R[0] & LHOLDAX & RST_LX & XFER_TYPE[0] & !_DEF_357
& !_DEF_367 & !CYC_CNT & !_DEF_356 & !_DEF_371)
A3031_R[0].C = CLKX

```

GLB glb01, A7

```

12 Input(s)
  (glb01.02, A3031_R[1], I16), (glb03.00, CYC_CNT, I15),
  (glb06.03, LHOLDAX, I8), (RST_L.0, RST_LX, I5), (glb01.01,
  XFER_TYPE[0], I17), (glb03.03, XFER_TYPE[1]_part1, I12),
  (LBE_L[1].0, _DEF_356, I4), (LBE_L[3].0, _DEF_357, I13),
  (WR_RDN.0, _DEF_365, I14), (ADS_L.0, _DEF_366, I9),
  (LBE_L[2].0, _DEF_367, I10), (LBE_L[0].0, _DEF_371, I7)
3 Output(s)
  (_GND_1026, 00), (XFER_TYPE[0], 01), (A3031_R[1], 02)
20 Product Term(s)

```

Output _GND_1026

```

0 Input(s)
1 Fanout(s)
  BB_L.IR
0 Product Term(s)
0 GLB Level(s)

_GND_1026 = GND

```

Output XFER_TYPE[0]

```

9 Input(s)
  RST_LX, _DEF_357, _DEF_356, _DEF_371, _DEF_367, _DEF_366,
  _DEF_365, XFER_TYPE[0], LHOLDAX
5 Fanout(s)
  glb06.I13, glb02.I9, glb03.I6, glb00.I6, glb01.I17
5 Product Term(s)
1 GLB Level(s)

XFER_TYPE[0].D = (LHOLDAX & RST_LX & XFER_TYPE[0] & _DEF_366
# LHOLDAX & RST_LX & !_DEF_366 & !_DEF_365
# LHOLDAX & RST_LX & !_DEF_366 & _DEF_356 & _DEF_371
# LHOLDAX & RST_LX & !_DEF_366 & !_DEF_357 & !_DEF_367
& !_DEF_356 & !_DEF_371)
$ LHOLDAX & RST_LX & !_DEF_366 & _DEF_357 & _DEF_365
& _DEF_367
XFER_TYPE[0].C = CLKX

```

Output A3031_R[1]

```

10 Input(s)
  RST_LX, _DEF_357, _DEF_356, XFER_TYPE[1]_part1, _DEF_371,

```

```

    A3031_R[1], _DEF_367, CYC_CNT, XFER_TYPE[0], LholdAX
2 Fanout(s)
    glb01.I16, A31.IR
15 Product Term(s)
1 GLB Level(s)

A3031_R[1].D = (LholdAX & RST_LX & XFER_TYPE[1]_part1 & _DEF_356
& _DEF_357 & !_DEF_367 & !XFER_TYPE[0] & !_DEF_371
# A3031_R[1] & LholdAX & RST_LX & XFER_TYPE[1]_part1
& !_DEF_357 & !_DEF_367 & !XFER_TYPE[0] & !_DEF_356
& !_DEF_371
# A3031_R[1] & LholdAX & RST_LX & XFER_TYPE[0] & _DEF_371
& !_DEF_357 & !XFER_TYPE[1]_part1 & !_DEF_356
# A3031_R[1] & LholdAX & RST_LX & XFER_TYPE[0] & _DEF_367
& !_DEF_357 & !XFER_TYPE[1]_part1 & !_DEF_356
# A3031_R[1] & LholdAX & RST_LX & XFER_TYPE[1]_part1
& _DEF_357 & _DEF_367 & !XFER_TYPE[0]
# LholdAX & RST_LX & XFER_TYPE[0] & _DEF_356 & _DEF_357
& _DEF_367 & !XFER_TYPE[1]_part1 & !CYC_CNT & !_DEF_371
# LholdAX & RST_LX & XFER_TYPE[0] & _DEF_356 & _DEF_357
& _DEF_371 & !_DEF_367 & !XFER_TYPE[1]_part1 & !CYC_CNT
# A3031_R[1] & LholdAX & RST_LX & XFER_TYPE[0] & _DEF_356
& _DEF_357 & !XFER_TYPE[1]_part1
# LholdAX & RST_LX & XFER_TYPE[1]_part1 & _DEF_357
& !_DEF_367 & !XFER_TYPE[0] & !CYC_CNT & !_DEF_356
# CYC_CNT & LholdAX & RST_LX & XFER_TYPE[1]_part1 & _DEF_356
& !_DEF_357 & !XFER_TYPE[0] & !_DEF_371
# A3031_R[1] & LholdAX & RST_LX & XFER_TYPE[0] & _DEF_357
& !_DEF_367 & !XFER_TYPE[1]_part1
# A3031_R[1] & LholdAX & RST_LX & XFER_TYPE[0] & _DEF_356
& !XFER_TYPE[1]_part1 & !_DEF_371
# A3031_R[1] & LholdAX & RST_LX & XFER_TYPE[1]_part1
& _DEF_356 & _DEF_371 & !XFER_TYPE[0]
# A3031_R[1] & CYC_CNT & LholdAX & RST_LX & XFER_TYPE[0]
# A3031_R[1] & CYC_CNT & LholdAX & RST_LX
& !XFER_TYPE[1]_part1)
A3031_R[1].C = CLKX

```

GLB glb02, A3

```

13 Input(s)
    (glb03.00, CYC_CNT, I0), (Lhold.0, LholdX, I1), (glb06.03,
    LholdAX, I7), (RST_L.0, RST_LX, I10), (glb02.02,
    TSIZE_R[0], I14), (glb02.01, TS_DIS, I17), (glb02.00,
    N3452, I16), (glb01.01, XFER_TYPE[0], I9), (glb03.03,
    XFER_TYPE[1]_part1, I3), (LBE_L[1].0, _DEF_356, I11),
    (LBE_L[3].0, _DEF_357, I2), (LBE_L[2].0, _DEF_367, I5),
    (LBE_L[0].0, _DEF_371, I8)
4 Output(s)
    (TS_DIS, 01), (TSIZE_R[0], 02), (N3452, 00), (BR, 03)
18 Product Term(s)

```

Output TS_DIS

```

6 Input(s)
    RST_LX, XFER_TYPE[1]_part1, CYC_CNT, TS_DIS, XFER_TYPE[0],
    LholdAX
2 Fanout(s)
    glb06.I9, glb02.I17
2 Product Term(s)

```

1 GLB Level(s)

```
TS_DIS.D = (CYC_CNT & LHOLDAX & RST_LX & TS_DIS
# CYC_CNT & LHOLDAX & RST_LX & XFER_TYPE[1]_part1
& !XFER_TYPE[0])
TS_DIS.C = CLKX
```

Output TSIZE_R[0]

```
11 Input(s)
N3452, RST_LX, _DEF_357, _DEF_356, XFER_TYPE[1]_part1,
_DEF_371, _DEF_367, TSIZE_R[0], CYC_CNT, XFER_TYPE[0],
LHOLDAX
2 Fanout(s)
glb02.I14, TSIZE[0].IR
5 Product Term(s)
2 GLB Level(s)

TSIZE_R[0].D = (TSIZE_R[0] & N3452
# CYC_CNT & LHOLDAX & RST_LX & !_DEF_356 & !_DEF_371 & !N3452
# LHOLDAX & RST_LX & XFER_TYPE[1]_part1 & !_DEF_357
& !_DEF_367 & !XFER_TYPE[0] & !CYC_CNT & !N3452
# LHOLDAX & RST_LX & XFER_TYPE[0] & _DEF_357 & _DEF_367
& !XFER_TYPE[1]_part1 & !_DEF_356 & !_DEF_371 & !N3452
# LHOLDAX & RST_LX & XFER_TYPE[0] & _DEF_356 & _DEF_371
& !_DEF_357 & !_DEF_367 & !XFER_TYPE[1]_part1 & !CYC_CNT
& !N3452)
TSIZE_R[0].C = CLKX
```

Output N3452

```
9 Input(s)
RST_LX, _DEF_357, _DEF_356, XFER_TYPE[1]_part1, _DEF_371,
_DEF_367, CYC_CNT, XFER_TYPE[0], LHOLDAX
2 Fanout(s)
glb02.I16, glb03.I3
10 Product Term(s)
1 GLB Level(s)

N3452 = (LHOLDAX & RST_LX & XFER_TYPE[0] & _DEF_356 & !_DEF_367
& !XFER_TYPE[1]_part1 & !_DEF_371
# LHOLDAX & RST_LX & XFER_TYPE[0] & _DEF_367 & !_DEF_357
& !XFER_TYPE[1]_part1 & !_DEF_371
# LHOLDAX & RST_LX & XFER_TYPE[0] & _DEF_371 & !_DEF_357
& !XFER_TYPE[1]_part1 & !_DEF_356
# LHOLDAX & RST_LX & XFER_TYPE[0] & _DEF_357 & !_DEF_367
& !XFER_TYPE[1]_part1 & !_DEF_356
# LHOLDAX & RST_LX & XFER_TYPE[1]_part1 & !_DEF_357
& !_DEF_367 & !XFER_TYPE[0] & !_DEF_356 & !_DEF_371
# LHOLDAX & RST_LX & XFER_TYPE[0] & _DEF_356 & _DEF_357
& _DEF_367 & _DEF_371 & !XFER_TYPE[1]_part1
# LHOLDAX & RST_LX & XFER_TYPE[1]_part1 & _DEF_357
& _DEF_367 & !XFER_TYPE[0]
# LHOLDAX & RST_LX & XFER_TYPE[1]_part1 & _DEF_356
& _DEF_371 & !XFER_TYPE[0]
# CYC_CNT & LHOLDAX & RST_LX & XFER_TYPE[0]
# CYC_CNT & LHOLDAX & RST_LX & !XFER_TYPE[1]_part1)
```

Output BR

3 Input(s)

```

    RST_LX, LHOLDX, LHOLDAX
1 Fanout(s)
    BR_L.IR
1 Product Term(s)
1 GLB Level(s)

BR.D = LHOLDX & RST_LX & !LHOLDAX
BR.C = CLKX

```

GLB glb03, A4

```

14 Input(s)
    (glb03.00, CYC_CNT, I15), (glb06.03, LHOLDAX, I8), (RST_L.0,
    RST_LX, I5), (glb03.02, TSIZE_R[1], I17), (glb02.00,
    N3452, I3), (glb01.01, XFER_TYPE[0], I6), (glb03.01,
    XFER_TYPE[1]_part2, I16), (TA_L.0, _DEF_355, I12), (LBE_L[1].0,
    _DEF_356, I4), (LBE_L[3].0, _DEF_357, I13), (WR_RDN.0,
    _DEF_365, I14), (ADS_L.0, _DEF_366, I9), (LBE_L[2].0,
    _DEF_367, I10), (LBE_L[0].0, _DEF_371, I7)
4 Output(s)
    (XFER_TYPE[1]_part2, 01), (TSIZE_R[1], 02), (CYC_CNT, 00),
    (XFER_TYPE[1]_part1, 03)
19 Product Term(s)

```

Output XFER_TYPE[1]_part2

```

9 Input(s)
    RST_LX, _DEF_357, _DEF_356, XFER_TYPE[1]_part2, _DEF_371,
    _DEF_367, _DEF_366, _DEF_365, LHOLDAX
2 Fanout(s)
    glb06.I5, glb03.I16
6 Product Term(s)
1 GLB Level(s)

XFER_TYPE[1]_part2.D = (LHOLDAX & RST_LX & XFER_TYPE[1]_part2
    & _DEF_366
    # LHOLDAX & RST_LX & !_DEF_366 & !_DEF_365
    # LHOLDAX & RST_LX & !_DEF_366 & _DEF_357 & !_DEF_367
    & !_DEF_356
    # LHOLDAX & RST_LX & !_DEF_366 & _DEF_371 & !_DEF_357
    & !_DEF_356
    # LHOLDAX & RST_LX & !_DEF_366 & _DEF_367 & !_DEF_357
    & !_DEF_371
    # LHOLDAX & RST_LX & !_DEF_366 & _DEF_356 & !_DEF_367
    & !_DEF_371)
XFER_TYPE[1]_part2.C = CLKX

```

Output TSIZE_R[1]

```

11 Input(s)
    N3452, TSIZE_R[1], RST_LX, _DEF_357, _DEF_356,
    XFER_TYPE[1]_part2, _DEF_371, _DEF_367, CYC_CNT,
    XFER_TYPE[0], LHOLDAX
2 Fanout(s)
    glb03.I17, TSIZE[1].IR
9 Product Term(s)
2 GLB Level(s)

TSIZE_R[1].D = (LHOLDAX & RST_LX & XFER_TYPE[1]_part2 & _DEF_357
    & !_DEF_367 & !XFER_TYPE[0] & !CYC_CNT & !N3452

```

```

# L_HOLDAX & RST_LX & XFER_TYPE[1]_part2 & _DEF_367
& !_DEF_357 & !XFER_TYPE[0] & !CYC_CNT & !N3452
# L_HOLDAX & RST_LX & XFER_TYPE[0] & _DEF_356
& !XFER_TYPE[1]_part2 & !_DEF_371 & !N3452
# L_HOLDAX & RST_LX & XFER_TYPE[0] & _DEF_371
& !XFER_TYPE[1]_part2 & !_DEF_356 & !N3452
# L_HOLDAX & RST_LX & XFER_TYPE[0] & _DEF_357 & !_DEF_367
& !XFER_TYPE[1]_part2 & !CYC_CNT & !N3452
# L_HOLDAX & RST_LX & XFER_TYPE[0] & _DEF_367 & !_DEF_357
& !XFER_TYPE[1]_part2 & !CYC_CNT & !N3452
# CYC_CNT & L_HOLDAX & RST_LX & _DEF_356 & !_DEF_371 & !N3452
# CYC_CNT & L_HOLDAX & RST_LX & _DEF_371 & !_DEF_356 & !N3452
# TSIZE_R[1] & N3452)
TSIZE_R[1].C = CLKX

```

Output CYC_CNT

```

6 Input(s)
  RST_LX, XFER_TYPE[1]_part2, _DEF_355, CYC_CNT, XFER_TYPE[0],
  L_HOLDAX
5 Fanout(s)
  glb06.I0, glb02.I0, glb03.I15, glb00.I15, glb01.I15
4 Product Term(s)
1 GLB Level(s)

CYC_CNT.D = (CYC_CNT & L_HOLDAX & RST_LX & !XFER_TYPE[1]_part2
# CYC_CNT & L_HOLDAX & RST_LX & XFER_TYPE[0]
# CYC_CNT & L_HOLDAX & RST_LX & _DEF_355
# L_HOLDAX & RST_LX & XFER_TYPE[1]_part2 & !XFER_TYPE[0]
& !CYC_CNT & !_DEF_355)
CYC_CNT.C = CLKX

```

Output XFER_TYPE[1]_part1

```

9 Input(s)
  RST_LX, _DEF_357, _DEF_356, XFER_TYPE[1]_part2, _DEF_371,
  _DEF_367, _DEF_366, _DEF_365, L_HOLDAX
3 Fanout(s)
  glb02.I3, glb00.I12, glb01.I12
6 Product Term(s)
1 GLB Level(s)

XFER_TYPE[1]_part1.D = (L_HOLDAX & RST_LX & XFER_TYPE[1]_part2
& _DEF_366
# L_HOLDAX & RST_LX & !_DEF_366 & !_DEF_365
# L_HOLDAX & RST_LX & !_DEF_366 & _DEF_357 & !_DEF_367
& !_DEF_356
# L_HOLDAX & RST_LX & !_DEF_366 & _DEF_371 & !_DEF_357
& !_DEF_356
# L_HOLDAX & RST_LX & !_DEF_366 & _DEF_367 & !_DEF_357
& !_DEF_371
# L_HOLDAX & RST_LX & !_DEF_366 & _DEF_356 & !_DEF_367
& !_DEF_371)
XFER_TYPE[1]_part1.C = CLKX

```

GLB glb04, A2

```

18 Input(s)
  (A28.0, A28X, I17), (A29.0, A29X, I15), (glb06.02,
  ADS_PL, I6), (glb04.02, BLAST_R, I1), (BTERM_EN.0,

```

```

    BTERM_ENX, I0), (glb04.01, BTERM_R, I2), (glb04.00,
    BURST_R, I16), (glb06.03, L_HOLDAX, I7), (READY0_L.0,
    READY0_LX, I9), (RST_L.0, RST_LX, I10), (TA_L.0,
    _DEF_355, I3), (TS_L.0, _DEF_358, I4), (BI_L.0, _DEF_359, I11),
    (A30.0, _DEF_360, I5), (BLAST_L.0, _DEF_361, I13), (BB_L.0,
    _DEF_368, I8), (BURST_L.0, _DEF_369, I12), (A31.0,
    _DEF_370, I14)
4 Output(s)
    (BURST_R, 00), (BTERM_R, 01), (BLAST_R, 02), (BI_R, 03)
13 Product Term(s)

```

Output BURST_R

```

7 Input(s)
    RST_LX, _DEF_361, _DEF_355, BTERM_R, BURST_R, L_HOLDAX,
    ADS_PL
2 Fanout(s)
    glb04.I16, BURST_L.IR
3 Product Term(s)
1 GLB Level(s)

BURST_R.D = (BURST_R & L_HOLDAX & RST_LX & _DEF_355
    # ADS_PL & L_HOLDAX & RST_LX & _DEF_361
    # BURST_R & L_HOLDAX & RST_LX & !BTERM_R & _DEF_361)
BURST_R.C = CLKX

```

Output BTERM_R

```

9 Input(s)
    RST_LX, _DEF_359, _DEF_361, A29X, BTERM_ENX, _DEF_355,
    BTERM_R, A28X, L_HOLDAX
2 Fanout(s)
    glb04.I2, BTERM_L.IR
3 Product Term(s)
1 GLB Level(s)

BTERM_R.D = (L_HOLDAX & RST_LX & !_DEF_359
    # BTERM_R & L_HOLDAX & RST_LX & _DEF_355
    # A28X & BTERM_ENX & L_HOLDAX & RST_LX & _DEF_361 & !A29X
    & !_DEF_355)
BTERM_R.C = CLKX

```

Output BLAST_R

```

9 Input(s)
    RST_LX, _DEF_358, A29X, _DEF_369, BLAST_R, _DEF_368,
    READY0_LX, A28X, L_HOLDAX
2 Fanout(s)
    glb04.I1, BLAST_L.IR
3 Product Term(s)
1 GLB Level(s)

BLAST_R.D = (RST_LX & _DEF_369 & !L_HOLDAX & !_DEF_368 & !_DEF_358
    # BLAST_R & READY0_LX & RST_LX & _DEF_358 & !L_HOLDAX
    & !_DEF_368
    # A28X & RST_LX & _DEF_358 & !L_HOLDAX & !A29X & !_DEF_368
    & !_DEF_369)
BLAST_R.C = CLKX

```

Output BI_R

```

7 Input(s)
  RST_LX, _DEF_358, _DEF_360, A29X, _DEF_369, _DEF_370, A28X
1 Fanout(s)
  BI_L.IR
4 Product Term(s)
1 GLB Level(s)

BI_R.D = (A28X & RST_LX & !_DEF_358 & !_DEF_369
# A29X & RST_LX & !_DEF_358 & !_DEF_369
# RST_LX & _DEF_360 & !_DEF_358 & !_DEF_369
# RST_LX & _DEF_370 & !_DEF_358 & !_DEF_369)
BI_R.C = CLKX

```

GLB glb05, A6

```

7 Input(s)
  (glb06.03, LholdAX, I8), (A30.0, _DEF_360, I10), (TSIZE[1].0,
  _DEF_362, I9), (TSIZE[0].0, _DEF_363, I5), (WR_RDN.0,
  _DEF_365, I14), (BB_L.0, _DEF_368, I7), (A31.0, _DEF_370, I1)
4 Output(s)
  (N3552, 00), (N3549, 01), (N3542, 02), (N3541, 03)
1 Enable Output
  N3547
13 Product Term(s)

```

Output N3552

```

5 Input(s)
  _DEF_363, _DEF_362, _DEF_360, _DEF_370, _DEF_365
1 Fanout(s)
  LBE_L[0].IR
3 Product Term(s)
1 GLB Level(s)

N3552 = (!_DEF_363 & _DEF_362 & _DEF_365 & !_DEF_360
# !_DEF_363 & _DEF_362 & _DEF_365 & !_DEF_370
# !_DEF_362 & _DEF_363 & _DEF_365 & !_DEF_360 & !_DEF_370)

```

Output N3549

```

5 Input(s)
  _DEF_363, _DEF_362, _DEF_360, _DEF_370, _DEF_365
1 Fanout(s)
  LBE_L[2].IR
3 Product Term(s)
1 GLB Level(s)

N3549 = (!_DEF_363 & _DEF_360 & _DEF_362 & _DEF_365
# !_DEF_363 & _DEF_362 & _DEF_365 & !_DEF_370
# !_DEF_362 & _DEF_360 & _DEF_363 & _DEF_365 & !_DEF_370)

```

Output N3542

```

5 Input(s)
  _DEF_363, _DEF_362, _DEF_360, _DEF_370, _DEF_365
1 Fanout(s)
  LBE_L[3].IR
3 Product Term(s)
1 GLB Level(s)

```

```

N3542 = (!_DEF_362 & _DEF_360 & _DEF_363 & _DEF_365 & !_DEF_370
# !_DEF_363 & _DEF_362 & _DEF_365 & _DEF_370
# !_DEF_363 & _DEF_360 & _DEF_362 & _DEF_365)

```

Output N3541

```

5 Input(s)
  _DEF_363, _DEF_362, _DEF_360, _DEF_370, _DEF_365
1 Fanout(s)
  LBE_L[1].IR
3 Product Term(s)
1 GLB Level(s)

N3541 = (!_DEF_363 & _DEF_362 & _DEF_365 & !_DEF_360
# !_DEF_362 & _DEF_363 & _DEF_365 & !_DEF_360 & !_DEF_370
# !_DEF_363 & _DEF_362 & _DEF_365 & _DEF_370)

```

Enable Output N3547

```

2 Input(s)
  LHOLDAX, _DEF_368
9 Fanout(s)
  LBE_L[0].OE0, LBE_L[2].OE0, ADS_L.OE0, WR_RDN.OE0,
  BLAST_L.OE0, BI_L.OE0, LBE_L[3].OE0, LBE_L[1].OE0, TA_L.OE0
1 Product Term(s)
1 GLB Level(s)

N3547 = !LHOLDAX & !_DEF_368

```

GLB glb06, A1

```

14 Input(s)
  (glb06.02, ADS_PL, I17), (BG_L.0, BG_LX, I2), (glb03.00,
  CYC_CNT, I0), (LHOLD.0, LHOLDX, I1), (glb06.03, LHOLDAX, I7),
  (glb00.00, RDY_GATEX, I4), (RST_L.0, RST_LX, I10), (glb02.01,
  TS_DIS, I9), (glb06.00, TS_RX, I16), (glb01.01,
  XFER_TYPE[0], I13), (glb03.01, XFER_TYPE[1]_part2, I5),
  (TA_L.0, _DEF_355, I3), (ADS_L.0, _DEF_366, I6), (BB_L.0,
  _DEF_368, I8)
4 Output(s)
  (TS_RX, 00), (READYI_LX, 01), (LHOLDAX, 03), (ADS_PL, 02)
11 Product Term(s)

```

Output TS_RX

```

8 Input(s)
  RST_LX, XFER_TYPE[1]_part2, TS_RX, CYC_CNT, TS_DIS,
  XFER_TYPE[0], LHOLDAX, ADS_PL
3 Fanout(s)
  glb06.I16, TS_L.IR, TS_R.IR
5 Product Term(s)
1 GLB Level(s)

TS_RX.D = (ADS_PL & LHOLDAX & RST_LX & XFER_TYPE[1]_part2
& !CYC_CNT
# CYC_CNT & LHOLDAX & RST_LX & TS_RX & !XFER_TYPE[1]_part2
# CYC_CNT & LHOLDAX & RST_LX & TS_RX & XFER_TYPE[0]
# ADS_PL & LHOLDAX & RST_LX & XFER_TYPE[0] & !CYC_CNT
# CYC_CNT & LHOLDAX & RST_LX & XFER_TYPE[1]_part2 & !TS_DIS
& !XFER_TYPE[0])

```

```
TS_RX.C = CLKX
```

Output READYI_LX

```
3 Input(s)
  _DEF_355, RDY_GATEX, LholdAX
1 Fanout(s)
  READYI_L.ID
3 Product Term(s)
1 GLB Level(s)
```

```
READYI_LX = !RDY_GATEX
# !LholdAX
# _DEF_355
```

Output LholdAX

```
5 Input(s)
  RST_LX, _DEF_368, LholdX, BG_LX, LholdAX
8 Fanout(s)
  glb06.I7, glb04.I7, glb02.I7, glb03.I8, glb00.I8, glb05.I8,
  glb01.I8, LholdA.IR
2 Product Term(s)
1 GLB Level(s)
```

```
LholdAX.D = (LholdX & LholdAX & RST_LX
# RST_LX & _DEF_368 & !BG_LX)
LholdAX.C = CLKX
```

Output ADS_PL

```
1 Input(s)
  _DEF_366
2 Fanout(s)
  glb06.I17, glb04.I6
1 Product Term(s)
1 GLB Level(s)
```

```
ADS_PL.D = !_DEF_366
ADS_PL.C = CLKX
```

GLB glb07, A0

```
4 Input(s)
  (READYO_L.0, READYO_LX, I9), (TS_L.0, _DEF_358, I0), (RD_WRN.0,
  _DEF_364, I4), (WR_RDN.0, _DEF_365, I1)
4 Output(s)
  (_BUF_903, 00), (_BUF_901, 01), (_BUF_900, 02), (_BUF_899, 03)
4 Product Term(s)
```

Output _BUF_903

```
1 Input(s)
  _DEF_365
1 Fanout(s)
  RD_WRN.IR
1 Product Term(s)
1 GLB Level(s)
```

```
_BUF_903 = _DEF_365
```

Output _BUF_901

```

1 Input(s)
  _DEF_364
1 Fanout(s)
  WR_RDN.ID
1 Product Term(s)
1 GLB Level(s)

_BUF_901 = _DEF_364

```

Output _BUF_900

```

1 Input(s)
  _DEF_358
1 Fanout(s)
  ADS_L.ID
1 Product Term(s)
1 GLB Level(s)

_BUF_900 = _DEF_358

```

Output _BUF_899

```

1 Input(s)
  READY0_LX
1 Fanout(s)
  TA_L.ID
1 Product Term(s)
1 GLB Level(s)

_BUF_899 = READY0_LX

```

Dedicated Input A28, I1

```

Output A28X
1 Fanout(s)
  glb04.I17

```

Input A29, I020

```

Output A29X
1 Fanout(s)
  glb04.I15

```

Bidirectional A30, I021

```

Input (glb00.01, A3031_R[0])
Output _DEF_360
2 Fanout(s)
  glb04.I5, glb05.I10
Global Enable (LHOLDA_FB.0, LHOLDA_FBX)

A30 = A3031_R[0]
A30.E = LHOLDA_FBX

```

Bidirectional A31, IO30

```

Input (glb01.02, A3031_R[1])
Output _DEF_370
    2 Fanout(s)
        glb04.I14, glb05.I1
Global Enable (LHOLDA_FB.0, LHOLDA_FBX)

A31 = A3031_R[1]
A31.E = LHOLDA_FBX

```

Bidirectional ADS_L, IO2

```

Input (glb07.02, _BUF_900)
Output _DEF_366
    3 Fanout(s)
        glb06.I6, glb03.I9, glb01.I9
Enable (glb05.0E, N3547)

ADS_L = _BUF_900
ADS_L.E = N3547

```

Bidirectional BB_L, IO28

```

Input (glb01.00, _GND_1026)
Output _DEF_368
    3 Fanout(s)
        glb06.I8, glb04.I8, glb05.I7
Global Enable (LHOLDA_FB.0, LHOLDA_FBX)

BB_L = _GND_1026
BB_L.E = LHOLDA_FBX

```

Input BG_L, IO25

```

Output BG_LX
    1 Fanout(s)
        glb06.I2

```

Bidirectional BI_L, IO15

```

Input (glb04.03, BI_R)
Output _DEF_359
    1 Fanout(s)
        glb04.I11
Enable (glb05.0E, N3547)

BI_L = !BI_R
BI_L.E = N3547

```

Bidirectional BLAST_L, IO6

```

Input (glb04.02, BLAST_R)
Output _DEF_361
    1 Fanout(s)
        glb04.I13

```

Enable (glb05.0E, N3547)

BLAST_L = !BLAST_R
BLAST_L.E = N3547

Output BR_L, I011

Input (glb02.03, BR)

BR_L = !BR

Input BTERM_EN, I027

Output BTERM_ENX
1 Fanout(s)
glb04.I0

Three-State Output BTERM_L, I09

Input (glb04.01, BTERM_R)
Global Enable (LHOLDA_FB.0, LHOLDA_FBX)

BTERM_L = !BTERM_R
BTERM_L.E = LHOLDA_FBX

Bidirectional BURST_L, I012

Input (glb04.00, BURST_R)
Output _DEF_369
1 Fanout(s)
glb04.I12
Global Enable (LHOLDA_FB.0, LHOLDA_FBX)

BURST_L = !BURST_R
BURST_L.E = LHOLDA_FBX

Clock Input CLK, Y0

Output CLKX
6 Fanout(s)
glb06.CLK0, glb04.CLK0, glb02.CLK0, glb03.CLK0, glb00.CLK0,
glb01.CLK0

Bidirectional LBE_L[0], I024

Input (glb05.00, N3552)
Output _DEF_371
4 Fanout(s)
glb02.I8, glb03.I7, glb00.I7, glb01.I7
Enable (glb05.0E, N3547)

LBE_L[0] = N3552
LBE_L[0].E = N3547

Bidirectional LBE_L[1], IO31

```

Input (glb05.03, N3541)
Output _DEF_356
    4 Fanout(s)
        glb02.I11, glb03.I4, glb00.I4, glb01.I4
Enable (glb05.0E, N3547)

LBE_L[1] = N3541
LBE_L[1].E = N3547

```

Bidirectional LBE_L[2], IO17

```

Input (glb05.01, N3549)
Output _DEF_367
    4 Fanout(s)
        glb02.I5, glb03.I10, glb00.I10, glb01.I10
Enable (glb05.0E, N3547)

LBE_L[2] = N3549
LBE_L[2].E = N3547

```

Bidirectional LBE_L[3], IO18

```

Input (glb05.02, N3542)
Output _DEF_357
    4 Fanout(s)
        glb02.I2, glb03.I13, glb00.I13, glb01.I13
Enable (glb05.0E, N3547)

LBE_L[3] = N3542
LBE_L[3].E = N3547

```

Input LHOLD, IO26

```

Output LHOLDX
    2 Fanout(s)
        glb06.I1, glb02.I1

```

Output LHOLDA, IO7

```

Input (glb06.03, LHOLDAX)

LHOLDA = LHOLDAX

```

Global Enable Input LHOLDA_FB, GOE0

```

Output LHOLDA_FBX
    9 Fanout(s)
        A31.GOE0, BURST_L.GOE0, BB_L.GOE0, RD_WRN.GOE0,
        TSIZE[0].GOE0, TSIZE[1].GOE0, A30.GOE0, TS_L.GOE0,
        BTERM_L.GOE0

```

Output RDY_GATE, IO16

```
Input (glb00.00, RDY_GATEX)
```

```
RDY_GATE = RDY_GATEX
```

```
Bidirectional RD_WRN, IO4
```

```
Input (glb07.00, _BUF_903)
```

```
Output _DEF_364
```

```
1 Fanout(s)
```

```
glb07.I4
```

```
Global Enable (LHOLDA_FB.0, LHOLDA_FBX)
```

```
RD_WRN = !_BUF_903
```

```
RD_WRN.E = LHOLDA_FBX
```

```
Output READYI_L, IO5
```

```
Input (glb06.01, READYI_LX)
```

```
READYI_L = READYI_LX
```

```
Input READYO_L, IO13
```

```
Output READYO_LX
```

```
2 Fanout(s)
```

```
glb07.I9, glb04.I9
```

```
Input RST_L, IO10
```

```
Output RST_LX
```

```
6 Fanout(s)
```

```
glb06.I10, glb04.I10, glb02.I10, glb03.I5, glb00.I14,  
glb01.I5
```

```
Bidirectional TA_L, IO3
```

```
Input (glb07.03, _BUF_899)
```

```
Output _DEF_355
```

```
3 Fanout(s)
```

```
glb06.I3, glb04.I3, glb03.I12
```

```
Enable (glb05.0E, N3547)
```

```
TA_L = _BUF_899
```

```
TA_L.E = N3547
```

```
Bidirectional TSIZE[0], IO14
```

```
Input (glb02.02, TSIZE_R[0])
```

```
Output _DEF_363
```

```
1 Fanout(s)
```

```
glb05.I5
```

```
Global Enable (LHOLDA_FB.0, LHOLDA_FBX)
```

```
TSIZE[0] = TSIZE_R[0]
```

```
TSIZE[0].E = LHOLDA_FBX
```

Bidirectional TSIZE[1], IO22

```

Input (glb03.02, TSIZE_R[1])
Output _DEF_362
    1 Fanout(s)
        glb05.I9
Global Enable (LHOLDA_FB.0, LHOLDA_FBX)

TSIZE[1] = TSIZE_R[1]
TSIZE[1].E = LHOLDA_FBX

```

Bidirectional TS_L, IO0

```

Input (glb06.00, TS_RX)
Output _DEF_358
    2 Fanout(s)
        glb07.I0, glb04.I4
Global Enable (LHOLDA_FB.0, LHOLDA_FBX)

TS_L = !TS_RX
TS_L.E = LHOLDA_FBX

```

Output TS_R, IO8

```

Input (glb06.00, TS_RX)

TS_R = TS_RX

```

Bidirectional WR_RDN, IO1

```

Input (glb07.01, _BUF_901)
Output _DEF_365
    4 Fanout(s)
        glb07.I1, glb03.I14, glb05.I14, glb01.I14
Enable (glb05.0E, N3547)

WR_RDN = !_BUF_901
WR_RDN.E = N3547

```

Clock Assignments

Net Name	Clock Assignment
CLKX	External CLK0

GLB and GLB Output Statistics

GLB Name, Location GLB Output Name	GLB Statistics Ins, Outs, PTs	GLB Output Statistics Ins, F0s, PTs, Levels
glb00, A5 A3031_R[0] RDY_GATEX	11, 2, 19	10, 2, 16, 1 6, 3, 4, 1

glb01, A7	12, 3, 20		
A3031_R[1]		10, 2, 15, 1	
XFER_TYPE[0]		9, 5, 5, 1	
_GND_1026		0, 1, 0, 0	
glb02, A3	13, 4, 18		
BR		3, 1, 1, 1	
N3452		9, 2, 10, 1	
TSIZE_R[0]		11, 2, 5, 2	
TS_DIS		6, 2, 2, 1	
glb03, A4	14, 4, 19		
CYC_CNT		6, 5, 4, 1	
TSIZE_R[1]		11, 2, 9, 2	
XFER_TYPE[1]_part1		9, 3, 6, 1	
XFER_TYPE[1]_part2		9, 2, 6, 1	
glb04, A2	18, 4, 13		
BI_R		7, 1, 4, 1	
BLAST_R		9, 2, 3, 1	
BTERM_R		9, 2, 3, 1	
BURST_R		7, 2, 3, 1	
glb05, A6	7, 4, 13		
N3541		5, 1, 3, 1	
N3542		5, 1, 3, 1	
N3549		5, 1, 3, 1	
N3552		5, 1, 3, 1	
glb06, A1	14, 4, 11		
ADS_PL		1, 2, 1, 1	
LHOLDAX		5, 8, 2, 1	
READYI_LX		3, 1, 3, 1	
TS_RX		8, 3, 5, 1	
glb07, A0	4, 4, 4		
_BUF_899		1, 1, 1, 1	
_BUF_900		1, 1, 1, 1	
_BUF_901		1, 1, 1, 1	
_BUF_903		1, 1, 1, 1	

Maximum-Level Trace

GLB Level, Name, Ins	GLB Output Name
2, glb02, 10	TSIZE_R[0]
1, glb02	N3452
2, glb03, 11	TSIZE_R[1]
1, glb02	N3452

Pin Assignments

Pin Name	Pin Assignment	Pin Type, Pin Attribute
LHOLDA_FB	2	Global Enable Input, PULLUP
LBE_L[0]	3	Bidirectional, PULLUP
BG_L	4	Input, PULLUP
LHOLD	5	Input, PULLUP

BTERM_EN	6	Input, PULLUP
BB_L	7	Bidirectional, PULLUP
A31	9	Bidirectional, PULLUP
LBE_L[1]	10	Bidirectional, PULLUP
CLK	11	Clock Input, PULLUP
TS_L	15	Bidirectional, PULLUP
WR_RDN	16	Bidirectional, CRIT, PULLUP
ADS_L	17	Bidirectional, CRIT, PULLUP
TA_L	18	Bidirectional, CRIT, PULLUP
RD_WRN	19	Bidirectional, PULLUP
READYI_L	20	Output, CRIT, PULLUP
BLAST_L	21	Bidirectional, PULLUP
LHOLDA	22	Output, PULLUP
A28	24	Dedicated Input, PULLUP
TS_R	25	Output, PULLUP
BTERM_L	26	Three-State Output, PULLUP
RST_L	27	Input, PULLUP
BR_L	28	Output, PULLUP
BURST_L	29	Bidirectional, PULLUP
READY0_L	30	Input, PULLUP
TSIZE[0]	31	Bidirectional, PULLUP
BI_L	32	Bidirectional, PULLUP
RDY_GATE	37	Output, PULLUP
LBE_L[2]	38	Bidirectional, PULLUP
LBE_L[3]	39	Bidirectional, PULLUP
A29	41	Input, PULLUP
A30	42	Bidirectional, PULLUP
TSIZE[1]	43	Bidirectional, PULLUP

Design process management completed successfully

pLSI2032 TIMING REPORT FILE

Timing Analysis - Short Report

Maximum Operating Frequency: 72 MHz

The following path determines the frequency:

Startpoint: GLB_XFER_TYPE%0%/Q0
(edge-triggered flip-flop)
Endpoint: GLB_TSIZE_R%0%/D0
(edge-triggered flip-flop)

Point Name/pin_name [type]	Delay	Path
GLB_XFER_TYPE%0%/Q0 [PGDFFR]	0.00	0.00
GRP_XFER_TYPE%0%_grp/A0 [PRBUF04]	0.00	0.00
GRP_XFER_TYPE%0%_grp/Z0 [PRBUF04]	1.30	1.30
GLB_A3_IN9/A0 [PGBUFI]	0.00	1.30
GLB_A3_IN9/Z0 [PGBUFI]	0.40	1.70
GLB_A3_P19/A2 [PGANDD8]	0.00	1.70
GLB_A3_P19/Z0 [PGANDD8]	1.10	2.80
GLB_A3_F5/A2 [PGORF77]	0.00	2.80
GLB_A3_F5/Z0 [PGORF77]	1.90	4.70
GLB_A3_G3/A0 [PGORG2]	0.00	4.70
GLB_A3_G3/Z0 [PGORG2]	1.00	5.70
GLB_A3_X00/A0 [PGXOR2]	0.00	5.70
GLB_A3_X00/Z0 [PGXOR2]	0.70	6.40
GLB_N3452/A0 [PGBUFX0]	0.00	6.40
GLB_N3452/Z0 [PGBUFX0]	0.00	6.40
GRP_N3452_ffb/A0 [PGBUFF]	0.00	6.40
GRP_N3452_ffb/Z0 [PGBUFF]	1.40	7.80
GLB_A3_IN16B/A0 [PGINVI]	0.00	7.80
GLB_A3_IN16B/ZN0 [PGINVI]	0.40	8.20
GLB_A3_P8/A0 [PGANDD10]	0.00	8.20
GLB_A3_P8/Z0 [PGANDD10]	1.10	9.30
GLB_A3_F4/A4 [PGORF55]	0.00	9.30
GLB_A3_F4/Z0 [PGORF55]	1.90	11.20
GLB_A3_G1/A0 [PGORG1]	0.00	11.20
GLB_A3_G1/Z0 [PGORG1]	1.00	12.20
GLB_TSIZE_R%0%_D0/A0 [PGXOR2]	0.00	12.20
GLB_TSIZE_R%0%_D0/Z0 [PGXOR2]	0.70	12.90
GLB_TSIZE_R%0%/D0 [PGDFFR]	0.00	12.90

The clock period is 13.90.

clock period = path delay + clock-to-output delay + setup time

path delay: 12.90
clock-to-output delay: 0.70
setup time: 0.30

Information for flip-flop:

Global reset-to-output delay: 1.10
Clock-to-output delay: 0.70

User reset-to-output delay: 1.10
 Data-to-output delay: 0.00
 Setup time: 0.30
 Hold time: 3.00
 Pulse-width time: 3.00

Longest Paths:

Startpoint Name/pin_name [type]	Endpoint Name/pin_name [type]	Path Delay

LBE_L[1] [bidi]	GLB_TSIZE_R%0%/D0 [PGDFFR]	14.00
LBE_L[1] [bidi]	GLB_TSIZE_R%1%/D0 [PGDFFR]	13.90
BB_L [bidi]	ADS_L [bidi]	12.00
BB_L [bidi]	BI_L [bidi]	12.00
BB_L [bidi]	BLAST_L [bidi]	12.00
BB_L [bidi]	LBE_L[0] [bidi]	12.00
BB_L [bidi]	LBE_L[1] [bidi]	12.00
BB_L [bidi]	LBE_L[2] [bidi]	12.00
BB_L [bidi]	LBE_L[3] [bidi]	12.00
BB_L [bidi]	TA_L [bidi]	12.00
BB_L [bidi]	WR_RDN [bidi]	12.00
WR_RDN [bidi]	RD_WRN [bidi]	9.90
RST_L [in]	GLB_A3031_R%0%/D0 [PGDFFR]	8.00
RST_L [in]	GLB_A3031_R%1%/D0 [PGDFFR]	7.50
A31 [bidi]	GLB_BI_R/D0 [PGDFFR]	7.50
BI_L [bidi]	GLB_BTERM_R/D0 [PGDFFR]	7.50
TA_L [bidi]	GLB_CYC_CNT/D0 [PGDFFR]	7.50
RST_L [in]	GLB_TS_RX/D0 [PGDFFR]	7.50
ADS_L [bidi]	GLB_XFER_TYPE%0%/D0 [PGDFFR]	7.50
LBE_L[3] [bidi]	GLB_X...part1 *1/D0 [PGDFFR]	7.50
LBE_L[3] [bidi]	GLB_X...part2 *2/D0 [PGDFFR]	7.50
ADS_L [bidi]	GLB_ADS_PL/D0 [PGDFFR]	7.40
RST_L [in]	GLB_BR/D0 [PGDFFR]	7.40
BURST_L [bidi]	GLB_BLAST_R/D0 [PGDFFR]	7.30
RST_L [in]	GLB_BURST_R/D0 [PGDFFR]	7.30
RST_L [in]	GLB_LHOLDAX/D0 [PGDFFR]	7.30
RST_L [in]	GLB_RDY_GATEX/D0 [PGDFFR]	7.30
RST_L [in]	GLB_TS_DIS/D0 [PGDFFR]	7.30
TA_L [bidi]	READYI_L [out]	7.30
LHOLDA_FB [in]	A30 [bidi]	6.00
LHOLDA_FB [in]	A31 [bidi]	6.00
LHOLDA_FB [in]	BB_L [bidi]	6.00
LHOLDA_FB [in]	BTERM_L [out]	6.00
LHOLDA_FB [in]	BURST_L [bidi]	6.00
LHOLDA_FB [in]	TSIZE[0] [bidi]	6.00
LHOLDA_FB [in]	TSIZE[1] [bidi]	6.00
LHOLDA_FB [in]	TS_L [bidi]	6.00
GLB_BR/Q0 [PGDFFR]	BR_L [out]	2.50
GLB_LHOLDAX/Q0 [PGDFFR]	LHOLDA [out]	2.50
GLB_RDY_GATEX/Q0 [PGDFFR]	RDY_GATE [out]	2.50
GLB_TS_RX/Q0 [PGDFFR]	TS_R [out]	2.50
CLK [in]	GLB_A3031_R%0%/CLK [PGDFFR]	2.30
CLK [in]	GLB_A3031_R%1%/CLK [PGDFFR]	2.30
CLK [in]	GLB_ADS_PL/CLK [PGDFFR]	2.30
CLK [in]	GLB_BI_R/CLK [PGDFFR]	2.30
CLK [in]	GLB_BLAST_R/CLK [PGDFFR]	2.30
CLK [in]	GLB_BR/CLK [PGDFFR]	2.30
CLK [in]	GLB_BTERM_R/CLK [PGDFFR]	2.30
CLK [in]	GLB_BURST_R/CLK [PGDFFR]	2.30

CLK [in]	GLB_CYC_CNT/CLK [PGDFFR]	2.30
CLK [in]	GLB_LHOLDAX/CLK [PGDFFR]	2.30
CLK [in]	GLB_RDY_GATEX/CLK [PGDFFR]	2.30
CLK [in]	GLB_TSIZE_R%0%/CLK [PGDFFR]	2.30
CLK [in]	GLB_TSIZE_R%1%/CLK [PGDFFR]	2.30
CLK [in]	GLB_TS_DIS/CLK [PGDFFR]	2.30
CLK [in]	GLB_TS_RX/CLK [PGDFFR]	2.30
CLK [in]	GLB_XFER_TYPE%0%/CLK [PGDFFR]	2.30
CLK [in]	GLB_X...part1 *1/CLK [PGDFFR]	2.30
CLK [in]	GLB_X...part2 *2/CLK [PGDFFR]	2.30

Index Name Table

- *1 GLB_XFER_TYPE%1%_part1
- *2 GLB_XFER_TYPE%1%_part2

Shortest Paths:

Startpoint Name/pin_name [type]	Endpoint Name/pin_name [type]	Path Delay
CLK [in]	GLB_A3031_R%0%/CLK [PGDFFR]	2.30
CLK [in]	GLB_A3031_R%1%/CLK [PGDFFR]	2.30
CLK [in]	GLB_ADS_PL/CLK [PGDFFR]	2.30
CLK [in]	GLB_BI_R/CLK [PGDFFR]	2.30
CLK [in]	GLB_BLAST_R/CLK [PGDFFR]	2.30
CLK [in]	GLB_BR/CLK [PGDFFR]	2.30
CLK [in]	GLB_BTERM_R/CLK [PGDFFR]	2.30
CLK [in]	GLB_BURST_R/CLK [PGDFFR]	2.30
CLK [in]	GLB_CYC_CNT/CLK [PGDFFR]	2.30
CLK [in]	GLB_LHOLDAX/CLK [PGDFFR]	2.30
CLK [in]	GLB_RDY_GATEX/CLK [PGDFFR]	2.30
CLK [in]	GLB_TSIZE_R%0%/CLK [PGDFFR]	2.30
CLK [in]	GLB_TSIZE_R%1%/CLK [PGDFFR]	2.30
CLK [in]	GLB_TS_DIS/CLK [PGDFFR]	2.30
CLK [in]	GLB_TS_RX/CLK [PGDFFR]	2.30
CLK [in]	GLB_XFER_TYPE%0%/CLK [PGDFFR]	2.30
CLK [in]	GLB_X...part1 *1/CLK [PGDFFR]	2.30
CLK [in]	GLB_X...part2 *2/CLK [PGDFFR]	2.30
GLB_A3031_R%0%/Q0 [PGDFFR]	A30 [bidi]	2.50
GLB_A3031_R%1%/Q0 [PGDFFR]	A31 [bidi]	2.50
GLB_BI_R/Q0 [PGDFFR]	BI_L [bidi]	2.50
GLB_BLAST_R/Q0 [PGDFFR]	BLAST_L [bidi]	2.50
GLB_BR/Q0 [PGDFFR]	BR_L [out]	2.50
GLB_BTERM_R/Q0 [PGDFFR]	BTERM_L [out]	2.50
GLB_BURST_R/Q0 [PGDFFR]	BURST_L [bidi]	2.50
GLB_LHOLDAX/Q0 [PGDFFR]	LHOLDA [out]	2.50
GLB_RDY_GATEX/Q0 [PGDFFR]	RDY_GATE [out]	2.50
GLB_TSIZE_R%0%/Q0 [PGDFFR]	TSIZE[0] [bidi]	2.50
GLB_TSIZE_R%1%/Q0 [PGDFFR]	TSIZE[1] [bidi]	2.50
GLB_TS_RX/Q0 [PGDFFR]	TS_L [bidi]	2.50
GLB_TS_RX/Q0 [PGDFFR]	TS_R [out]	2.50
LHOLDA_FB [in]	BB_L [bidi]	6.00
LHOLDA_FB [in]	RD_WRN [bidi]	6.00
GLB_CYC_CNT/Q0 [PGDFFR]	GLB_A3031_R%0%/D0 [PGDFFR]	6.20
GLB_CYC_CNT/Q0 [PGDFFR]	GLB_A3031_R%1%/D0 [PGDFFR]	6.20
GLB_LHOLDAX/Q0 [PGDFFR]	GLB_BLAST_R/D0 [PGDFFR]	6.20
GLB_LHOLDAX/Q0 [PGDFFR]	GLB_BTERM_R/D0 [PGDFFR]	6.20
GLB_LHOLDAX/Q0 [PGDFFR]	GLB_BURST_R/D0 [PGDFFR]	6.20
GLB_CYC_CNT/Q0 [PGDFFR]	GLB_CYC_CNT/D0 [PGDFFR]	6.20

GLB_LHOLDAX/Q0 [PGDFFR]	GLB_LHOLDAX/D0 [PGDFFR]	6.20
GLB_CYC_CNT/Q0 [PGDFFR]	GLB_RDY_GATEX/D0 [PGDFFR]	6.20
GLB_LHOLDAX/Q0 [PGDFFR]	GLB_TSIZE_R%0%/D0 [PGDFFR]	6.20
GLB_CYC_CNT/Q0 [PGDFFR]	GLB_TSIZE_R%1%/D0 [PGDFFR]	6.20
GLB_LHOLDAX/Q0 [PGDFFR]	GLB_TS_DIS/D0 [PGDFFR]	6.20
GLB_XFER_TYPE%0%/Q0 [PGDFFR]	GLB_TS_RX/D0 [PGDFFR]	6.20
GLB_LHOLDAX/Q0 [PGDFFR]	GLB_XFER_TYPE%0%/D0 [PGDFFR]	6.20
GLB_LHOLDAX/Q0 [PGDFFR]	GLB_X...part1 *1/D0 [PGDFFR]	6.20
GLB_LHOLDAX/Q0 [PGDFFR]	GLB_X...part2 *2/D0 [PGDFFR]	6.20
GLB_RDY_GATEX/Q0 [PGDFFR]	READYI_L [out]	6.20
GLB_LHOLDAX/Q0 [PGDFFR]	GLB_BR/D0 [PGDFFR]	6.30
A28 [in]	GLB_BI_R/D0 [PGDFFR]	7.10
TS_L [bidi]	ADS_L [bidi]	7.30
READYO_L [in]	TA_L [bidi]	7.30
RD_WRN [bidi]	WR_RDN [bidi]	7.30
ADS_L [bidi]	GLB_ADS_PL/D0 [PGDFFR]	7.40
WR_RDN [bidi]	LBE_L[0] [bidi]	9.80
WR_RDN [bidi]	LBE_L[1] [bidi]	9.80
WR_RDN [bidi]	LBE_L[2] [bidi]	9.80
WR_RDN [bidi]	LBE_L[3] [bidi]	9.80

Index Name Table

- *1 GLB_XFER_TYPE%1%_part1
- *2 GLB_XFER_TYPE%1%_part2

Required Setup and Hold:

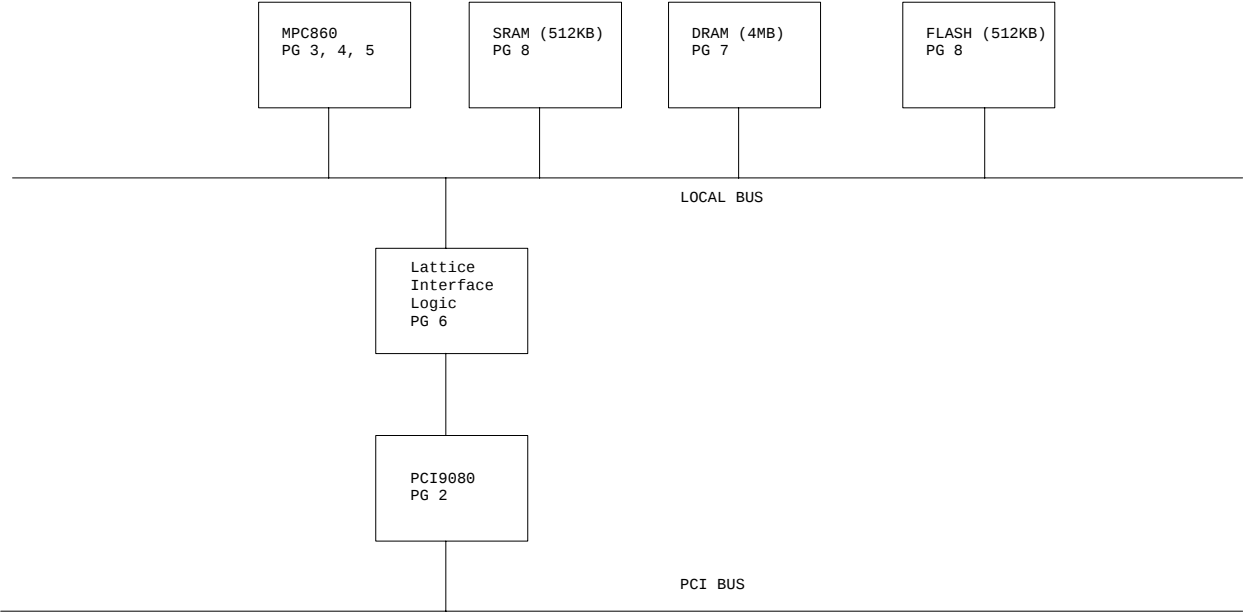
Inst.Name	Data	Clock	Setup	Hold
GLB_XFER_TYPE%0%	RST_L	CLK	5.50	-2.00
GLB_XFER_TYPE%0%	LBE_L[3]	CLK	5.40	-2.00
GLB_XFER_TYPE%0%	LBE_L[0]	CLK	5.30	-2.00
GLB_XFER_TYPE%0%	ADS_L	CLK	5.50	-2.00
GLB_XFER_TYPE%0%	LBE_L[1]	CLK	5.30	-2.00
GLB_XFER_TYPE%0%	LBE_L[2]	CLK	5.40	-2.00
GLB_XFER_TYPE%0%	WR_RDN	CLK	5.40	-2.00
GLB_A3031_R%1%	RST_L	CLK	5.50	-2.00
GLB_A3031_R%1%	LBE_L[3]	CLK	5.50	-2.00
GLB_A3031_R%1%	LBE_L[0]	CLK	5.50	-2.00
GLB_A3031_R%1%	LBE_L[1]	CLK	5.50	-2.00
GLB_A3031_R%1%	LBE_L[2]	CLK	5.50	-2.00
GLB_RDY_GATEX	RST_L	CLK	5.30	-2.00
GLB_A3031_R%0%	RST_L	CLK	6.00	-2.00
GLB_A3031_R%0%	LBE_L[3]	CLK	6.00	-2.00
GLB_A3031_R%0%	LBE_L[0]	CLK	6.00	-2.00
GLB_A3031_R%0%	LBE_L[1]	CLK	6.00	-2.00
GLB_A3031_R%0%	LBE_L[2]	CLK	6.00	-2.00
GLB_CYC_CNT	RST_L	CLK	5.50	-2.00
GLB_CYC_CNT	TA_L	CLK	5.50	-2.00
GLB_XFER_TYPE%1%_part2	RST_L	CLK	5.50	-2.00
GLB_XFER_TYPE%1%_part2	LBE_L[3]	CLK	5.50	-2.00
GLB_XFER_TYPE%1%_part2	LBE_L[0]	CLK	5.50	-2.00
GLB_XFER_TYPE%1%_part2	ADS_L	CLK	5.50	-2.00
GLB_XFER_TYPE%1%_part2	LBE_L[1]	CLK	5.50	-2.00
GLB_XFER_TYPE%1%_part2	LBE_L[2]	CLK	5.30	-2.00
GLB_XFER_TYPE%1%_part2	WR_RDN	CLK	5.30	-2.00
GLB_TSIZE_R%1%	RST_L	CLK	11.90	-2.00
GLB_TSIZE_R%1%	LBE_L[3]	CLK	11.90	-2.00
GLB_TSIZE_R%1%	LBE_L[0]	CLK	11.90	-2.00

GLB_TSIZE_R%1%	LBE_L[1]	CLK	11.90	-2.00
GLB_TSIZE_R%1%	LBE_L[2]	CLK	11.90	-2.00
GLB_XFER_TYPE%1%_part1	RST_L	CLK	5.50	-2.00
GLB_XFER_TYPE%1%_part1	LBE_L[3]	CLK	5.50	-2.00
GLB_XFER_TYPE%1%_part1	LBE_L[0]	CLK	5.50	-2.00
GLB_XFER_TYPE%1%_part1	ADS_L	CLK	5.50	-2.00
GLB_XFER_TYPE%1%_part1	LBE_L[1]	CLK	5.50	-2.00
GLB_XFER_TYPE%1%_part1	LBE_L[2]	CLK	5.30	-2.00
GLB_XFER_TYPE%1%_part1	WR_RDN	CLK	5.30	-2.00
GLB_TS_DIS	RST_L	CLK	5.30	-2.00
GLB_TSIZE_R%0%	RST_L	CLK	12.00	-2.00
GLB_TSIZE_R%0%	LBE_L[3]	CLK	12.00	-2.00
GLB_TSIZE_R%0%	LBE_L[0]	CLK	12.00	-2.00
GLB_TSIZE_R%0%	LBE_L[1]	CLK	12.00	-2.00
GLB_TSIZE_R%0%	LBE_L[2]	CLK	12.00	-2.00
GLB_BR	RST_L	CLK	5.40	-2.10
GLB_BR	LHOLD	CLK	5.40	-2.10
GLB_BURST_R	RST_L	CLK	5.30	-2.00
GLB_BURST_R	BLAST_L	CLK	5.30	-2.00
GLB_BURST_R	TA_L	CLK	5.30	-2.00
GLB_BTERM_R	BTERM_EN	CLK	5.30	-2.00
GLB_BTERM_R	RST_L	CLK	5.50	-2.00
GLB_BTERM_R	A28	CLK	5.10	-1.80
GLB_BTERM_R	A29	CLK	5.30	-2.00
GLB_BTERM_R	BLAST_L	CLK	5.30	-2.00
GLB_BTERM_R	BI_L	CLK	5.50	-2.20
GLB_BTERM_R	TA_L	CLK	5.30	-2.00
GLB_BLAST_R	READY0_L	CLK	5.30	-2.00
GLB_BLAST_R	RST_L	CLK	5.30	-2.00
GLB_BLAST_R	A28	CLK	5.10	-1.80
GLB_BLAST_R	A29	CLK	5.30	-2.00
GLB_BLAST_R	BB_L	CLK	5.30	-2.00
GLB_BLAST_R	TS_L	CLK	5.30	-2.00
GLB_BLAST_R	BURST_L	CLK	5.30	-2.00
GLB_BI_R	RST_L	CLK	5.50	-2.00
GLB_BI_R	A28	CLK	5.10	-1.80
GLB_BI_R	A29	CLK	5.30	-2.00
GLB_BI_R	TS_L	CLK	5.50	-2.00
GLB_BI_R	A30	CLK	5.30	-2.00
GLB_BI_R	A31	CLK	5.50	-2.20
GLB_BI_R	BURST_L	CLK	5.50	-2.00
GLB_TS_RX	RST_L	CLK	5.50	-2.00
GLB_ADS_PL	ADS_L	CLK	5.40	-2.10
GLB_LHOLDAX	BG_L	CLK	5.30	-2.00
GLB_LHOLDAX	RST_L	CLK	5.30	-2.00
GLB_LHOLDAX	LHOLD	CLK	5.30	-2.00
GLB_LHOLDAX	BB_L	CLK	5.30	-2.00

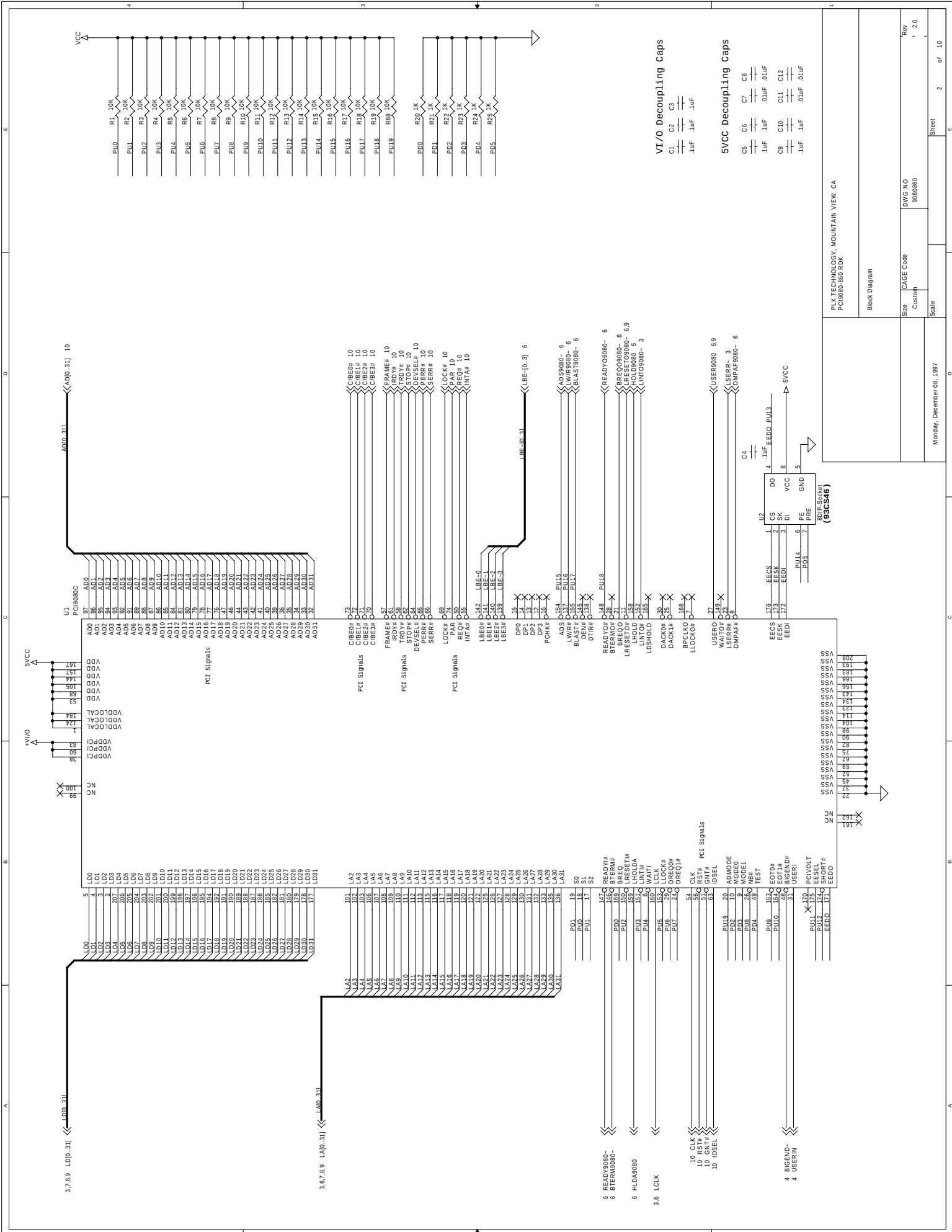
ORCAD SCHEMATICS

PCI9080RDK-860 BLOCK DIAGRAM

ECN HISTORY		
ECN NUMBER	DATE	APPROVAL
PCI9080RDK-860 Rev 1.0	7/23/97	
PCI9080RDK-860 Rev 2.0	12/5/97	



This reference design has been developed and tested by Vitana Corp. Vitana Corporation Tel: 613-749-4445 Email: rdk@vitana.com Web: www.vitana.com	PLX TECHNOLOGY, MOUNTAIN VIEW, CA PCI9080-860 RDK		
	Block Diagram		
H. W. Leahy	Size Custom	CAGE Code 9080860	DWG NO 9080860
Friday, December 05, 1997	Scale	Sheet	1 of 10



PLX TECHNOLOGY, MOUNTAIN VIEW, CA	
PCI9080-800 RDK	
Block Diagram	
Size	CAGE Code
Custom	DWG NO
Scale	9080800
Monday, December 08, 1997	Rev
Sheet	2 of 10

U1	PCI9080C
U2	93C46
C1	10K
C2	10K
C3	10K
C4	10K
C5	10K
C6	10K
C7	10K
C8	10K
C9	10K
C10	10K
C11	10K
C12	10K

U1	PCI9080C
U2	93C46
C1	10K
C2	10K
C3	10K
C4	10K
C5	10K
C6	10K
C7	10K
C8	10K
C9	10K
C10	10K
C11	10K
C12	10K

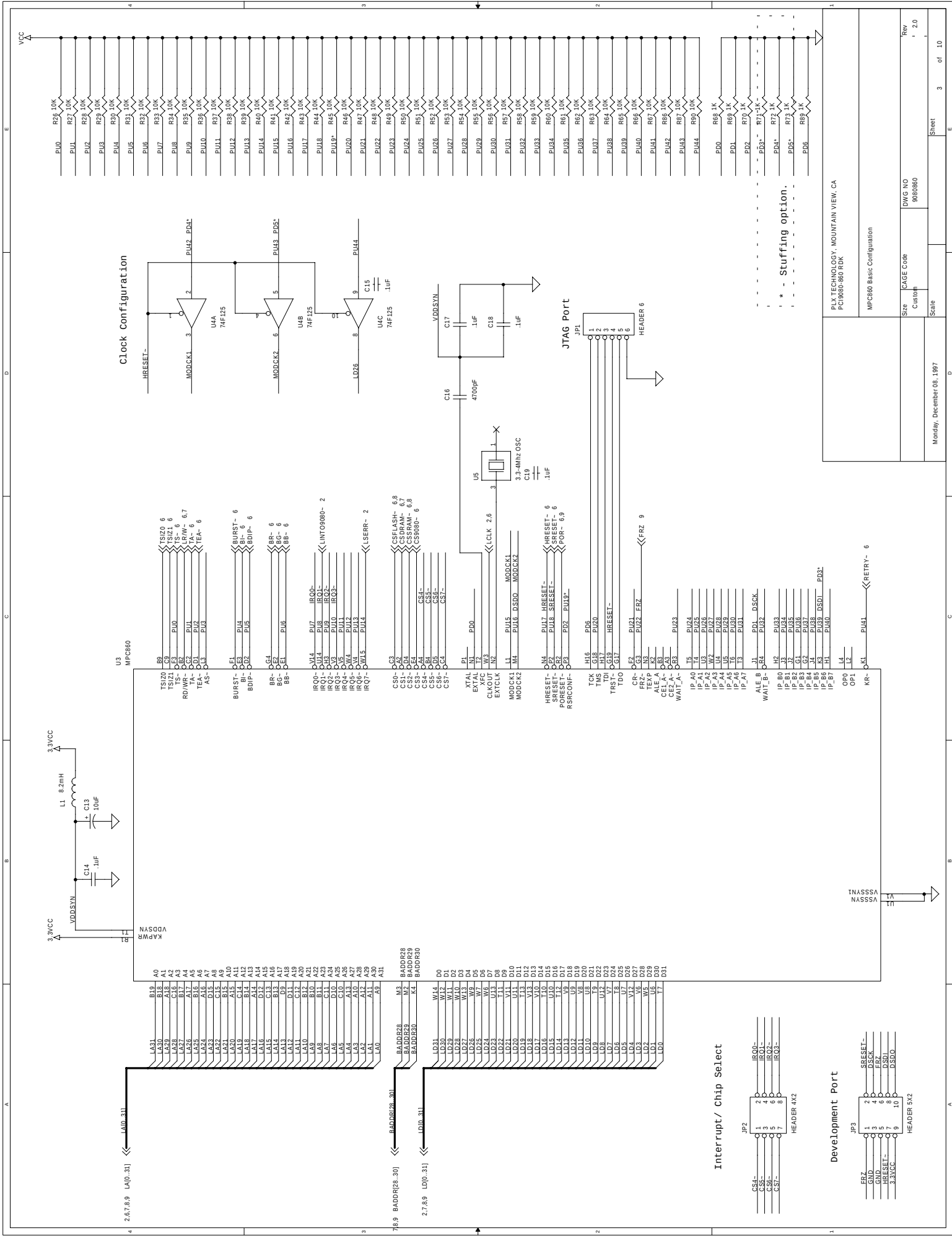
U1	PCI9080C
U2	93C46
C1	10K
C2	10K
C3	10K
C4	10K
C5	10K
C6	10K
C7	10K
C8	10K
C9	10K
C10	10K
C11	10K
C12	10K

U1	PCI9080C
U2	93C46
C1	10K
C2	10K
C3	10K
C4	10K
C5	10K
C6	10K
C7	10K
C8	10K
C9	10K
C10	10K
C11	10K
C12	10K

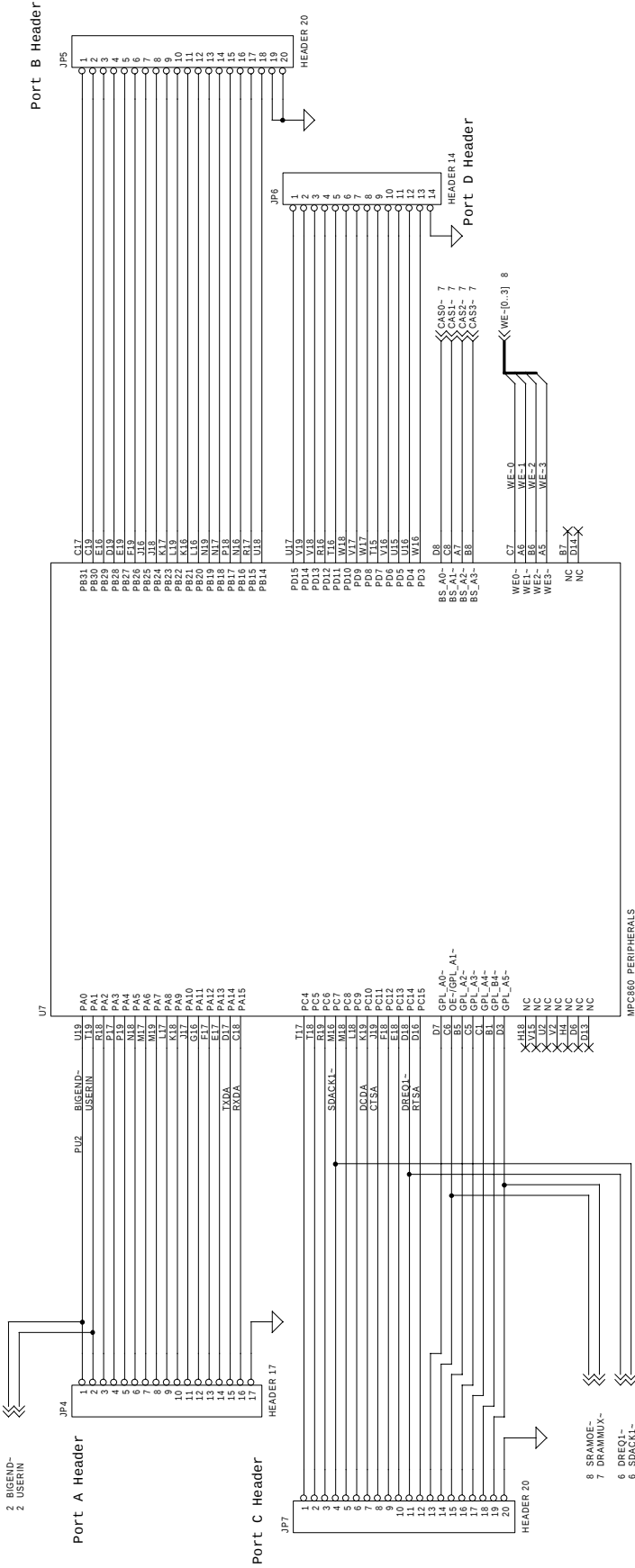
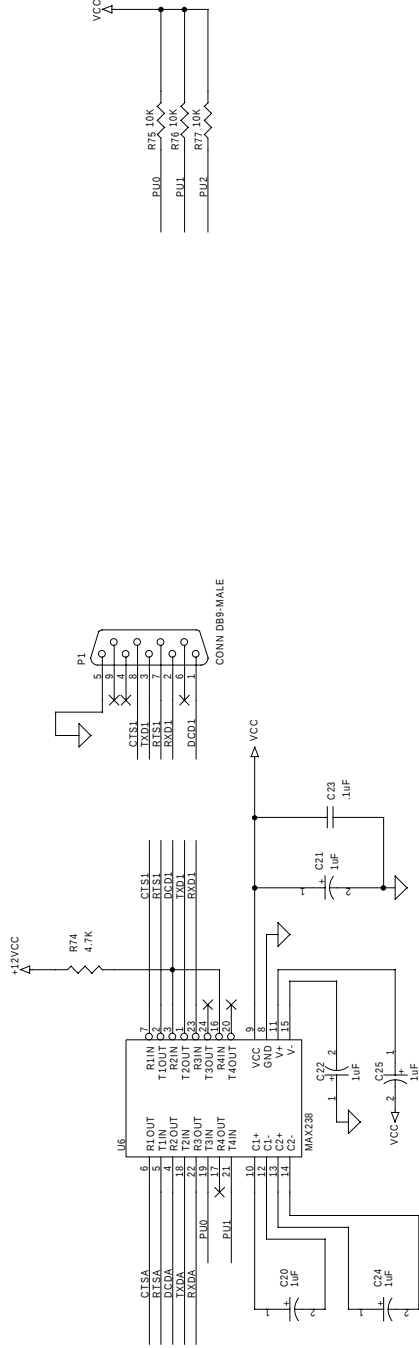
U1	PCI9080C
U2	93C46
C1	10K
C2	10K
C3	10K
C4	10K
C5	10K
C6	10K
C7	10K
C8	10K
C9	10K
C10	10K
C11	10K
C12	10K

U1	PCI9080C
U2	93C46
C1	10K
C2	10K
C3	10K
C4	10K
C5	10K
C6	10K
C7	10K
C8	10K
C9	10K
C10	10K
C11	10K
C12	10K

U1	PCI9080C
U2	93C46
C1	10K
C2	10K
C3	10K
C4	10K
C5	10K
C6	10K
C7	10K
C8	10K
C9	10K
C10	10K
C11	10K
C12	10K



Serial Port



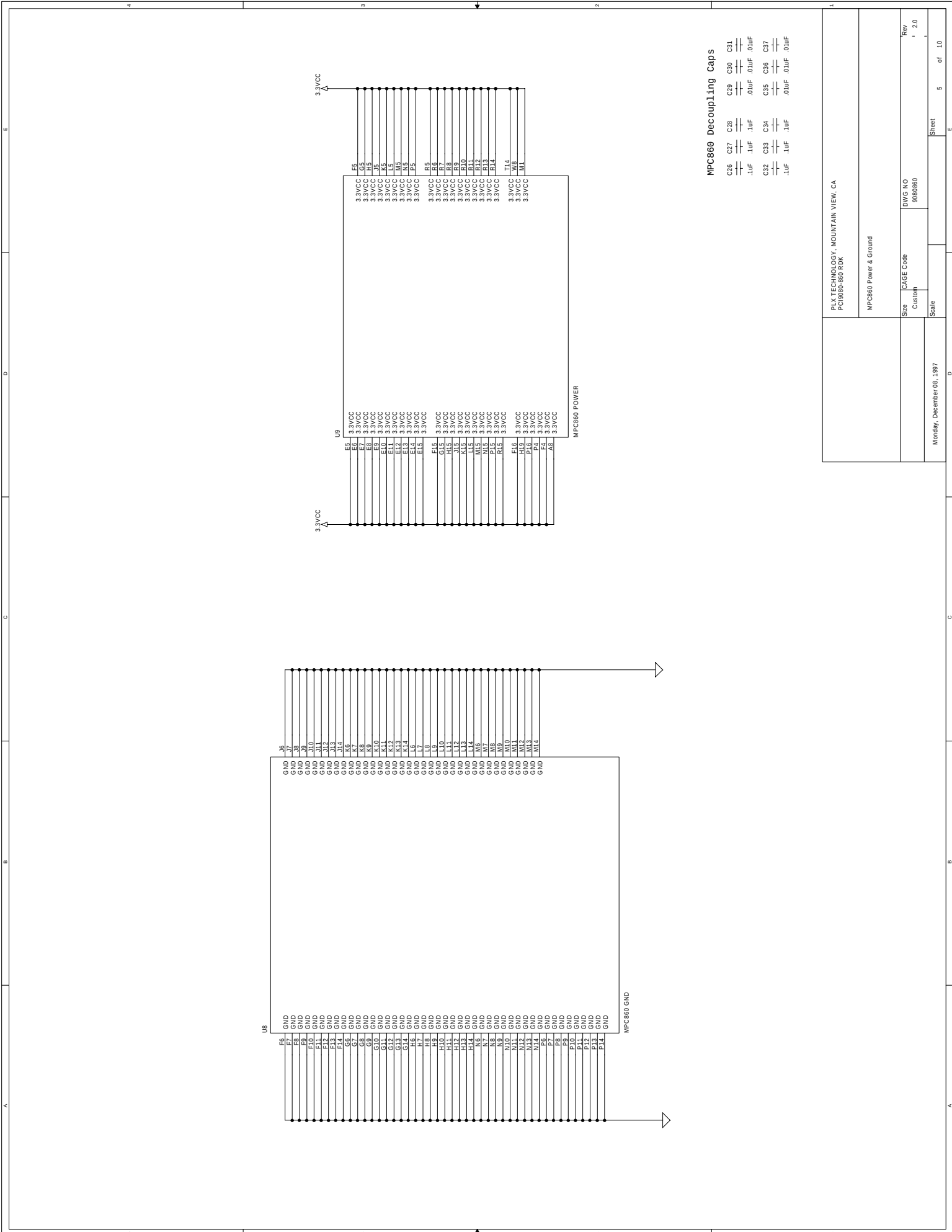
Note:
Serial Port 1 uses SCC1

MPC860_PERIPHERALS

PLX TECHNOLOGY, MOUNTAIN VIEW, CA
PCI9880-880 RDK

MPC860 Peripherals

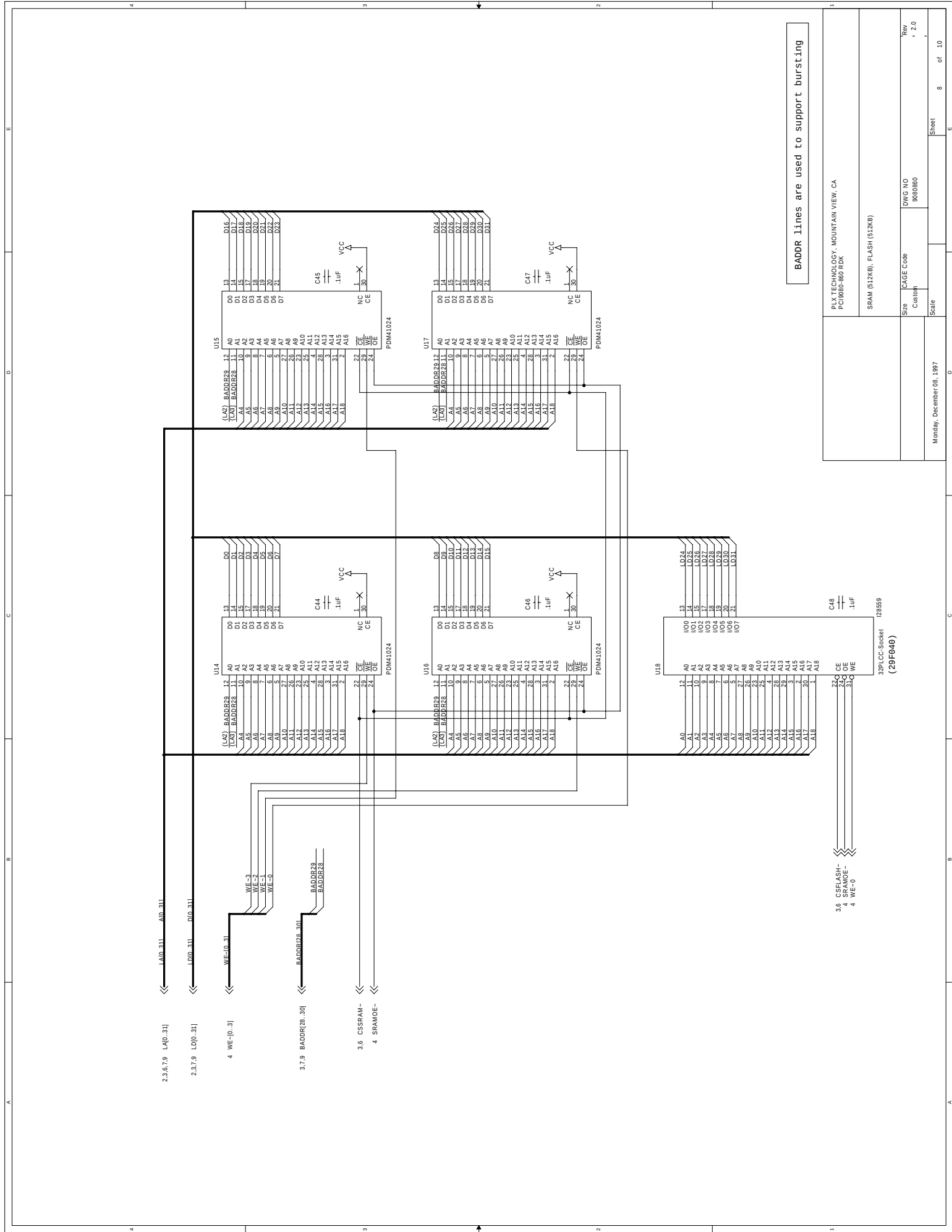
Size	CAGE Code	DWG NO	Rev
Custom		9080880	2.0
Scale			
Monday, December 08, 1997			
Sheet			4 of 10



MPC860 Decoupling Caps

C26	C27	C28	C29	C30	C31
.1uF	.1uF	.1uF	.01uF	.01uF	.01uF
C32	C33	C34	C35	C36	C37
.1uF	.1uF	.1uF	.01uF	.01uF	.01uF

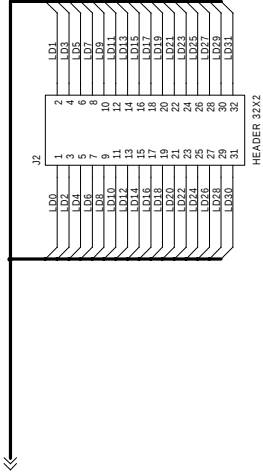
PLX TECHNOLOGY, MOUNTAIN VIEW, CA		PC1980-880 RDK	
MPC860 Power & Ground			
Size	CAGE Code	DWG NO	Rev
Custom		9080880	2.0
Monday, December 08, 1997		Sheet	5 of 10



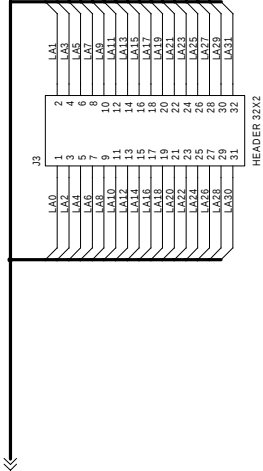
BADDR lines are used to support bursting

PLX TECHNOLOGY, MOUNTAIN VIEW, CA PCI9880-880 RDK		SRAM (512KB) FLASH (512KB)	
Size	CAGE Code	DWG NO	Rev
Custom		9080880	r. 2.0
Scale		Sheet	8 of 10
Monday, December 08, 1997			

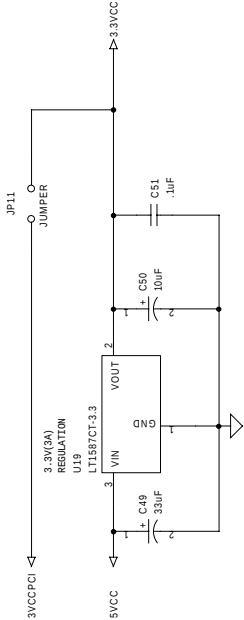
Data Bus



Address Bus

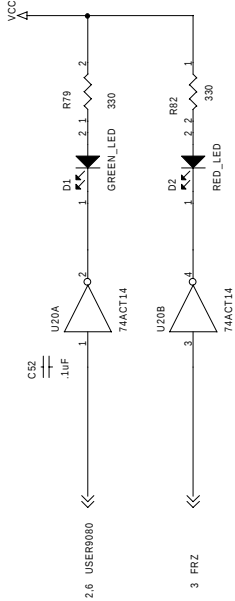


Local Bus Power Source

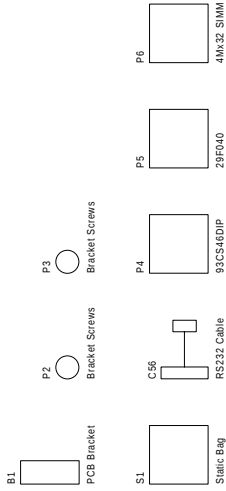


Use regulator U19 if PCI motherboard does not support 3.3V
Use Jumper JP11 if motherboard does support 3.3V

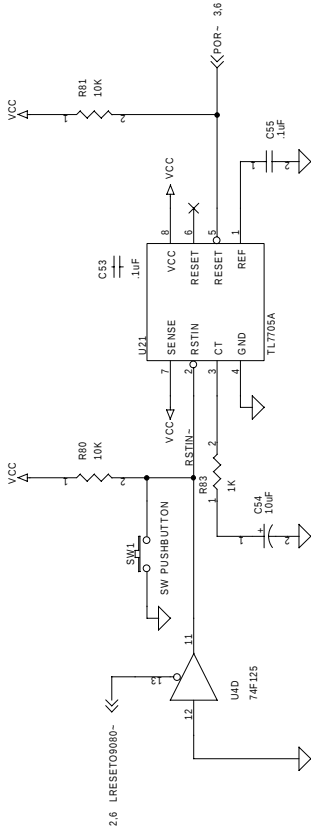
Indicator LEDs



Misc Parts



Reset Circuitry

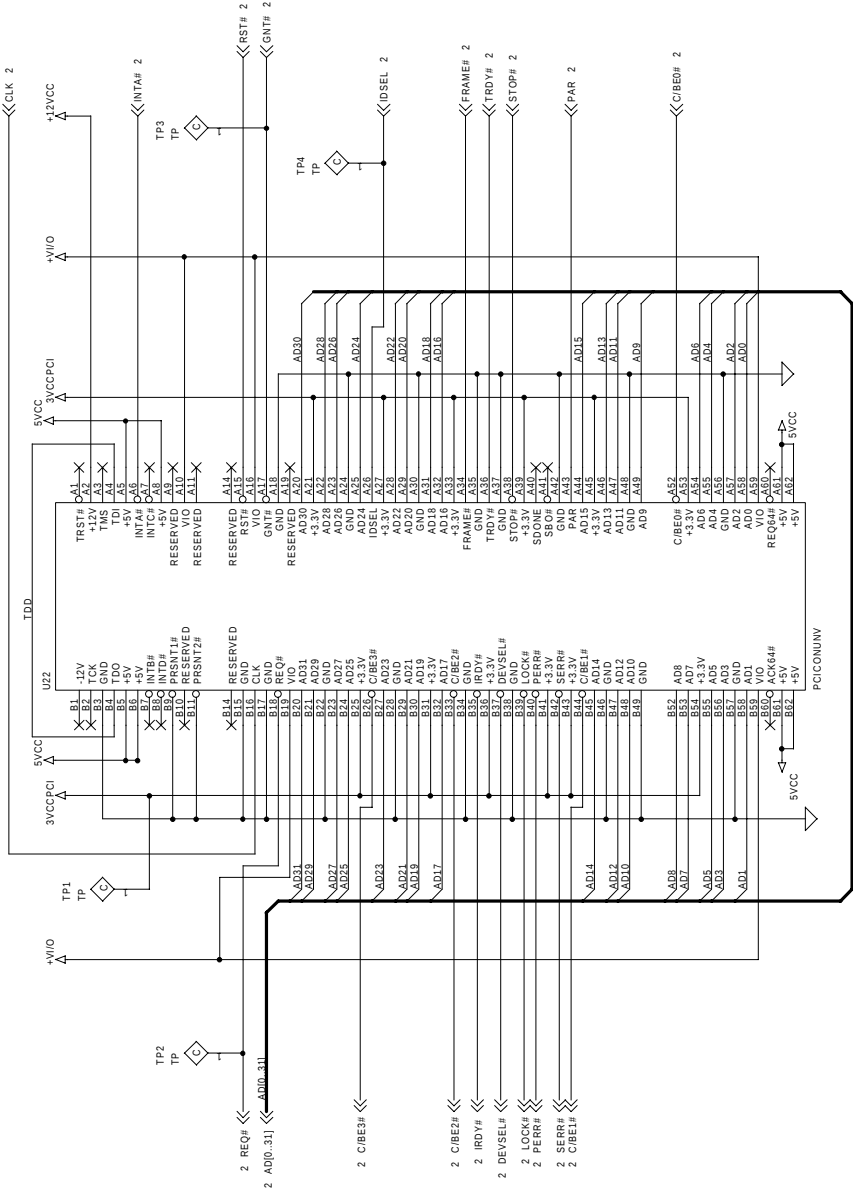


PLX TECHNOLOGY, MOUNTAIN VIEW, CA
PCI9880-880 RDK

Reset

Size	CAGE Code	DWG NO	Rev
Custom		9080880	2.0
Scale	Sheet	of	10
Monday, December 08, 1997	9		

PCI Edge Connector



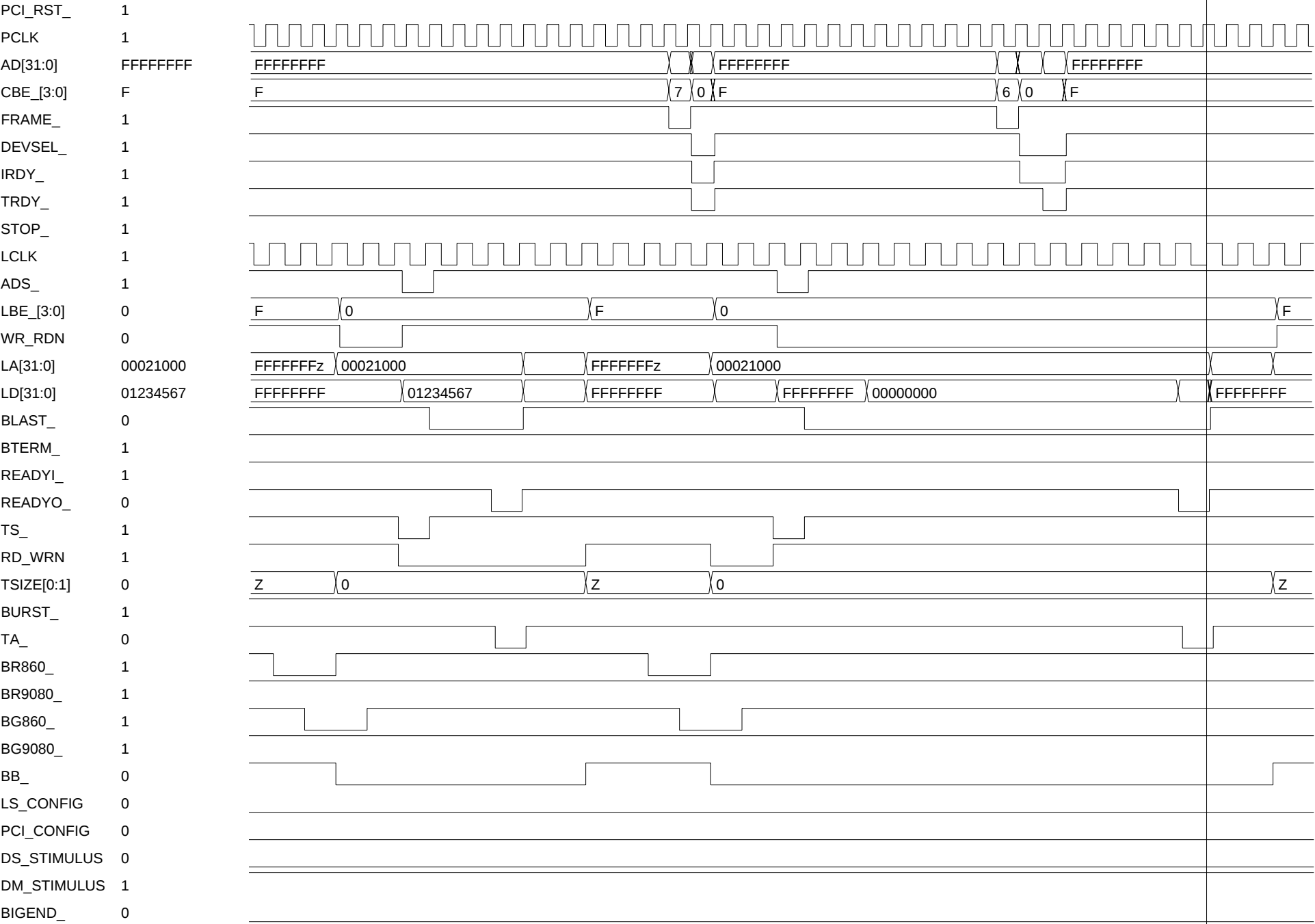
Note:
This connector is keyed for both 3.3v and
5V PCI I/O

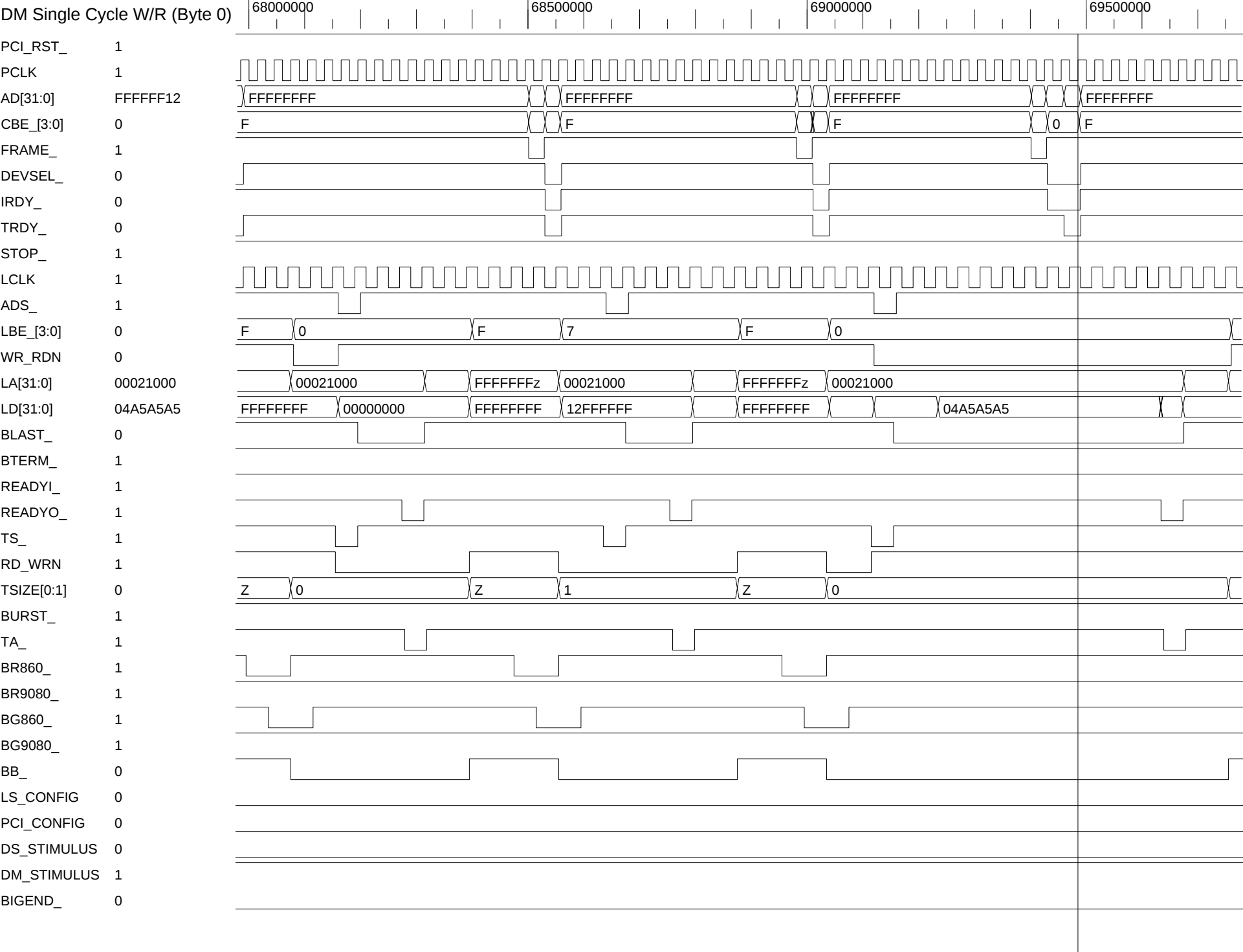
PLX TECHNOLOGY, MOUNTAIN VIEW, CA PCI9860-860 RDK		PCI Edge Connector	
Size	CAGE Code	DWG NO	Rev
Custom	9080860		2.0
Scale			
Monday, December 08, 1997		Sheet	10 of 10

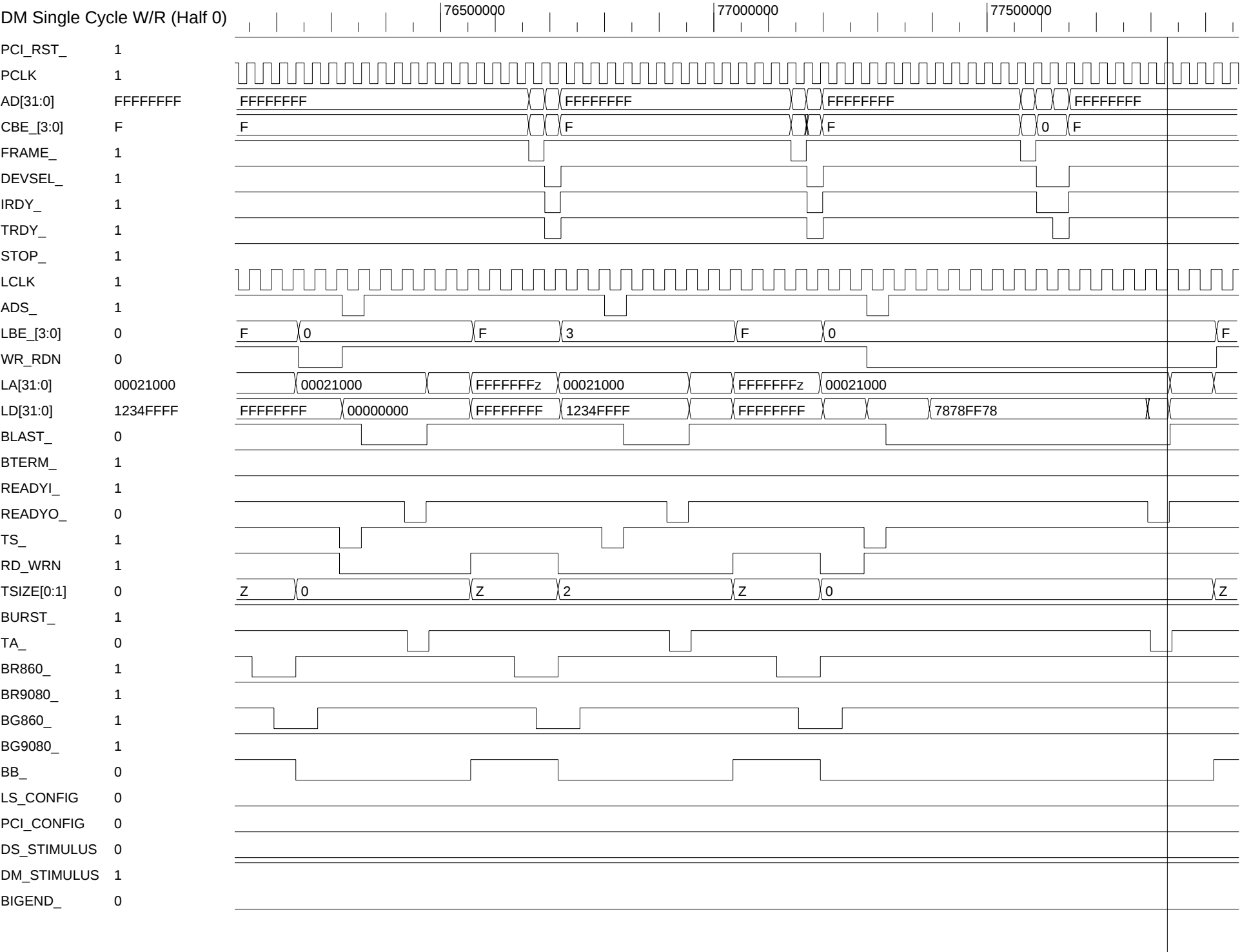
TIMING DIAGRAMS



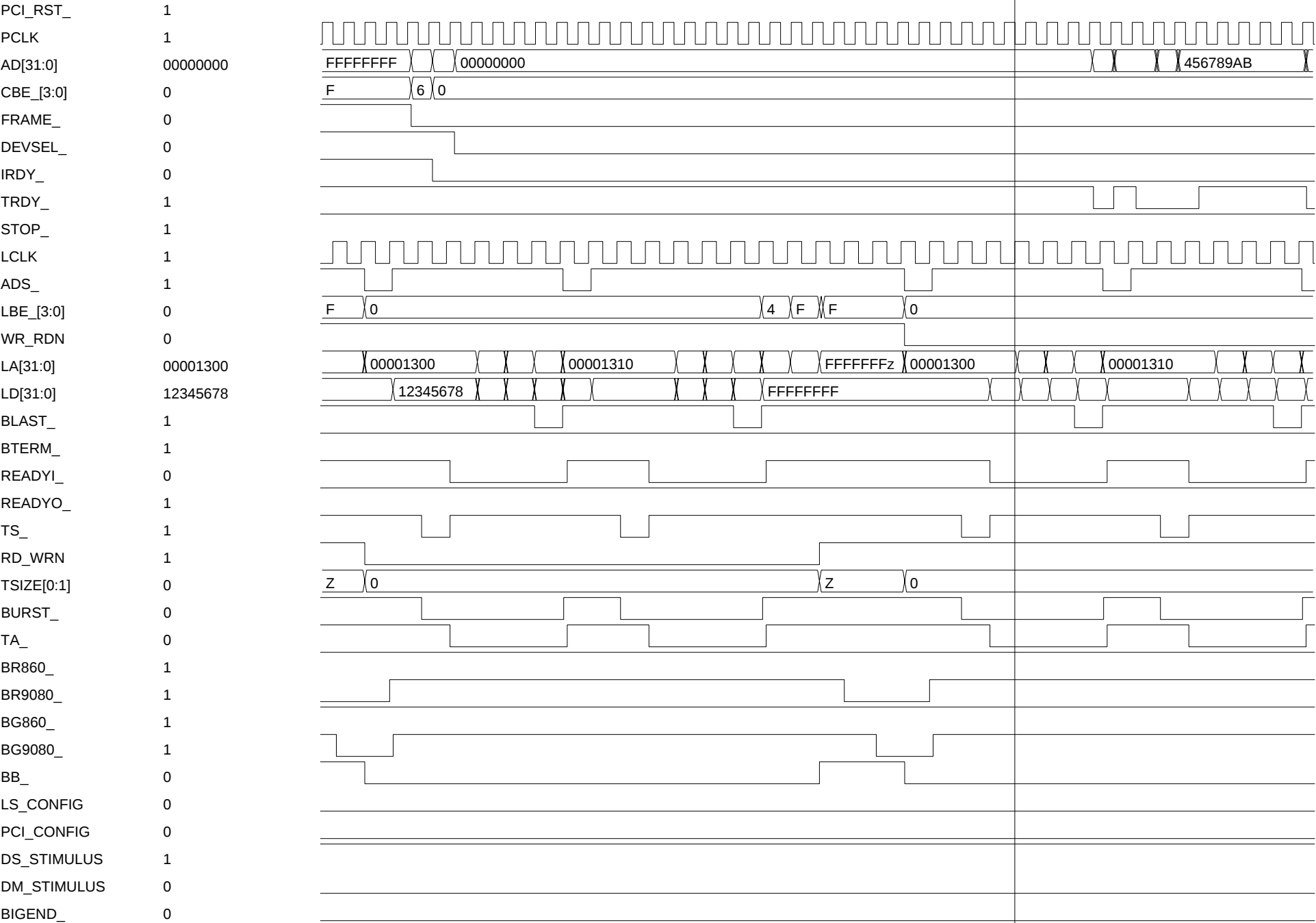
DM Single Cycle W/R







DS Burst W/R 8 Dwords



DS Single Cycle Write/Read

