CY7C924DX/CY7C9689 HOTLink® Evaluation Board User's Guide

Overview

This document describes the operation, interfaces and construction of the CY7C924DX-EB evaluation board. Complete layout, schematics and the VHDL code for programming the CPLD are included at the end of this document.

The CY7C924DX/CY7C9689 Evaluation Board is a PCI-bus compliant circuit card designed to demonstrate the full capability of the CY7C924DX HOTLinks® transceiver or the CY7C9689 TAXI™ compatible HOTLink Transceiver in point-to-point communication applications.

The CY7C924DX Evaluation Board comes with three different serial interface options: CAT 5 100Ω unshielded twisted pair (UTP) interface, 75Ω RG-59 coaxial cable interface, and multi-mode Optical Fiber interface. The CY7C9689 Evaluation Board has two different serial interface options: CAT 5 100Ω Unshielded Twisted Pair (UTP) interface and 75Ω RG-59 coaxial cable interface.

Functional Description

The CY7C924DX/CY7C9689 evaluation board enables transfer of data between two computers through HOTLink transceivers at speed-up to 200 MBd. This evaluation board interfaces to the host system through a PCI bus. Refer to *Figure 1*.

The host will write data to be transmitted into the PCI-DualPort (PCI-DP) through the PCI-bus. The data transfer block size is fixed to 512 bytes. The host then updates a mailbox in the PCI-DP and causes a local interrupt on the PCI-DP's local bus. Upon the reception of the local interrupt, the CPLD will transfer the block of data from the PCI-DP into the transmit external FIFO. The HOTLink Transceiver will then automatically read from the external FIFO to its internal FIFO and transmit the block of data out to the serial medium.

The transceiver receives serial data from the serial medium. Any K28.5 SYNC character is filtered out by the transceiver, then valid data is written into the internal receive FIFO. These data blocks will then be automatically transferred into the external receive FIFO. The CPLD detects valid data present at the RXFIFO and it will transfer 512 bytes of data from the receive FIFO into the PCI-DP. The CPLD will then update a mailbox which is programmed to cause an interrupt on the PCI-bus. Once an interrupt is detected, the host's interrupt service routine will read the block of data from the PCI-DP. Another method is polling, where the host will poll the mailbox and read the block of data from PCI-DP upon change of state in the mailbox.

Block Diagram

Figure 1 is the block diagram of the system evaluation board.

PCI Interface

This evaluation board uses the CY7C09449 PCI-DP as its PCI interface. One side of this specialized PCI-DP SRAM interfaces to the 32-bit 33-MHz PCI bus, the other side of this Dual Port interfaces to the 16-bit Local Bus running at 24 MHz.

Local Bus to FIFOs Bridge (LBFB)

The 37K CPLD (CY7C37256P208) connects the bidirectional local bus of the PCI-DP and two CY7C4275 (32K X 18) syn-

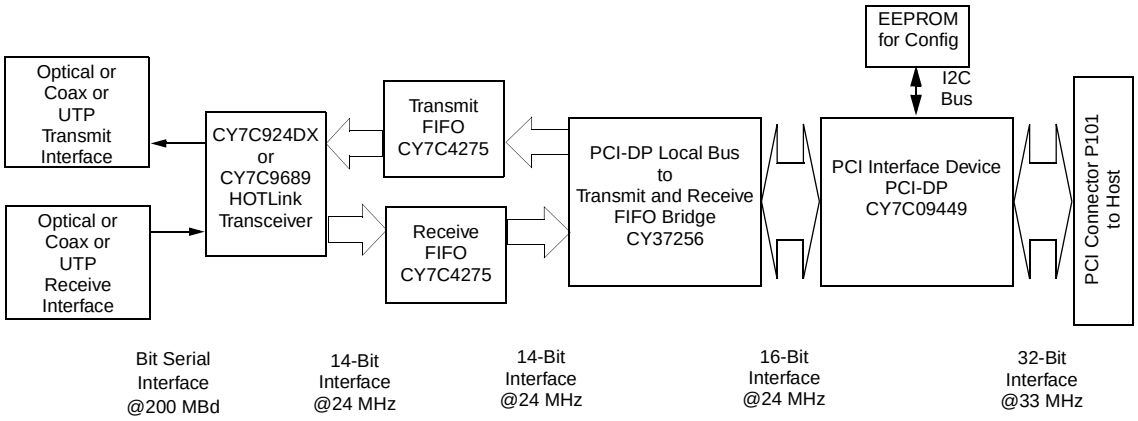


Figure 1. CY7C924DX/CY7C9689 Evaluation Board Block Diagram

chronous FIFOs. The CPLD acts as arbitrator and data transfer paths between the FIFOs and the local bus.

Data Buffering FIFOs

Two 32k X18 synchronous FIFOs are used on this board for data buffering. One FIFO is used for Transmit Buffering (TXFIFO) and the other is used for Receive Buffering (RXFIFO).

High-Speed Serial Transceiver

The CY7C924DX/CY7C9689 is the transceiver device on the board. The transceiver's transmit and receive parallel buses are connected to the TXFIFO and RXFIFO parallel buses. On the transmit side, the HOTLink device reads data from the transmit parallel bus, serializes the data and transmits this serial data to the medium. On the receive side, the HOTLink receives serial data, deserializes it and writes it to the receive parallel bus.

Clock Distribution

The CY7B991 RoboClock performs clock distribution to the PCI-DP's Local Bus, TXFIFO and RXFIFO buses.

Configuration Settings

The jumper block J201 must be configured correctly before the evaluation board can be powered up or installed into the PCI slot within a PC. This jumper block controls the CY7C924DX or CY7C9689 inputs directly. Six static configuration inputs of the CY7C924DX (or five of the CY7C9689) can be controlled by this jumper block. When a jumper is not installed, the corresponding input signal is pulled HIGH through a 4.7 kΩ resistor. When a jumper is installed, the corresponding signal is pulled to ground (logic LOW).

The A/B* serial inputs select, TEST*, DLB1 and DLB0 diagnostic loopback selects (note that the DLB1 location does not apply to the CY7C9689 version), SDPSEL speed select, and RANGESEL range select inputs of the CY7C924DX and CY7C9689 are controlled by J201 (see *Figure 2*).

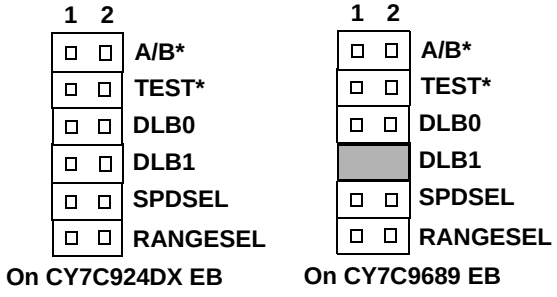


Figure 2. Jumper Block J201

A/B* Input Select

When the A/B* jumper is installed, the INB± serial input is selected. When this jumper is not installed the INA± serial input is selected.

For the Fiber-optic and Coaxial Cable option, INA± is always used; therefore, this jumper must NOT be installed.

For the UTP option, this jumper selects the incoming signal from either J301 or J302. When this jumper is installed, the incoming serial signal from J301 is selected. When this jumper is not installed, the incoming signal from J302 is selected.

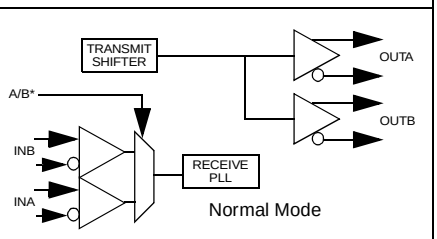
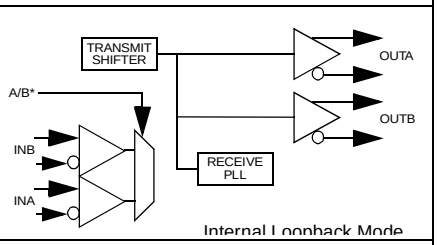
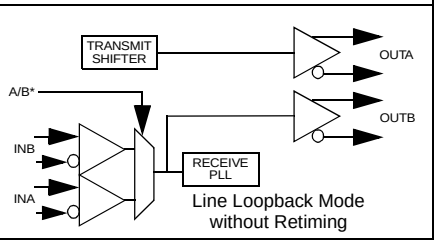
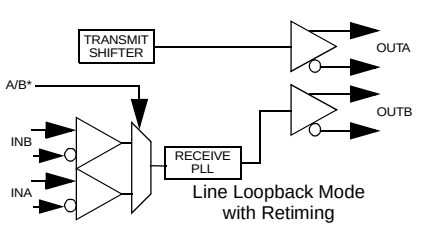
TEST*

The TEST* jumper directly controls the TEST* input of the CY7C924DX or CY7C9689. For normal operation the TEST* input must be pulled HIGH. Normally, no jumper should be installed at this position. When a jumper is installed at this position, the CY7C924DX or CY7C9689 will be put into factory test mode.

DLB[1:0] Diagnostic Loopback

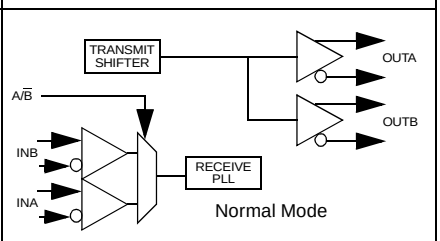
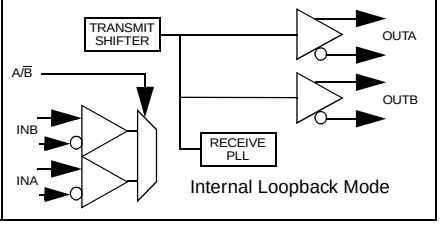
For the CY7C924Dx version of the evaluation board, the DLB0 and DLB1 jumpers control the respective diagnostic loopback inputs of the CY7C924DX. The jumper settings are described in *Table 1*.

Table 1. CY7C924DX DLB[1:0] Control Functions

DLB[1]	DLB[0]	Data Connections
With Jumper	With Jumper	
With Jumper	No Jumper	
No Jumper	With Jumper	
No Jumper	No Jumper	

For the CY7C9689 version of the evaluation board, the DLB1 jumper is not used. The DLB1 jumper location is a no connect, therefore a jumper should not be installed at this position. The DLB0 jumper location is connected to the DLB input of the CY7C9689. When a jumper is installed at DLB0, the DLB input is pulled LOW, and the CY7C9689 operates normally. When no jumper is installed at this location, the DLB input is asserted HIGH, which causes the CY7C9689 to internally loopback the transmit serial data to the receiver input. This is depicted in *Table 2*.

Table 2. CY7C9689 DLB[0] Control Functions

DLB[0]	Data Connections
With Jumper	
No Jumper	

Speed Select

The SPDSEL jumper location controls the SPDSEL input of the CY7C924DX or CYC9689. Along with the RANGESEL control jumper, this jumper controls the serial link operation data rate. Refer to *Table 3* for the link speed selection.

Range Select

The RANGESEL jumper location controls the RANGESEL input of the CY7C924DX or CY7C9689 directly. Along with the SPDSEL control jumper, it controls the serial link operation data rate. Refer to *Table 3* for the link speed selection. Please refer to the CY7C924DX or CY7C9689 data sheet for more details on the full operating range.

Table 3. Speed Select and Range Select Settings

SPDSEL	RANGESEL	Serial Data Rate (MBaud)	REFCLK Frequency (MHz)
With Jumper	With Jumper	100	20
With Jumper	No Jumper	50	20
No Jumper	With Jumper	200	20
No Jumper	No Jumper	100	20

Transceiver Static Control Settings

The CY7C924DX and CY7C9689 are configured specifically for this evaluation board application. The static control inputs of the transceivers are directly controlled by the CPLD.

The CY7C924DX is configured to 8-bit mode with internal FIFO enabled. The external FIFO feature is also enabled. The receive character discard policy to mode 2, where all K28.5 characters are discarded.

The CY7C9689 is configured to 8-bit mode with internal FIFOs enabled. The external FIFO feature is also enabled. The receive character discard policy is also set to mode 2, where all the JK synch characters are discarded.

Table 4. CY7C924DX Static Control Configuration

Control Pin	State
BYTE8/10*	High
FIFOBYP*	LOW
EXTFIFO	HIGH
RXMODE0	LOW
RXMODE1	HIGH

Table 5. CY7C924DX Static Control Configuration

Control Pin	State
BYTE8/10*	High
FIFOBYP*	LOW
EXTFIFO	HIGH
RXMODE0	LOW
RXMODE1	HIGH

Board Power Supply

The evaluation board is designed to be powered by a single 5V power supply. This 5V power can be supplied from the PCI bus connector, or the power connector J401. The Power Connector J401 is used when the board is needed to be powered up out of the PC chassis. This three-pin power connect is used to supply the whole board. *Figure 3* shows the pin assignment of the J401 power connector.

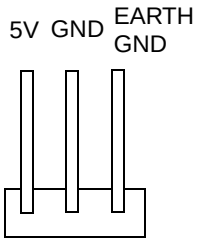


Figure 3. J401 Power Connector Pin Assignment

The CY09449 PCI-DP is a 3.3V device. The power to the PCI-DP is supplied by a linear power regulator. This linear power regulator converts 5V power supply to 3.3V. Therefore only a single 5V power supply is needed.

Theory of Operation

The CY7C924DX/CY7C9689 Evaluation Board allows basic data transfer between two PCs equipped with the same type of boards. Conceptually, the host accesses the evaluation board through the PCI bus. The PCI interface on the evaluation board is usually in slave mode, except when it is instructed to perform Master DMA to host memory. Through the PCI bus, the host can perform four basic functions to the evaluation board: EEPROM initialization, evaluation board initialization, transmit data, and receive data.

The PCI-DP is compliant to PCI standard version 2.1 on the PCI-bus side; its local bus can be configured to gluelessly interface to many different types of microprocessors, e.g. MPC7X0, 960i, etc. The PCI-DP has a feature which allows the PCI configuration registers and the local bus configuration register to be loaded from an EEPROM during every PCI-bus reset. This allows the appropriate vendor, board specific in-

formation and local bus configuration to be loaded into the respective registers. The PCI-DP also has the capability to program the EEPROM through the I^2C bus. (For more detail, please refer to the PCI-DP data sheet.)

During a PCI-bus reset, the PCI-DP would read the EEPROM value and try to load the read values into the respective registers. At the same time a sequence of fixed value is expected, i.e. the “signature”. If the signature is not correct, the PCI-DP would use the hardwired default values instead. Thus, in the case of a new board with an “empty” EEPROM, the evaluation board equipped with the PCI-DP will still come up properly and the driver would still be able to communicate with the board.

Every time a new EEPROM is placed on the board, it must be programmed to work with the appropriate local bus settings. In the case of this evaluation board, the local bus must be configured to a 16-bit bus with other specific local bus control signals settings. The local bus configuration settings will be discussed in a later section.

Assuming point-to-point configuration, the two ends of the link will power up at different times. The evaluation board that powers up first would start to receive extraneous data, since the incoming serial signal is not valid and the receiver would fill the internal RXFIFO with invalid characters. This invalid information would then be transferred to the external RXFIFO, and the CPLD would in turn transfer the data to the PCI-DP. Consequently, when both PCs are completely powered up, there would be invalid data stored in the PCI-DP.

The Initialization function is used to reset the CPLD, TXFIFO, RXFIFO, and the transceiver’s internal TXFIFO and RXFIFO. Thus resetting the state of the Evaluation Board to a “clean” state. The detailed sequence of initialization will be discussed later. It is important that both ends of the link are initialized properly before any data transfer takes place.

Transmit and receive data transfers are done at a granularity of 512 byte blocks. When the host performs data transfer operations to the evaluation board, it must transmit or receive 512 bytes at a time. There is no protocol overhead with this fixed block size approach. More details on transmit and receive data transfer operations will follow in the up-coming sections.

EEPROM Initialization

As discussed before, the local bus of the PCI-DP is configured by reading from the EEPROM during a PCI-bus reset. In order for the PCI-DP to communicate properly with the CPLD state machine, a value of 0x00000900 must be written to the Local Bus Configuration Register (LBUSCFG) at offset 0x04FC. This defines the communication protocol between the PCI-DP and the CPLD state machine. Therefore, the internal byte offset 0x6C of the EEPROM must be programmed with this value.

For these new EEPROM settings to take effect, the PC needs to be powered down and powered up or perform PCI-bus reset. The contents of the EEPROM are only read during PCI-bus reset. Once the EEPROM is written, it retains the setting until modified.

Evaluation Board Initialization

To clear all the FIFOs on the evaluation board, the board needs to be reset by writing to the Host Control Register (HCTL) of the PCI-DP located at offset 0x04E0. A value of 0x00000001 is written to the HCTL register to hold the board in reset and then 0x00000000 is written to clear the reset. The register bit HCTL[0] controls the RSTOUTD# of the PCI-DP, when this bit is set this pin asserts LOW; when this bit is cleared this pin asserts HIGH. The RSTOUTD# pin is directly connected to the CPLD. The CPLD is programmed to assert reset to all on-board FIFOs and all internal finite state machines (F.S.M.). The HCTL[1] bit controls the internal reset of the PCI-DP.

Next, the local interrupt from the PCI-DP to the CPLD has to be setup using the Local Processor Interrupt Control Status Register (LINT), located at offset 0x04F4, by writing 0x0008FFFF to it. This enables the Host-to-Local Mailbox interrupt on the local side.

Finally, the buffer flags of the CPLD state machine must be set. This is done by writing 0x01000001 to the Host to Local Data Mailbox (HLDATA) located at offset 0x04E8. After the write to the HLDATA register, the CPLD state machine sets its internal flags and clears the local interrupt. Reading the LINT register indicates whether the CPLD has processed the local interrupt. If LINT reads 0x00080008, it means that the CPLD has not cleared the interrupt. If reading the LINT gives 0x00080000, this indicates that the CPLD has processed the interrupt.

There are two main reasons for the CPLD not clearing the interrupt. The first reason is that the CPLD is still processing the interrupt and did not write to LINT to clear the interrupt. The second reason is that the incoming signal does not meet the link requirements, e.g., signal level, transition density, or frequency range, i.e., LFI* output from the transceiver is asserted. This is indicated by the red LED. The initialization state machine will hold the CPLD, TX and RXFIFOs in reset state. When the CPLD is in this state, it will not service incoming local interrupts. Once LFI* is deasserted, the initialization state machine will carry on, release resets to the TX and RX FIFOs and initialize the other state machines within the CPLD to operational states.

Transmit Path

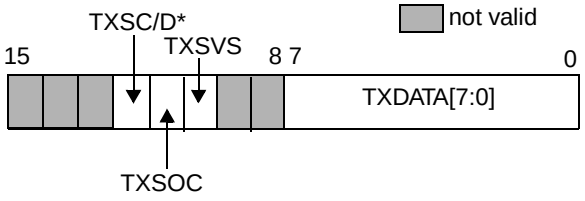
Overview

The evaluation board transfers data in blocks from the host to the local side and then, the HOTLink Transceiver transmits the data one byte at a time serially. *Figure 1* shows a block diagram of the data transfer path. Under host control, data is transferred to the PCI-DP's shared memory. Then, a shared register is set in the PCI-DP. This causes an interrupt on the local side. The CPLD receives the interrupt signal and the state machine in the CPLD starts to process the data. Data is transferred from the PCI-DP's shared memory through the CPLD and into the Tx FIFO. Then, the CPLD clears the interrupt on the local bus side. Once the HOTLink transceiver receives a signal from the Tx FIFO indicating data is available, it transmits the data. The sections below explain the five main sections of the data transmit path in detail.

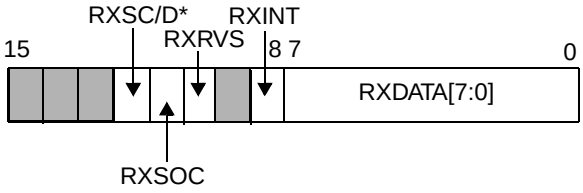
Data Byte Format

The PCI-DP SRAM address space is divided into two sections. One section is used for transmit data buffering (TXBUFF) and another section is used for receive data buffering (RXBUFF). The transmit buffer starts at offset address 0x4000 and the receive buffer starts at 0x6000. This board allows the sending of command characters through software. A data byte is represented by 13 bits of information (Note: 14

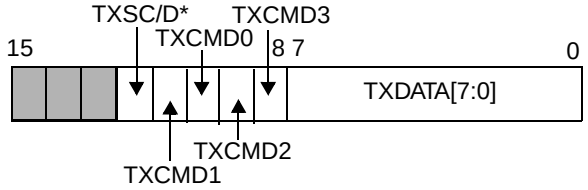
bits are used for the CY7C9689 version receive side). Therefore, two bytes are used for one byte of data transfer. The format of the transmit and receive data byte are different for the CY7C924DX version and CY7C9689 version. *Figure 4* shows the data byte formats.



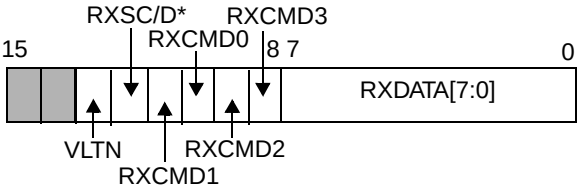
a) CY7C924 version transmit data byte format



b) CY7C924 version receive data byte format



c) CY7C9689 version transmit data byte format



d) CY7C9689 version receive data byte format

Figure 4. Data Byte Formats

In the CY7C924DX version, on the transmit side bit 8 and 9 are not valid. The evaluation board hardware does not care about the value in these two bit positions. They are recommended to be set to zero. Please refer to *Table 6* for more details on the usage of TXSC/D*, TXSOC, and TXSVS. On the receive side, bit 9 is not valid, it can be a one or a zero. Please refer to *Table 7* for the interpretation of the RXSC/D*, RXSOC and RXRVS bits.

In the CY7C9689 version, the TXSC/D* bit is used for indicating whether the TXCMD[3:0] field or TXDATA[7:0] field is valid. When TXSC/D* bit is a one, the TXCMD[3:0] field is valid, else the TXDATA[7:0] field is valid. On the receive side, the RXSC/D* has the same function. When it is a logic “1” the RXCMD[3:0] field is valid, else the RXDATA[7:0] field is valid. When the VLTN bit is a one, both RXCMD and RXDATA fields

Table 6. CY7C924DX Transmit Data Formatting

TXSOC	TXSC/D*	TXSVS	Data Format Operation
0	0	0	Normal Data Encode
0	0	1	Replace Character with C0.7 Exception
0	1	0	Normal Command Encode
0	1	1	Replace Character with C0.7 Exception
1	0	0	Send Start of Cell Marker (C8.0) + Data Character
1	0	1	Replace Character with C0.7 Exception
1	1	0	Send Extended Command Marker (C9.0) + Data Character
1	1	1	Send Serial Address Marker (C10.0) + Data Character

Table 7. Receive Data Formatting

RXSOC	RXSC/D*	RXRVS	Data Format Indication
0	0	0	Normal Data Character
0	0	1	Reserved
0	1	0	Normal Command Character
0	1	1	Received C0.7 Exception Character or Other Character Exception
1	0	0	Received Start of Cell Marker (C8.0) + Data Character
1	0	1	Received Illegal Sequence
1	1	0	Received Extended Command Marker (C9.0) + Data Character (interpreted as a command)
1	1	1	Received Serial Address Marker (C10.0) + Data Character (interpreted as an address)

are invalid. This means that the transceiver cannot decode the incoming bit stream.

Host Side Data Transfer

The Evaluation board receives data from the host on the system PCI bus. The PCI-DP handles the interaction between the host side and local side of the board. To transmit a block of data, the host CPU needs to transfer data to the shared memory of the PCI-DP. This data transfer can be done many ways. The CPU can do this transfer through a DMA transfer or a CPU controlled transfer. The data transferred must have a total size of 512 16-bit words. Also, this data block must be written into the PCI-DP starting at offset address 0x4000.

Message Passing Mailboxes

Two mailbox registers in the PCI-DP are used to communicate the states of the TXBUFFER and RXBUFFER between the host and the CPLD. The Host to Local Data Mailbox (HLDATA) is used for host to local buffer state communication. The Local to Host Data Mailbox (LHDATA) is used for local to host buffer state communication.

HLDATA Mailbox

Three bits of the HLDATA register are used. HLDATA[0] is the RXBUFF_EMPTY bit. The host writes a one to this bit when the RXBUFFER is empty (i.e., data is read from the PCI-DP already). When no data is read from the RXBUFFER, a zero must be written to this bit.

HLDATA[1] is the TXBUFF_FULL bit. The host writes a zero to this bit when the TXBUFFER is empty or no new data is present. When new data is written in the TXBUFFER, the host must write a one to this bit.

When a one is written to HLDATA[24], interrupt to local bus bit, the PCI-DP will assert the local interrupt LIRQ*, if and only if the local interrupt is enabled in the LINT register. This bit will be cleared when the LINT[3] bit is written with a one.

For example, when the host writes new data into the TXBUFFER and no data was read from the RXBUFFER, then the value 0x00010003 to the HLDATA register. This indicates to the CPLD that the TXBUFFER is full and the RXBUFFER is empty.

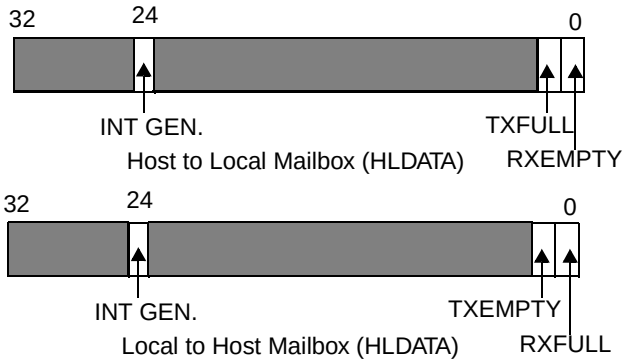


Figure 5. Dual-Port Communication Mail-

LHDATA Mailbox

The Local to Host Data Mailbox (LHDATA) register is maintained by the CPLD State Machine. Three bits of this register are used. LHDATA[0] is the RXBUFF_FULL bit. When this bit is set a “1”, there is new data available in the RXBUFFER. When set a “0”, no new data is available in the RXBUFFER.

LHDATA[1] is the TXBUFF_EMPTY bit, when this bit is written with a one, data in the TXBUFFER has been read. When this bit is written with a zero, no data was read from the TXBUFFER.

LHDATA[24] is the interrupt to host bit. When set to a “1”, INTA# will be asserted if and only if this interrupt is enabled (see Host Interrupt Control and Status Register HINT description in CY7C09449 PCI-DP datasheet) This bit will be cleared when the HINT[3] bit is set to “1”.

Interrupt Handling

Once the data block to be transmitted is in the PCI-DP's shared memory, the host needs to send a message to the local side. This is done through the Host to Local Data Mailbox (HLDATA) located at offset 0x04E8. The previous section discussed the role of this register.

The host side and the local side both share access to the PCI-DP's shared memory. The host side and local side keep track of the status of the receive and transmit buffers of the PCI-DP internally. To insure that both sides store the same state of the buffers, they share information about the buffers through the HLDATA and LHDATA registers. The HLDATA register is used by the host side to update the local side's internal flags and conversely, the LHDATA register is used by the local side to update the host side's internal flags. This type of message passing is needed because of the way data transfer takes place. Only the host side knows when it has written a block of data into the PCI-DP to transfer. So, the host side needs to update the local side. Similarly, only the local side knows when it has taken data from the PCI-DP's transmit buffer. So, the local side needs to update the host side. A similar argument applies to the PCI-DP's receive buffer.

The application program that runs on the host side controls the method by which the host flags are updated. During a transmit cycle, the data block is written to the Tx buffer in the PCI-DP. Then HLDATA register is updated with a '1' at the TXBUFF_FULL bit position and the value of the receive buffer at the RXBUFF_EMPTY bit position according to its internal state. The host program can use the LHDATA register and its own internal flags to determine the value of the receive buffer bit. Once the flag bit values are determined, the HLDATA register is updated and the interrupt bit (bit-24) needs to be set. Setting the interrupt bit causes an interrupt on the local side of the Evaluation board.

CPLD State Machine

When there is a local interrupt and the CPLD state machine is not busy, the CPLD state machine processes the interrupt. It reads the contents of the HLDATA register in the PCI-DP to determine what type of transaction is required. Then, the state machine updates its internal flags, writes to the LHDATA register to indicate the new status of the CPLD internal flags, and clears the interrupt by writing to LINT. Then the CPLD state machine goes to the idle state. If another interrupt is requested, it processes that interrupt. Otherwise, it examines the new state of its internal flags to determine what action needs to be taken. The CPLD state machine has an internal balance control scheme that divides the CPLD time between “transmits” and “receives” when there are blocks available to be transmitted and received. If a block of data needs to be transferred, the CPLD state machine triggers the transmit process in the CPLD.

The transmit process transfers a block of data from the PCI-DP's shared memory to the external transmit FIFO. If the transmit FIFO is almost full, it waits until the external FIFO's almost full flag is removed. The CPLD reads one word from the PCI-DP's transmit buffer and writes it into the external transmit FIFO. It repeats this action until all 512 words are transferred or if the transmit FIFO PAF* flag is LOW. When PAF* goes LOW, the data transfer is halted until some of the data in the transmit FIFO has been read out by the HOTLink. When PAF* eventually goes HIGH, the data transfer process is resumed.

Once the block of data has been transferred, the CPLD state machine updates the Local to Host Data Mailbox (LHDATA) in the PCI-DP located at offset 0x04F8 with the value 0x00010002 (when the RXBUFFER is not filled with new data) or 0x00010003 (when the RXBUFFER has new data). If the Local to Host Mailbox interrupt is enable, setting the interrupt bit of the LHDATA register will trigger an interrupt on the host side. If interrupts on the host side are not enabled, the

host side needs to poll this register to determine when the data block has been taken from the PCI-DP's transmit buffer.

HOTLink Transceiver

Whenever there is data available to transmit, the HOTLink transceiver transmits it. The HOTLink transceiver takes data from the external transmit FIFO and stores it in its internal 14-bit word FIFO. The data from the HOTLink's internal FIFO is then transmitted serially. The CY7C924DX and CY7C9689 datasheets detailed information about the operation of the HOTLink Transceiver.

Receive Path

Overview

The evaluation board receives serial data at its serial port and this is routed to the 924 or 9689. The data is deserialized and decoded, then presented to the parallel output port of the transceiver. This data is written to the external receive FIFO. Status flags on this FIFO signal the CPLD, which transfers the data from the FIFO to the PCI-DP. A block of 512-byte data is transferred to the PCI-DP shared memory. Then the CPLD writes to the LHDATA Mailbox to inform the host that the receive buffer of the PCI-DP is now full. The host reads the status flags in this mailbox, and transfers data from the shared memory to the host memory. The HLDATA mailbox is then written to inform the local side that the receive buffer of the PCI-DP is now empty.

HOTLink Transceiver

Serial data arriving on serial ports A or B is deserialized, decoded, framed and placed in the internal receive FIFO. For more details, refer to the CY7C924 and CY7C9689 data sheets. This data is automatically transferred to the external receive FIFO. The control/status bits are used for data flow control by the CPLD.

CPLD Receive State Machine

As soon as data is placed in this FIFO, the EF* (empty flag) is deasserted. If the CPLD is in an idle state, the receive state machine begins to move data from the receive FIFO to the PCI-DP. Data is written starting at offset 0x6000. Blocks of 512 words are transferred from the RXFIFO to the PCI-DP at a time. Each data word is transferred in turn to the local data bus and then into the shared memory on the following clock cycle. One byte is transferred to the data bus while another is enabled out of the FIFO. Thus, only two bytes are being processed at any one time. If no data is available in the FIFO, the state machine waits until data is available before transferring another word. This procedure continues until 512 words are transferred to the PCI-DP.

Local Side Data Transfer

Once the data block received is in the PCI-DP's shared memory, the CPLD needs to send a message to the host side for notification. This is done by writing to the LHDATA mailbox register located at offset 0x04F8. The value 0x0001003 is written to the LHDATA register when the TXBUFFER is empty and RXBUFFER is stored with new data. Then 0x00010001 is written when data in the TXBUFFER is not read yet, and new data is available in the RXBUFFER. The HOST reads this mailbox and accesses the shared memory to transfer 512 words from the PCI-DP to the host memory. Then the host updates HLDATA mailbox register to updating status of the RXBUFFER to 'empty' by setting HLDATA[0] to a '1'. This causes a local interrupt to the CPLD (note: if this interrupt is enabled). The CPLD reads the HLDATA register and updates its internal buffer status flags. Then the CPLD writes the LHDATA register to clear the RXBUFF_FULL bit.

Flow Control

Overview

Assuming the PCI bus on the receiving board is busy, the host cannot acquire the PCI bus to read the 512-byte data block from the PCI-DP. While this board is waiting, incoming serial data can continuously fill up the external and internal RXFIFOs. Once these two FIFOs are full, and the 512-byte data block within the PCI-DP is still not serviced, then the subsequent incoming data will be lost.

One solution to this problem is to make sure that the host services the PCI-DP RXBUFFER before the on-board FIFOs are full. This must be implemented by the host software. This solution applies to the CY7C9689 version evaluation board. Another solution to this problem is to notify the transmitting-end to stop transmission of data before the FIFOs on the receiving board are full. This hardware flow control mechanism is implemented on the CY7C924DX version evaluation board and transmission can continue.

Hardware Flow Control Implementation (CY7C924DX only)

Three special-function pins available only on the CY7C924DX are used to implement this hardware flow control mechanism. On the receiving end, the TXINT pin is used to send an "interrupt" signal to the transmitting end to stop data transmission when the receiving end RXFIFO PAF* (Programmable Almost Full) flag asserts. On the transmitting end, the RXINT output reflects the "interrupt" sent by the receiving-end. Once the "interrupt" signal is received by the transmitting end (RXINT is asserted), the TXHALT* input will be asserted to halt data transmission at the CY7C924DX immediately. Once TXHALT* is asserted, the serial link will be filled with K28.5 SYNC characters until TXHALT* is released.

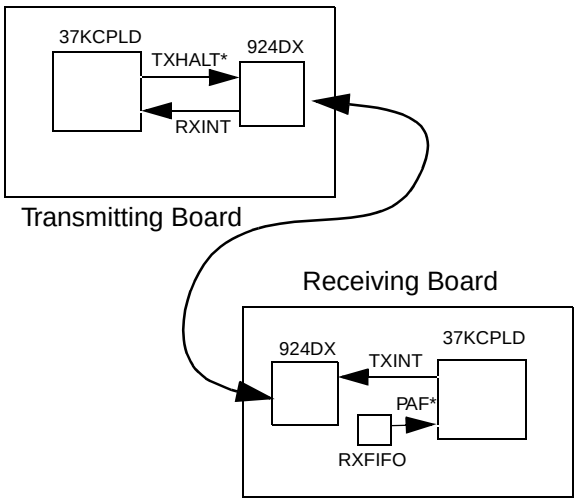


Figure 6. Hardware Flow Control Implementation

Status Indications

On both the CY7C924DX and CY7C9689 Evaluation Boards there are two types of board status displays. The two status indications are power and link status.

The two of the six LEDs located on the top side of the board are used in this application. A green LED, D306, is used for board power indication. When the board is powered up with a 5V power supply either by the PCI bus or the external power supply (through J401), this green LED will light up. The red LED, D302, is used for link status indication. This LED reflects the state of the LFI output of the transceiver. The red LED will light when:

1.When no cable is plugged-in on the receive side or,

2. there is no signal present at the selected input or,

3. the signal is out of frequency range or,

4. the signal does not have enough transition density

Serial Interfaces

The CY7C924DX Evaluation Board has three serial interface options: Unshielded Twisted Pair (UTP), Coaxial Cable (Coax), and Multimode Optical Fiber. The CY7C9689 Evaluation Board has two serial interface options: UTP and Coax. The serial output drivers and input receivers on the CY7C924DX and the CY7C9689 are identical in design. Therefore, the serial interface design on both versions of the Evaluation Boards are identical. The following sections discuss in detail each serial interface option.

Unshielded Twisted Pair (UTP) Serial I/O

There are two RJ-45 ports on evaluation boards with the UTP option. The UTP Option is designed to work with standard 100Ω CAT 5 UTP ethernet cabling. Each RJ-45 port contains two active pairs. One pair is used for transmit and the other pair is used for receive. On J301(RJ-45) pin 3 and 6 are connected to OUTB±, and pin 1 and 2 are connected INB±. On

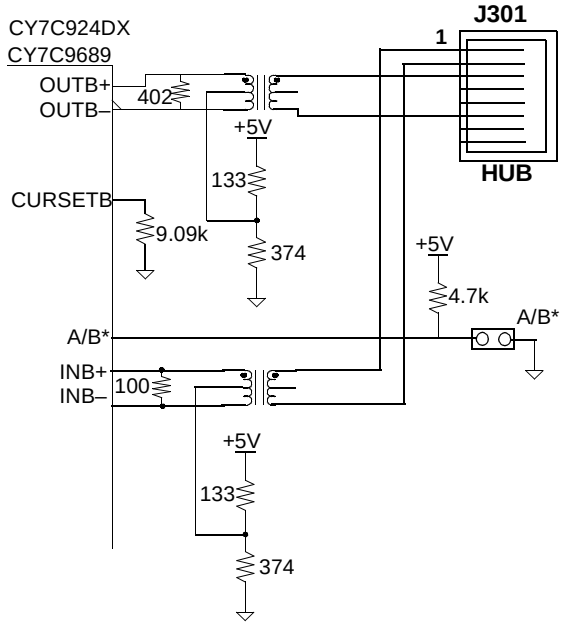


Figure 7. J301 UTP Interface

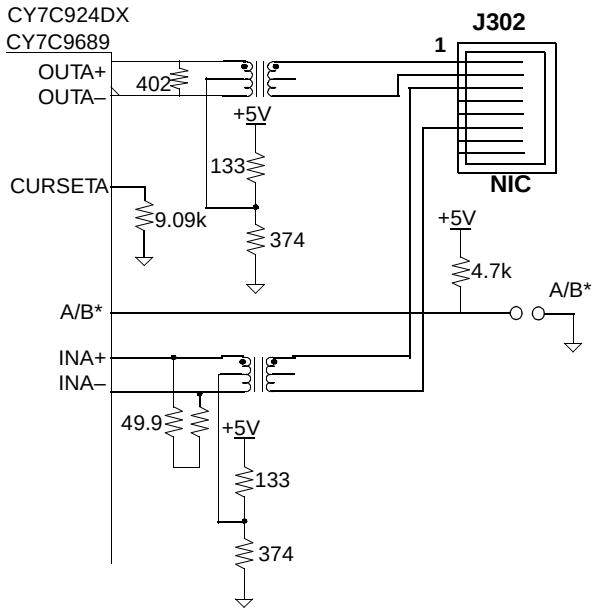


Figure 8. J302 UTP Interface

J302, pin 1 and 2 are connected to OUTA±, and pin 3 and 6 are connected to INA±, respectively.

The pin assignment of J301 is identical to 10/100BASE Ethernet HUB side; it is labelled “HUB”. Pin assignment of J302 is identical to the NIC side interface; it is labelled “NIC.” Therefore conventional ethernet cables can be used to connect the evaluation boards. Using a “straight through” ethernet cable, one end of the cable should be connected to the “HUB” RJ-45; the other end of the cable should be connected to the “NIC” RJ-45. On the evaluation board which the HUB RJ-45 port is used, the A/B* jumper on jumper block J201 should be installed.

The serial outputs and inputs are transformer coupled. The transformer used is a dual center-tapped transformer. The OUTA± and INA± shares the same dual transformers X301. OUTB± and INB± share dual transformers X302. The CURSET resistors are chosen for 100Ω balanced cabling. The CURSET resistor value is calcuated as follows:

$$R_{CURSET} = \frac{90 \times \left(\frac{400\Omega \times 100\Omega}{400\Omega + 100\Omega}\right)}{0.8V} = 9k\Omega \qquad \text{Eq. 1}$$

Coaxial (Coax) Serial I/O

With the coaxial option, the evaluation board is configured to drive and receive from 75Ω coaxial cable. The OUTB± output pair is used to drive the coax cable, and the INA± input is used to receive a signal from the coax cable.

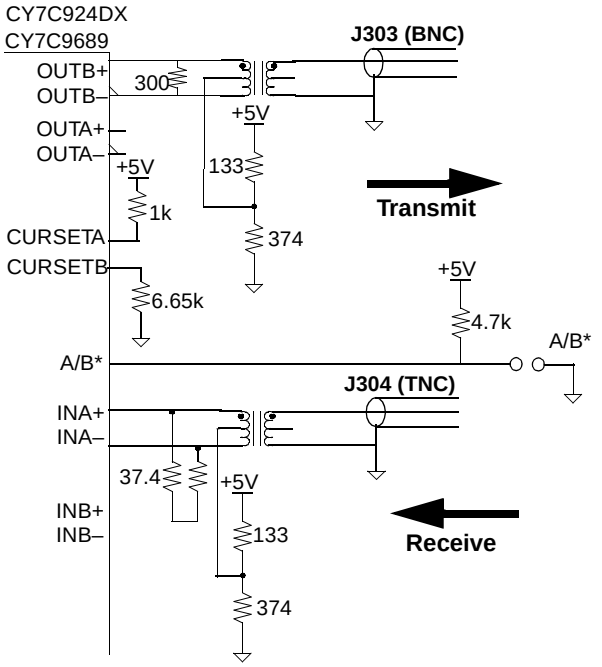


Figure 9. Coax Interface

The OUTA± output pair is not used in this option. The CURSETA input is pulled-up through a 1k resistor to shut down the OUTA± current source. This minimizes power dissipated by the OUTA+/- outputs. The INA± input must be selected in this option. The A/B* jumper at jumper block J201 must not be installed for this option.

Center-tapped transformers are also used on this interface. OUTB± is coupled through the X302 transformer. INB± is coupled through X301. The transformers play two roles, on the transmit side, they transform a differential signal into a single-ended signal. On the receive side, the transformer is acting as a Balun, transforming the single-ended signaling to differential. The CURSETA input is pulled up to V_{DD} through a 1-kΩ resistor. This disables the OUTA± outputs. CURSETB is set with a 6.65-kΩ resistor for 75Ω unbalanced cabling. *Equation2* shows the calculation for the CURSET value.

$$R_{CURSET} = \frac{180 \times \left(\frac{150\Omega \times 37.5\Omega}{150\Omega + 37.5\Omega}\right)}{0.8V} = 6.75k\Omega \qquad \text{Eq. 2}$$

Multimode Optical Fiber (OPT) Interface

The evaluation board with the fiber-optic option implements a fiber-optic-based serial interface. This interface uses industry-standard LED-based fiber-optic modules that accept SC-type fiber-optic connectors.

The fiber-optic option is designed for the de facto standard 1X9 or “Endfire” optical modules. The optical module selected is a LED-base 1300-nm module. This module is compatible with Multi-Mode Fiber with 50-µm or 62.5-µm core and 125-µm cladding. This fiber-optic module is compatible with SC-type connectors.

With the fiber-optic option, the OUTB± outputs are used to drive the optical transmitter and the INA± inputs are used for receiving PECL signals from the optical receiver. The circuit diagram is shown in *Figure 10*.

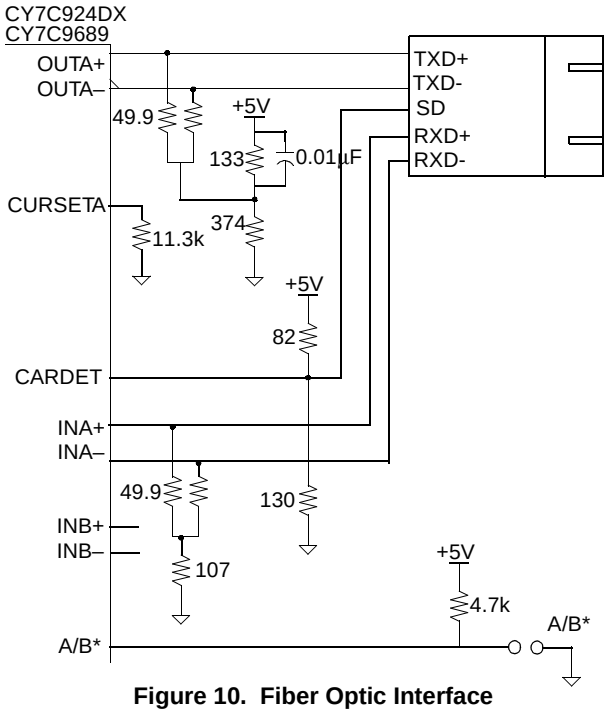


Figure 10. Fiber Optic Interface

Table 8. Bill of Materials for 924 Optical Option

Item	Quantity	Description	Reference Designator	Manufacturer	Part Number
1	41	0.01 µF (0805)	C302 C411 C412 C413 C414 C415 C416 C417 C418 C419 C421 C422 C423 C424 C425 C426 C427 C428 C430 C431 C433 C434 C435 C436 C437 C438 C439 C440 C445 C446 C447 C448 C449 C450 C451 C452 C453 C454 C455 C456 C472	AVX Murata	08055C103KATMA GRM40X7R103K050AL
2	27	0.1 µF (0805)	C101 C103 C104 C201 C303 C304 C305 C306 C404 C405 C407 C408 C409 C410 C420 C459 C460 C461 C462 C463 C464 C465 C466 C467 C468 C469 C470	AVX	08055C104KATMA
3	7	10 µF, 10V (3528)	C102 C441 C442 C443 C444 C471 C473	AVX	TAJB106M016R
4	2	47 µF (3528)	C457 C458		
5	2	1.0 µH (1210)	L301 L302	DELEVAN KOA	1210-102K KL32TE1R0J
6	9	0 ohm (0805)	R204 R222 R224 R226 R233 R326 R327 R337 R338	AVX	CJ21-000JT
7	1	0 ohm (0603)	R430		
8	8	39 ohm (0805)	R401 R402 R405 R406 R407 R408 R409 R410	DALE	CRCW0805-390JRT1
9	14	49.9 ohm 1% (0805)	R321 R322 R341 R342 R343 R344 R345 R346 R347 R348 R349 R350 R351 R352	ROHM	MCR10EZHF49R9
10	2	49.9 ohm 1% (1206)	R339 R358		
11	1	82ohm(1206)	R357		
12	2	90.9 (1210)	R228 R230		
13	1	90.9 (1206)	R404		
14	1	100 ohm 1% (0805)	R103	AVX	CR32-1000FT
15	1	133 ohm (0805)	R328		
16	1	133 ohm (1206)	R359		
17	1	107 ohm 1% (0805)	R323	KOA	RK73H2A1070F

Table 8. Bill of Materials for 924 Optical Option (Continued)

18	1	130 ohm (0805)	R340		
19	2	130 ohm 1% (1210)	R227 R229	KOA	RK73H2E1300F
20	1	130 ohm (1206)	R403		
21	1	374 ohm (1206)	R360		
22	1	402 ohm (0805)	R329		
23	17	510 ohm (0603)	R412 R413 R414 R415 R416 R417 R418 R419 R420 R421 R422 R423 R424 R425 R426 R427 R429	AVX	CR10-511JT
24	23	510 ohm (0805)	R102 R205 R206 R207 R208 R209 R210 R211 R212 R213 R214 R215 R216 R217 R218 R219 R220 R301 R302 R303 R304 R305 R306		
25	1	1K (0805)	R428	AVX	CR21-102JT
26	10	4.7K (0805)	R101 R231 R307 R308 R309 R310 R311 R312 R313 R314	DALE AVX	CRCW0805-472JRT1 CR21-472JT
27	2	11.3K (0805)	R201 R202		
28	1	100K (0805)	R232	AVX	CR21-104JT
29	5	4.7K Bussed R-Pack	RP101 RP102 RP104 RP201 RP202	BOURNS DIGI-KEY	4816P-2-472 4816P-2-472-ND
30	1	Schottky Di-ode	D401	MICROSEMI DIGI-KEY	SK33MSCT SK33MSCT-ND
31	1	Amber LED	D301	DIGI-KEY	160-1185-1-ND
32	1	Red LED	D302	DIGI-KEY	160-1186-1-ND
33	1	Orange LED	D303	DIGI-KEY	160-1187-1-ND
34	2	Green LED	D304 D306	DIGI-KEY	160-1188-1-ND
35	1	Yellow LED	D305	DIGI-KEY	160-1189-1-ND
36	1	7-Segment Display	D201	LUMEX DIGI-KEY	LDD-A514RI 67-1455-ND
37	1	20 Mhz 1/2-sz Osc	Y402	CTS DIGI-KEY VALP FISH	MXO-45HST-20.000 CTX199-ND VF70H-1T20.0MHZ
38	1	24 MHz 1/2-sz Osc	Y401	CTS DIGI-KEY	MXO-45HST-24.000 CTX250-ND
39	2	Osc Socket	Y401X Y402X	ARIES DIGI-KEY	1108800 A463-ND

Table 8. Bill of Materials for 924 Optical Option (Continued)

40	1	CY37512 (QFP208-3) OR CY37512 (QFP208-3)	U103	CYPRESS	CY37512P208-125NC OR CY37256P208-125NC
41	1	74FCT16244	U204	TI	74FCT16244ETPAC (OPTIONAL)
42	1	CY7B991	U401	CYPRESS	CY7B991-5JC
43	2	CY7C4275	U201 U202	CYPRESS	CY7C4275-10ASC
44	1	CY7C924DX	U203	CYPRESS	CY7C924DX-AC
45	1	SERIAL EE-PROM	U102	MICROCHIP	24LC02B/SN
46	1	PCI-DP	U101	CYPRESS	CY7C09449
47	1	Reset Chip	U106	TI	TL7705BID
48	1	Fiber Trans-ceiver	U302	HP	HFBR-5302
49	1	Voltage Regu-lator	U402	LINEAR	LT1763 CS8 3.3 SO-8
50	1	3.15-Amp Fuse	F401	WICKMANN DIGI-KEY	3951315044 WK4324BK-ND
51	1	Fuse Socket	F401X	WICKMANN DIGI-KEY	562SM WK0010-ND
52	2	SPST Switch	S101 S201	C&K DIGI-KEY	KT11P2CM CKN4051-ND
53	1	8-Pos. DIP-Switch	S301	CTS DIGI-KEY	206-8ST CT2068ST-ND
54	6	Probe Jack	J206 J207 J208 J209 J210 J211	JOHNSON DIGI-KEY	129-0701-202 J570-ND
55	1	Shroud 5x2 Header	J402	AMP DIGI-KEY	104339-1 104339-1-ND
56	1	2x1 Header	J405	AMP DIGI-KEY	103783-1 103783-1-ND
57	1	3x1 Header	J404	AMP DIGI-KEY	103747-3 103747-3-ND
58	3	6x2 Header	J201 J202 J203	AMP	103783-6
59	3	8x2 Header	J102 J204 J205	"AMP	
DI-GI-KEY	""103783-8				
103783-8-ND"					

Table 8. Bill of Materials for 924 Optical Option (Continued)

60	1	Power Connector	J401	AMP	171826-3
61	1	SMA Connector	J403	"AMP	
EF JOHNSON					
DIGI-KEY	""221789-1				
142-0701-201					
221789-1-ND"					
62	1	Power Cable: consists of an AMP #171822-3 housing and 3 AMP #170204-1 pins			
63	1	Simplex M.M. SC Fiber		Cabling System Warehouse	800-SSC-SSCM-2M

Table 9. Bill of Materials for 924 Coax Option

Item	Quantity	Description	Reference Designator	Manufacturer	Part Number
1	40	0.01 µF (0805)	C418 C419 C421 C422 C423 C424 C425 C426 C427 C428 C430 C431 C433 C434 C435 C436 C437 C438 C439 C440 C445 C446 C447 C448 C449 C450 C451 C452 C453 C454 C455 C456 C472	AVX MURATA	08055C103KATMA GRM40X7R103K050AL
2	1	0.068 µF (1825)	C301	AVX FAHRNER – MILLER	1825AC683KATME 1825AC683KATME
3	24	0.1 µF (0805)	C101 C103 C104 C201 C306 C404 C405 C407 C408 C409 C410 C420 C459 C460 C461 C462 C463 C464 C465 C466 C467 C468 C469 C470	AVX	08055C104KATMA
4	7	10 µF, 10V (3528)	C102 C441 C442 C443 C444 C471 C473	AVX	TAJB106M016R
5	2	47 µF (3528)	C457 C458		
6	11	0 ohm (0805)	R204 R222 R224 R226 R233 R324 R325 R333 R334 R335 R336	AVX	CJ21-000JT
7	1	0 ohm (0603)	R430		
8	8	39ohm(0805)	R401 R402 R405 R406 R407 R408 R409 R410	DALE	CRCW0805-390JRT1
9	2	37.5 ohm 1% (0805)	R321 R322		
10	12	49.9 ohm 1% (0805)	R341 R342 R343 R344 R345 R346 R347 R348 R349 R350 R351 R352	ROHM	MCR10EZHF49R9
11	4	75 ohm 1% (0805)	R319 R320 R331 R332	DALE	CRCW0805-75R0FRT1
12	1	82 ohm (1206)	R357		
13	2	90.9 (1210)	R228 R230		
14	1	90.9 (1206)	R404		
15	1	100 ohm 1% (0805)	R103	AVX	CR32-1000FT
16	1	100 ohm (1206)	R315		
17	4	133 ohm (0805)	R316 R328 R353 R355		

Table 9. Bill of Materials for 924 Coax Option (Continued)

18	2	130 ohm 1% (1210)	R227 R229	KOA	RK73H2E1300F
19	1	130 ohm (1206)	R403		
20	2	374 ohm (0805)	R354 R356		
21	1	374 ohm (1206)	R360		
22	2	402 ohm (0805)	R317 R329		
23	1	40 ohm (1206)	R330		
24	1	301 ohm (1206)	R318		
25	17	510 ohm (0603)	R412 R413 R414 R415 R416 R417 R418 R419 R420 R421 R422 R423 R424 R425 R426 R427 R429	AVX	CR10-511JT
26	23	510 ohm (0805)	R102 R205 R206 R207 R208 R209 R210 R211 R212 R213 R214 R215 R216 R217 R218 R219 R220 R301 R302 R303 R304 R305 R306		
27	2	1K (0805)	R234 R428	AVX	CR21-102JT
28	10	4.7K (0805)	R101 R231 R307 R308 R309 R310 R311 R312 R313 R314	DALE AVX	CRCW0805-472JRT1 CR21-472JT
29	1	6.65K (0805)	R201		
30	1	100K (0805)	R232	AVX	CR21-104JT
31	5	4.7K Bussed R-Pack	RP101 RP102 RP104 RP201 RP202	BOURNS DIGI-KEY	4816P-2-472 4816P-2-472-ND
32	1	Schottky Diode	D401	MICROSEMI DIGI-KEY	SK33MSCT SK33MSCT-ND
33	2	Transformer	X301 X302	PULSE ENG. FAHRNER – MILLER	PE68517L PE68517L
35	1	Amber LED	D301	DIGI-KEY	160-1185-1-ND
36	1	Red LED	D302	DIGI-KEY	160-1186-1-ND
37	1	Orange LED	D303	DIGI-KEY	160-1187-1-ND
38	2	Green LED	D304 D306	DIGI-KEY	160-1188-1-ND
39	1	Yellow LED	D305	DIGI-KEY	160-1189-1-ND
40	1	7-Segment Display	D201	LUMEX DIGI-KEY	LDD-A514RI 67-1455-ND

Table 9. Bill of Materials for 924 Coax Option (Continued)

41	1	20 Mhz 1/2-sz Osc	Y402	CTS DIGI-KEY VALP FISH	MXO-45HST-20.000 CTX199-ND VF70H-1T20.0MHZ
42	1	24 MHz 1/2-sz Osc	Y401	CTS DIGI-KEY	MXO-45HST-24.000 CTX250-ND
43	2	Osc Socket	Y401X Y402X	ARIES DIGI-KEY	1108800 A463-ND
44	1	CY37512 (QFP208-3) OR CY37512 (QFP208-3)	U103	CYPRESS	CY37512P208-125NC OR CY37256P208-125NC
45	1	74FCT16244	U204	TI	74FCT16244ETPAC (OPTIONAL)
46	1	CY7B991	U401	CYPRESS	CY7B991-5JC
47	2	CY7C4275	U201 U202	CYPRESS	CY7C4275-10ASC
48	1	CY7C924DX	U203	CYPRESS	CY7C924DX-AC
49	1	SERIAL EE-PROM	U102	MICROCHIP	24LC02B/SN
50	1	PCI-DP	U101	CYPRESS	CY7C09449
51	1	Reset Chip	U106	TI	TL7705BID
52	1	Voltage Regu-lator	U402	LINEAR	LT1763 CS8 3.3 SO-8
53	1	3.15-Amp Fuse	F401	WICKMANN DIGI-KEY	3951315044 WK4324BK-ND
54	1	Fuse Socket	F401X	WICKMANN DIGI-KEY	562SM WK0010-ND
55	2	SPST Switch	S101 S201	C&K DIGI-KEY	KT11P2CM CKN4051-ND
56	1	8-Pos. DIP-Switch	S301	CTS DIGI-KEY	206-8ST CT2068ST-ND
57	6	Probe Jack	J206 J207 J208 J209 J210 J211	JOHNSON DIGI-KEY	129-0701-202 J570-ND
58	1	Shroud 5x2 Header	J402	AMP DIGI-KEY	104339-1 104339-1-ND
59	1	2x1 Header	J405	AMP DIGI-KEY	103783-1 103783-1-ND
60	1	3x1 Header	J404	AMP DIGI-KEY	103747-3 103747-3-ND
61	3	6x2 Header	J201 J202 J203	AMP	103783-6
62	3	8x2 Header	J102 J204 J205	AMP DIGI-KEY	103783-8 103783-8-ND

Table 9. Bill of Materials for 924 Coax Option (Continued)

63	1	Power Connector	J401	AMP	171826-3
64	1	SMA Connector	J403	AMP EF JOHNSON DIGI-KEY	221789-1 142-0701-201 221789-1-ND
65	1	BNC Connector	J303	AMP DIGI-KEY	413194-1 413194-1-ND
66	1	TNC Connector	J304	AMP	413506-1
67	1	RG-59 Cable BNC to TNC		Cabling System Warehouse	159-BNC-TNC-6

Table 10. Bill of Materials for 924 RJ-45

Item	Quantity	Description	Reference Designator	Manufacturer	Part Number
1	40	0.01 µF (0805)	C411 C412 C413 C414 C415 C416 C417 C418 C419 C421 C422 C423 C424 C425 C426 C427 C428 C430 C431 C433 C434 C435 C436 C437 C438 C439 C440 C445 C446 C447 C448 C449 C450 C451 C452 C453 C454 C455 C456 C472	AVX MURATA	08055C103KATMA GRM40X7R103K050AL
2	1	0.068 µF (1825)	C301	AVX FAHRNER – MILLER	1825AC683KATME 1825AC683KATME
3	24	0.1 µF (0805)	C101 C103 C104 C201 C306 C404 C405 C407 C408 C409 C410 C420 C459 C460 C461 C462 C463 C464 C465 C466 C467 C468 C469 C470	AVX	08055C104KATMA
4	7	10 µF, 10V (3528)	C102 C441 C442 C443 C444 C471 C473	AVX	TAJB106M016R
5	2	47 µF (3528)	C457 C458		
6	9	0 ohm (0805)	R204 R222 R224 R226 R233 R324 R325 R335 R336	AVX	CJ21-000JT
7	1	0 ohm (0603)	R430		
8	8	39 ohm (0805)	R401 R402 R405 R406 R407 R408 R409 R410	DALE	CRCW0805-390JRT1
9	14	49.9 ohm 1% (0805)	R321 R322 R341 R342 R343 R344 R345 R346 R347 R348 R349 R350 R351 R352	ROHM	MCR10EZHF49R9
10	4	75 ohm 1% (0805)	R319 R320 R331 R332	DALE	CRCW0805-75R0FRT1
11	1	82 ohm (1206)	R357		
12	2	90.9 (1210)	R228 R230		
13	1	90.9 (1206)	R404		
14	1	100 ohm 1% (0805)	R103	AVX	CR32-1000FT
15	1	100 ohm (1206)	R315		
16	4	133 ohm (0805)	R316 R328 R353 R355		
17	2	130 ohm 1% (1210)	R227 R229	KOA	RK73H2E1300F
18	1	130 ohm (1206)	R403		

Table 10. Bill of Materials for 924 RJ-45 (Continued)

19	2	374 ohm (0805)	R354 R356		
20	1	374 ohm (1206)	R360		
21	2	402 ohm (0805)	R317 R329		
22	2	402 ohm (1206)	R318 R330		
23	17	510 ohm (0603)	R412 R413 R414 R415 R416 R417 R418 R419 R420 R421 R422 R423 R424 R425 R426 R427 R429	AVX	CR10-511JT
24	23	510 ohm (0805)	R102 R205 R206 R207 R208 R209 R210 R211 R212 R213 R214 R215 R216 R217 R218 R219 R220 R301 R302 R303 R304 R305 R306		
25	2	1K (0805)	R428	AVX	CR21-102JT
26	10	4.7K (0805)	R101 R231 R307 R308 R309 R310 R311 R312 R313 R314	DALE AVX	CRCW0805-472JRT1 CR21-472JT
27	2	9.09K (0805)	R201 R202		
28	1	100K (0805)	R232	AVX	CR21-104JT
29	5	4.7K Bussed R-Pack	RP101 RP102 RP104 RP201 RP202	BOURNS DIGI-KEY	4816P-2-472 4816P-2-472-ND
30	1	Schottky Diode	D401	MICROSEMI DIGI-KEY	SK33MSCT SK33MSCT-ND
31	2	Transformer	X301 X302	PULSE ENG. FAHRNER – MILLER	PE68517L PE68517L
32	1	Amber LED	D301	DIGI-KEY	160-1185-1-ND
33	1	Red LED	D302	DIGI-KEY	160-1186-1-ND
34	1	Orange LED	D303	DIGI-KEY	160-1187-1-ND
35	2	Green LED	D304 D306	DIGI-KEY	160-1188-1-ND
36	1	Yellow LED	D305	DIGI-KEY	160-1189-1-ND
37	1	7-Segment Display	D201	LUMEX DIGI-KEY	LDD-A514RI 67-1455-ND
38	1	20 Mhz 1/2-sz Osc	Y402	CTS DIGI-KEY VALP FISH	MXO-45HST-20.000 CTX199-ND VF70H-1T20.0MHZ
39	1	24 MHz 1/2-sz Osc	Y401	CTS DIGI-KEY	MXO-45HST-24.000 CTX250-ND
40	2	Osc Socket	Y401X Y402X	ARIES DIGI-KEY	1108800 A463-ND

Table 10. Bill of Materials for 924 RJ-45 (Continued)

41	1	CY37512 (QFP208-3) OR CY37512 (QFP208-3)	U103	CYPRESS	CY37512P208-125NC OR CY37256P208-125NC
42	1	74FCT16244	U204	TI	74FCT16244ETPAC (OPTIONAL)
43	1	CY7B991	U401	CYPRESS	CY7B991-5JC
44	2	CY7C4275	U201 U202	CYPRESS	CY7C4275-10ASC
45	1	CY7C924DX	U203	CYPRESS	CY7C924DX-AC
46	1	SERIAL EEPROM	U102	MICROCHIP	24LC02B/SN
47	1	PCI-DP	U101	CYPRESS	CY7C09449
48	1	Reset Chip	U106	TI	TL7705BID
49	1	Voltage Regulator	U402	LINEAR	LT1763 CS8 3.3 SO-8
50	1	3.15-Amp Fuse	F401	WICKMANN DIGI-KEY	3951315044 WK4324BK-ND
51	1	Fuse Socket	F401X	WICKMANN DIGI-KEY	562SM WK0010-ND
52	2	SPST Switch	S101 S201	C&K DIGI-KEY	KT11P2CM CKN4051-ND
53	1	8-Pos. DIP-Switch	S301	CTS DIGI-KEY	206-8ST CT2068ST-ND
54	6	Probe Jack	J206 J207 J208 J209 J210 J211	JOHNSON DIGI-KEY	129-0701-202 J570-ND
55	1	Shroud 5x2 Header	J402	AMP DIGI-KEY	104339-1 104339-1-ND
56	1	2x1 Header	J405	AMP DIGI-KEY	103783-1-ND 103783-1
57	1	3x1 Header	J404	AMP DIGI-KEY	103747-3 103747-3-ND
58	3	6x2 Header	J201 J202 J203	AMP	103783-6
59	3	8x2 Header	J102 J204 J205	AMP DIGI-KEY	103783-8 103783-8-ND
60	1	Power Connector	J401	AMP	171826-3
61	1	SMA Connector	J403	AMP EF JOHNSON DIGI-KEY	221789-1 142-0701-201 221789-1-ND
62	2	RJ45 Female Conn.	J301 J302	AMP DIGI-KEY	555153-1 A9026-ND

Table 10. Bill of Materials for 924 RJ-45 (Continued)

63	1	Power Cable: consists of an AMP #171822-3 housing and 3 AMP #170204-1 pins			
64	1	CAT5 UTP 100-Ohm		Cabling System Warehouse	05A-06-A-XX

Table 11. Bill of Materials for 9689 RF-45 Option

Item	Quantity	Description	Reference Designator	Manufacturer	Part Number
1	40	0.01 µF (0805)	C411 C412 C413 C414 C415 C416 C417 C418 C419 C421 C422 C423 C424 C425 C426 C427 C428 C430 C431 C433 C434 C435 C436 C437 C438 C439 C440 C445 C446 C447 C448 C449 C450 C451 C452 C453 C454 C455 C456 C472	AVX MURATA	08055C103KATMA GRM40X7R103K050AL
2	1	0.068 µF (1825)	C301	AVX FAHRNER – MILLER	1825AC683KATME 1825AC683KATME
3	24	0.1 µF (0805)	C101 C103 C104 C201 C306 C404 C405 C407 C408 C409 C410 C420 C459 C460 C461 C462 C463 C464 C465 C466 C467 C468 C469 C470	AVX	08055C104KATMA
4	7	10 µF, 10V (3528)	C102 C441 C442 C443 C444 C471 C473	AVX	TAJB106M016R
5	2	47 µF (3528)	C457 C458		
6	11	0 ohm (0805)	R203 R221 R223 R225 R233 R235 R237 R324 R325 R335 R336	AVX	CJ21-000JT
7	1	0 ohm (0603)	R430		
8	8	39 ohm (0805)	R401 R402 R405 R406 R407 R408 R409 R410	DALE	CRCW0805-390JRT1
9	14	49.9 ohm 1% (0805)	R321 R322 R341 R342 R343 R344 R345 R346 R347 R348 R349 R350 R351 R352	ROHM	MR10EZHF49R9
10	4	75 ohm 1% (0805)	R319 R320 R331 R332	DALE	CRCW0805-75R0FRT1
11	1	82 ohm (1206)	R357		
12	2	90.9 (1210)	R228 R230		
13	1	90.9 (1206)	R404		
14	1	100 ohm 1% (0805)	R103	AVX	CR32-1000FT
15	1	100 ohm (1206)	R315		
16	4	133 ohm (0805)	R316 R328 R353 R355		
17	2	130 ohm 1% (1210)	R227 R229	KOA	RK73H2E1300F

Table 11. Bill of Materials for 9689 RF-45 Option (Continued)

18	1	130 ohm (1206)	R403		
19	2	374 ohm (0805)	R354 R356		
20	1	374 ohm (1206)	R360		
21	2	402 ohm (0805)	R317 R329		
22	2	402 ohm (1206)	R318 R330		
23	17	510 ohm (0603)	R412 R413 R414 R415 R416 R417 R418 R419 R420 R421 R422 R423 R424 R425 R426 R427 R429	AVX	CR10-511JT
24	23	510 ohm (0805)	R102 R205 R206 R207 R208 R209 R210 R211 R212 R213 R214 R215 R216 R217 R218 R219 R220 R301 R302 R303 R304 R305 R306		
25	1	1K (0805)	R428	AVX	CR21-102JT
26	10	4.7K (0805)	R101 R231 R307 R308 R309 R310 R311 R312 R313 R314	DALE AVX	CRCW0805-472JRT1 CR21-472JT
27	2	9.09K (0805)	R201 R202		
28	1	100K (0805)	R232	AVX	CR21-104JT
29	5	4.7K Bussed R-Pack	RP101 RP102 RP104 RP201 RP202	BOURNS DIGI-KEY	4816P-2-472 4816P-2-472-ND
30	1	Schottky Diode	D401	MICROSEMI DIGI-KEY	SK33MSCT SK33MSCT-ND
31	2	Transformer	X301 X302	PULSE ENG. FAHRNER – MILLER	PE68517L PE68517L
32	1	Amber LED	D301	DIGI-KEY	160-1185-1-ND
33	1	Red LED	D302	DIGI-KEY	160-1186-1-ND
34	1	Orange LED	D303	DIGI-KEY	160-1187-1-ND
35	2	Green LED	D304 D306	DIGI-KEY	160-1188-1-ND
36	1	Yellow LED	D305	DIGI-KEY	160-1189-1-ND
37	1	7-Segment Display	D201	LUMEX DIGI-KEY	LDD-A514RI 67-1455-ND
38	1	20 Mhz 1/2-sz Osc	Y402	CTS DIGI-KEY VALP FISH	MXO-45HST-20.000 CTX199-ND VF70H-1T20.0MHZ

Table 11. Bill of Materials for 9689 RF-45 Option (Continued)

39	1	24 MHz 1/2-sz Osc	Y401	CTS DIGI-KEY	MXO-45HST-24.000 CTX250-ND
40	2	Osc Socket	Y401X Y402X	ARIES DIGI-KEY	1108800 A463-ND
41	1	CY37512 (QFP208-3) OR CY37512 (QFP208-3)	U103	CYPRESS	CY37512P208-125NC OR CY37256P208-125NC
42	1	74FCT16244	U204	TI	74FCT16244ETPAC (OPTIONAL)
43	1	CY7B991	U401	CYPRESS	CY7B991-5JC
44	2	CY7C4275	U201 U202	CYPRESS	CY7C4275-10ASC
45	1	CY7C924DX	U203	CYPRESS	CY7C924DX-AC
46	1	SERIAL EEPROM	U102	MICROCHIP	24LC02B/SN
47	1	PCI-DP	U101	CYPRESS	CY7C09449
48	1	Reset Chip	U106	TI	TL7705BID
49	1	Voltage Regu-lator	U402	LINEAR	LT1763 CS8 3.3 SO-8
50	1	3.15-Amp Fuse	F401	WICKMANN DIGI-KEY	3951315044 WK4324BK-ND
51	1	Fuse Socket	F401X	WICKMANN DIGI-KEY	562SM WK0010-ND
52	2	SPST Switch	S101 S201	C&K DIGI-KEY	KT11P2CM CKN4051-ND
53	1	8-Pos. DIP-Switch	S301	CTS DIGI-KEY	206-8ST CT2068ST-ND
54	6	Probe Jack	J206 J207 J208 J209 J210 J211	JOHNSON DIGI-KEY	129-0701-202 J570-ND
55	1	Shroud 5x2 Header	J402	AMP DIGI-KEY	104339-1 104339-1-ND
56	2	2x1 Header	J405	AMP DIGI-KEY	103783-1 103783-1-ND
57	1	3x1 Header	J404	AMP DIGI-KEY	103747-3 103747-3-ND
58	3	6x2 Header	J201 J202 J203	AMP	103783-6
59	3	8x2 Header	J102 J204 J205	AMP DIGI-KEY	103783-8 103783-8-ND
60	1	Power Connector	J401	AMP	171826-3

Table 11. Bill of Materials for 9689 RF-45 Option (Continued)

61	1	SMA Connector	J403	AMP EF JOHNSON DIGI-KEY	221789-1 142-0701-201 221789-1-ND
62	2	RJ45 Female Conn.	J301 J302	AMP DIGI-KEY	555153-1 A9026-ND
63	1	Power Cable: consists of an AMP #171822-3 housing and 3 AMP #170204-1 pins			
64	1	CAT5 UTP 100-Ohm		Cabling System Warehouse	05A-06-A-XX

Table 12. Bill of Materials for 9689 Coax Option

Item	Quantity	Description	Reference Designator	Manufacturer	Part Number
1	40	0.01 µF (0805)	C411 C412 C413 C414 C415 C416 C417 C418 C419 C421 C422 C423 C424 C425 C426 C427 C428 C430 C431 C433 C434 C435 C436 C437 C438 C439 C440 C445 C446 C447 C448 C449 C450 C451 C452 C453 C454 C455 C456 C472	AVX MURATA	08055C103KATMA GRM40X7R103K050AL
2	1	0.068 µF (1825)	C301	AVX FAHRNER – MILLER	1825AC683KATME 1825AC683KATME
3	24	0.1 µF (0805)	C101 C103 C104 C201 C306 C404 C405 C407 C408 C409 C410 C420 C459 C460 C461 C462 C463 C464 C465 C466 C467 C468 C469 C470	AVX	08055C104KATMA
4	7	10 µF, 10V (3528)	C102 C441 C442 C443 C444 C471 C473	AVX	TAJB106M016R
5	2	47 µF (3528)	C457 C458		
6	13	0 ohm (0805)	R203 R221 R223 R225 R233 R235 R237 R324 R325 R333 R334 R335 R336	AVX	CJ21-000JT
7	1	0 ohm (0603)	R430		
8	2	37.5 ohm (0805)	R321 R322		
9	8	39 ohm (0805)	R401 R402 R405 R406 R407 R408 R409 R410	DALE	CRCW0805-390JRT1
10	12	49.9 ohm 1% (0805)	R341 R342 R343 R344 R345 R346 R347 R348 R349 R350 R351 R352	ROHM	MCR10EZHF49R9
11	4	75 ohm 1% (0805)	R319 R320 R331 R332	DALE	CRCW0805-75R0FRT1
12	1	82 ohm (1206)	R357		
13	2	90.9 (1210)	R228 R230		
14	1	90.9 (1206)	R404		
15	1	100 ohm 1% (0805)	R103	AVX	CR32-1000FT
16	1	100 ohm (1206)	R315		
17	4	133 ohm (0805)	R316 R328 R353 R355		

Table 12. Bill of Materials for 9689 Coax Option (Continued)

18	2	130 ohm 1% (1210)	R227 R229	KOA	RK73H2E1300F
19	1	130 ohm (1206)	R403		
20	2	374 ohm (0805)	R354 R356		
21	1	374 ohm (1206)	R360		
22	2	402 ohm (0805)	R317 R329		
23	1	402 ohm (1206)	R330		
24	1	301 ohm (1206)	R318		
25	17	510 ohm (0603)	R412 R413 R414 R415 R416 R417 R418 R419 R420 R421 R422 R423 R424 R425 R426 R427 R429	AVX	CR10-511JT
26	23	510 ohm (0805)	R102 R205 R206 R207 R208 R209 R210 R211 R212 R213 R214 R215 R216 R217 R218 R219 R220 R301 R302 R303 R304 R305 R306		
27	2	1K (0805)	R234 R428	AVX	CR21-102JT
28	10	4.7K (0805)	R101 R231 R307 R308 R309 R310 R311 R312 R313 R314	DALE AVX	CRCW0805-472JRT1 CR21-472JT
29	1	6.65K (0805)	R201		
30	1	100K (0805)	R232	AVX	CR21-104JT
31	5	4.7K Bussed R-Pack	RP101 RP102 RP104 RP201 RP202	BOURNS DIGI-KEY	4816P-2-472 4816P-2-472-ND
32	1	Schottky Diode	D401	MICROSEMI DIGI-KEY	SK33MSCT SK33MSCT-ND
33	2	Transformer	X301 X302	PULSE ENG. FAHRNER – MILLER	PE68517L PE68517L
34	1	Amber LED	D301	DIGI-KEY	160-1185-1-ND
35	1	Red LED	D302	DIGI-KEY	160-1186-1-ND
36	1	Orange LED	D303	DIGI-KEY	160-1187-1-ND
37	2	Green LED	D304 D306	DIGI-KEY	160-1188-1-ND
38	1	Yellow LED	D305	DIGI-KEY	160-1189-1-ND
39	1	7-Segment Display	D201	LUMEX DIGI-KEY	"LDD-A514RI 67-1455-ND

Table 12. Bill of Materials for 9689 Coax Option (Continued)

40	1	20 Mhz 1/2-sz Osc	Y402	CTS DIGI-KEY VALP FISH	MXO-45HST-20.000 CTX199-ND VF70H-1T20.0MHZ
41	1	24 MHz 1/2-sz Osc	Y401	CTS DIGI-KEY	MXO-45HST-24.000 CTX250-ND
42	2	Osc Socket	Y401X Y402X	ARIES DIGI-KEY	1108800 A463-ND
43	1	CY37512 (QFP208-3) OR CY37512 (QFP208-3)	U103	CYPRESS	CY37512P208-125NC OR CY37256P208-125NC
44	1	74FCT16244	U204	TI	74FCT16244ETPAC (OPTIONAL)
45	1	CY7B991	U401	CYPRESS	CY7B991-5JC
46	2	CY7C4275	U201 U202	CYPRESS	CY7C4275-10ASC
47	1	CY7C924DX	U203	CYPRESS	CY7C924DX-AC
48	1	SERIAL EE-PROM	U102	MICROCHIP	24LC02B/SN
49	1	PCI-DP	U101	CYPRESS	CY7C09449
50	1	Reset Chip	U106	TI	TL7705BID
51	1	Voltage Regu-lator	U402	LINEAR	LT1763 CS8 3.3 SO-8
52	1	3.15-Amp Fuse	F401	WICKMANN DIGI-KEY	3951315044 WK4324BK-ND
53	1	Fuse Socket	F401X	WICKMANN DIGI-KEY	562SM WK0010-ND
54	2	SPST Switch	S101 S201	C&K DIGI-KEY	KT11P2CM CKN4051-ND
55	1	8-Pos. DIP-Switch	S301	CTS DIGI-KEY	206-8ST CT2068ST-ND
56	6	Probe Jack	J206 J207 J208 J209 J210 J211	JOHNSON DIGI-KEY	129-0701-202 J570-ND
57	1	Shroud 5x2 Header	J402	AMP DIGI-KEY	104339-1 104339-1-ND
58	2	2x1 Header	J405	AMP DIGI-KEY	103783-1 103783-1-ND
59	1	3x1 Header	J404	AMP DIGI-KEY	103747-3 103747-3-ND
60	3	6x2 Header	J201 J202 J203	AMP	103783-6
61	3	8x2 Header	J102 J204 J205	AMP DIGI-KEY	103783-8 103783-8-ND

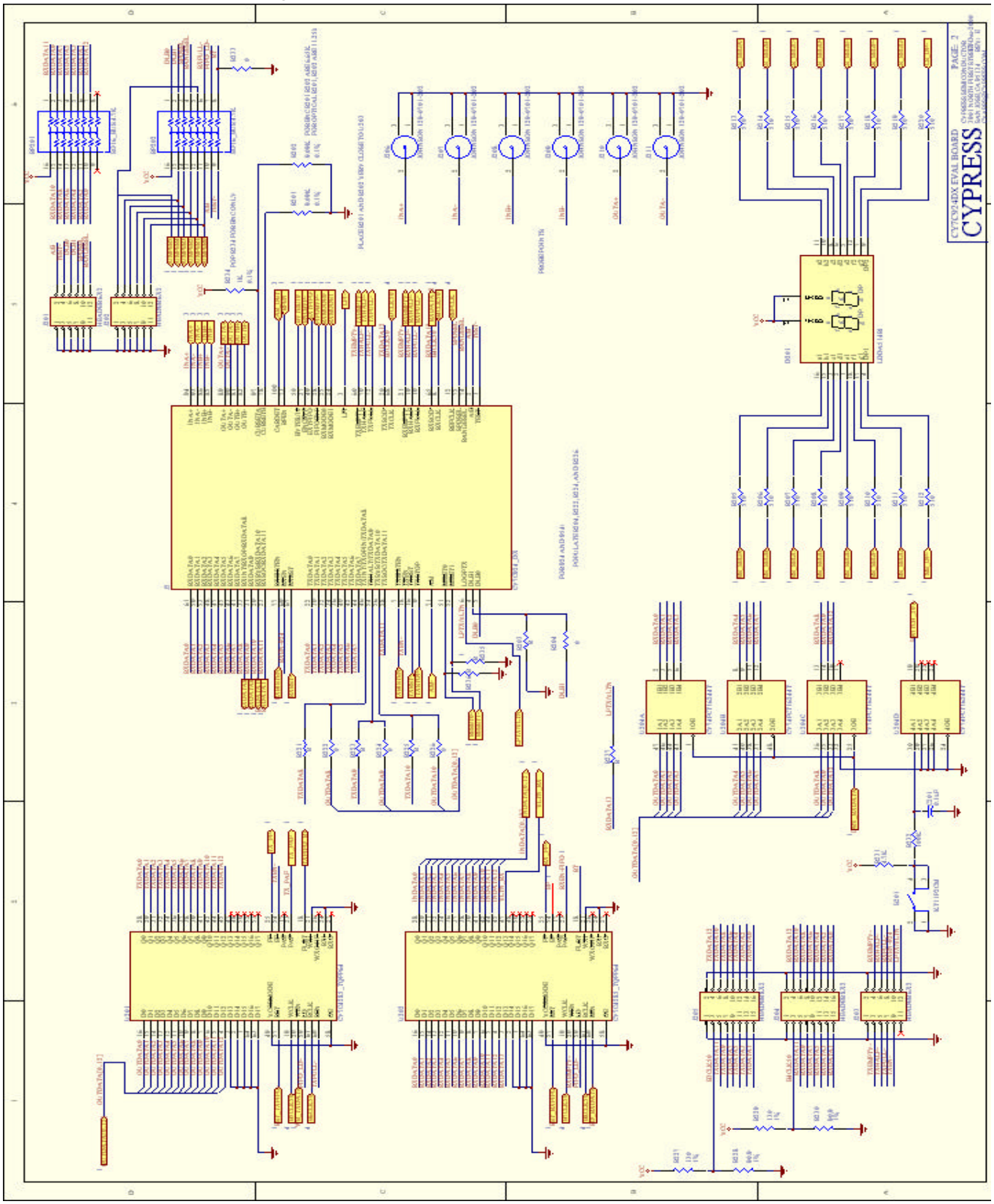
Table 12. Bill of Materials for 9689 Coax Option (Continued)

62	1	Power Connector	J401	AMP	171826-3
63	1	SMA Connector	J403	AMP EF JOHNSON DIGI-KEY	221789-1 142-0701-201 221789-1-ND
64	1	BNC Connector	J303	AMP DIGI-KEY	413194-1 413194-1-ND
65	1	TNC Connector	J304	AMP	413506-1
66	1	Power Cable: consists of an AMP #171822-3 housing and 3 AMP #170204-1 pins			
67	1	RG-59 Cable BNC to TNC		Cabling System Warehouse	159-BNC-TNC-6

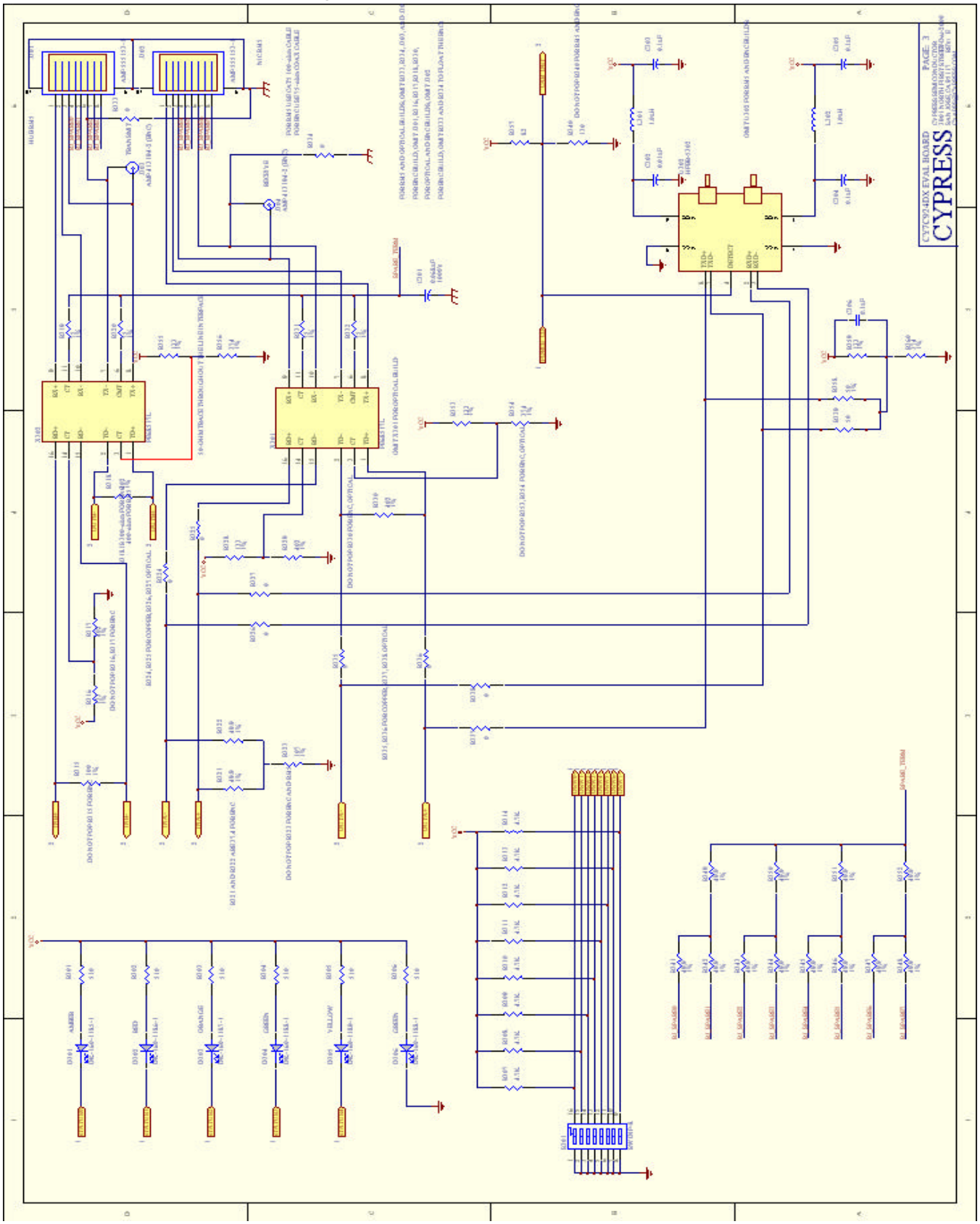
Schematics

CY7C924DX Evaluation Schematic Page 1

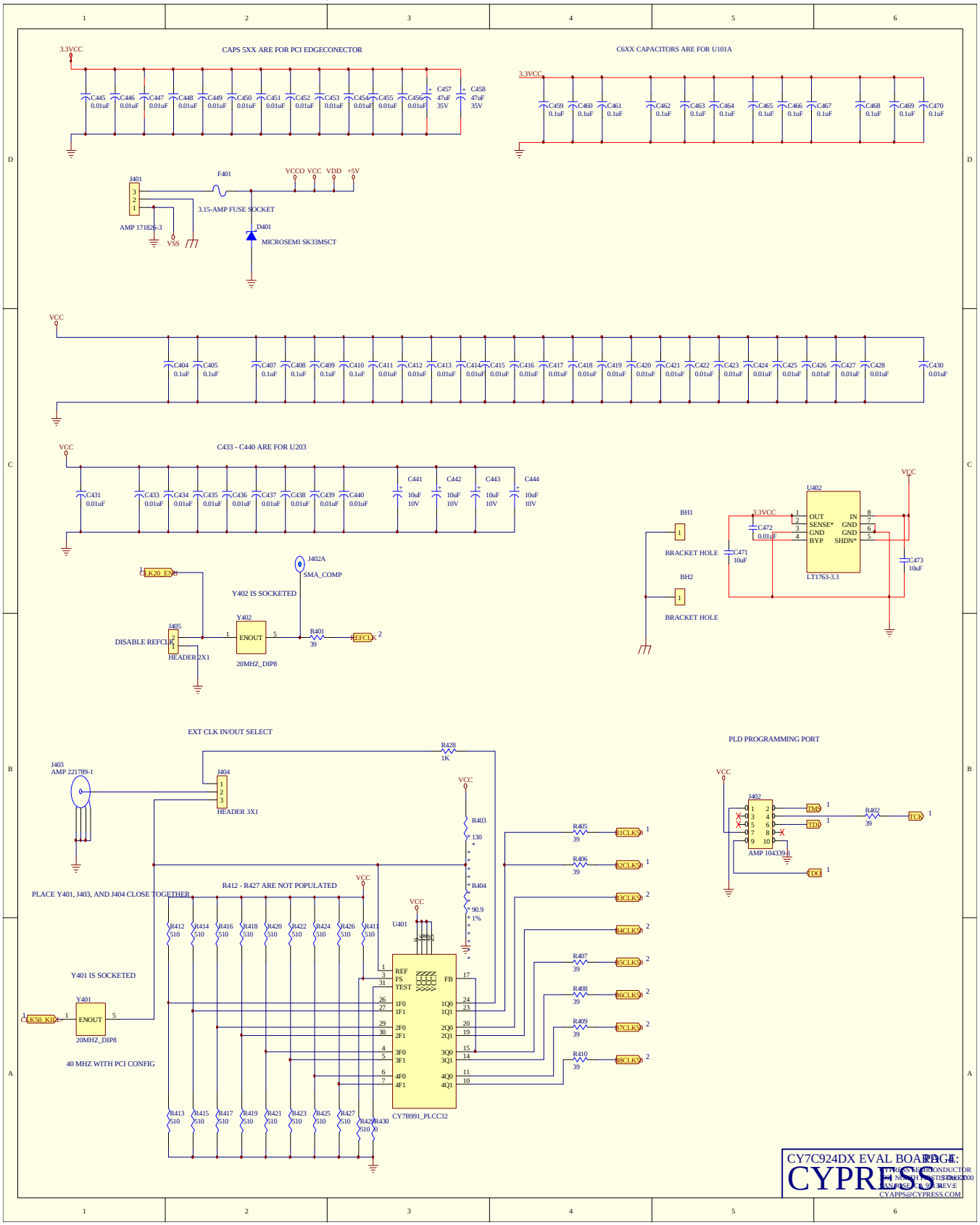
CY7C924DX Evaluation Board Schematic Page 2



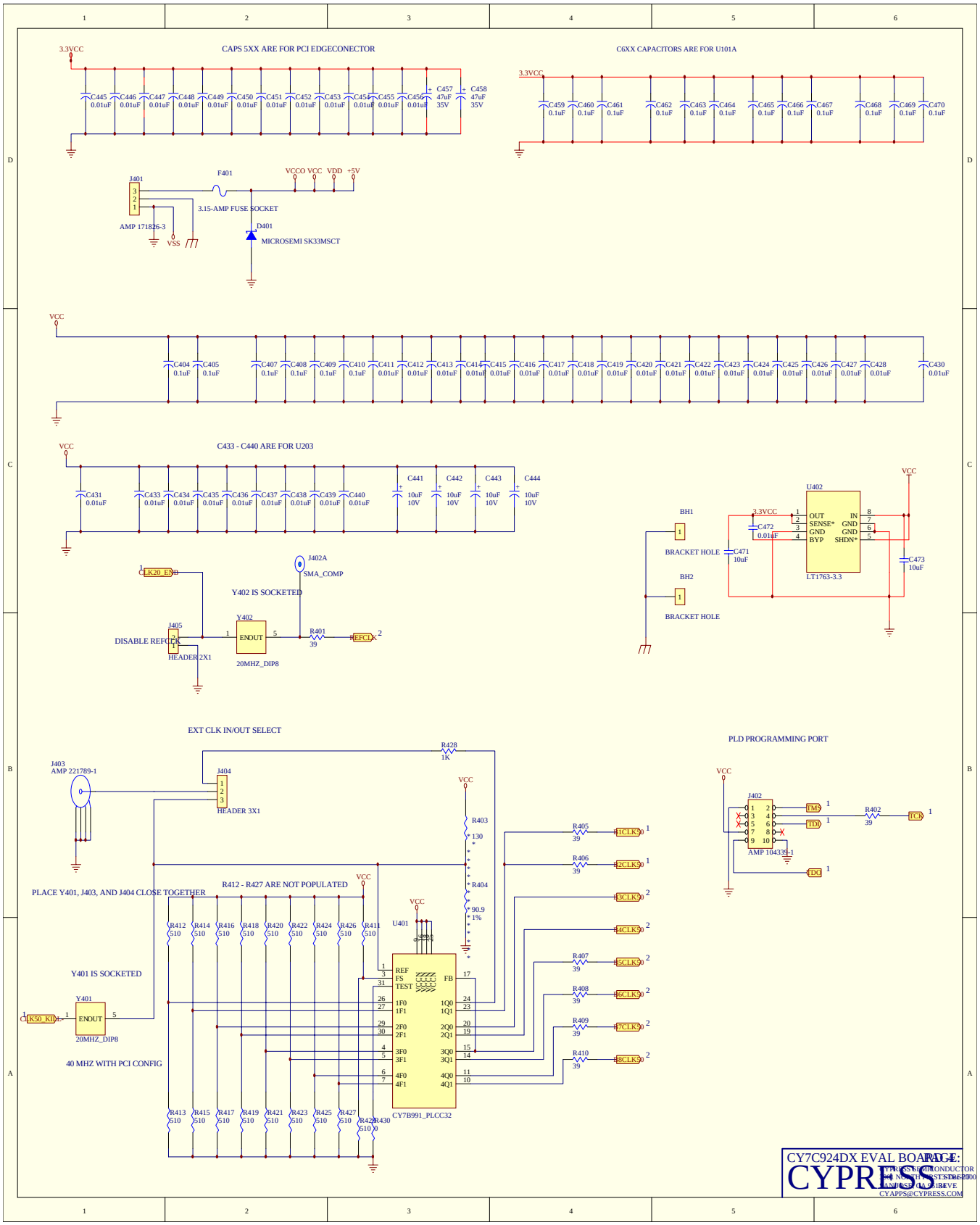
CY7C924DX Evaluation Board Schematic Page 3



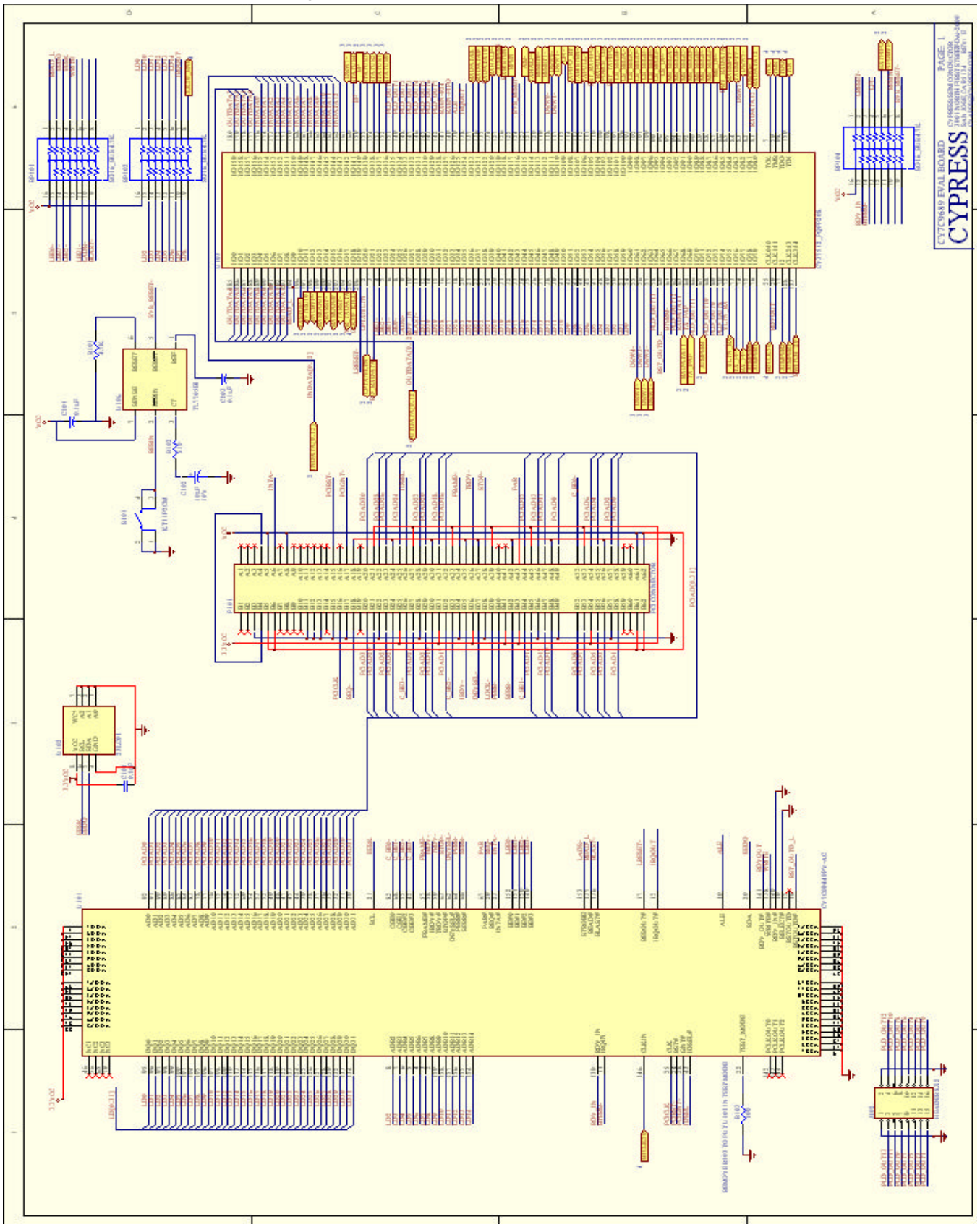
CY7C924DX Evaluation Board Schematic Page 4



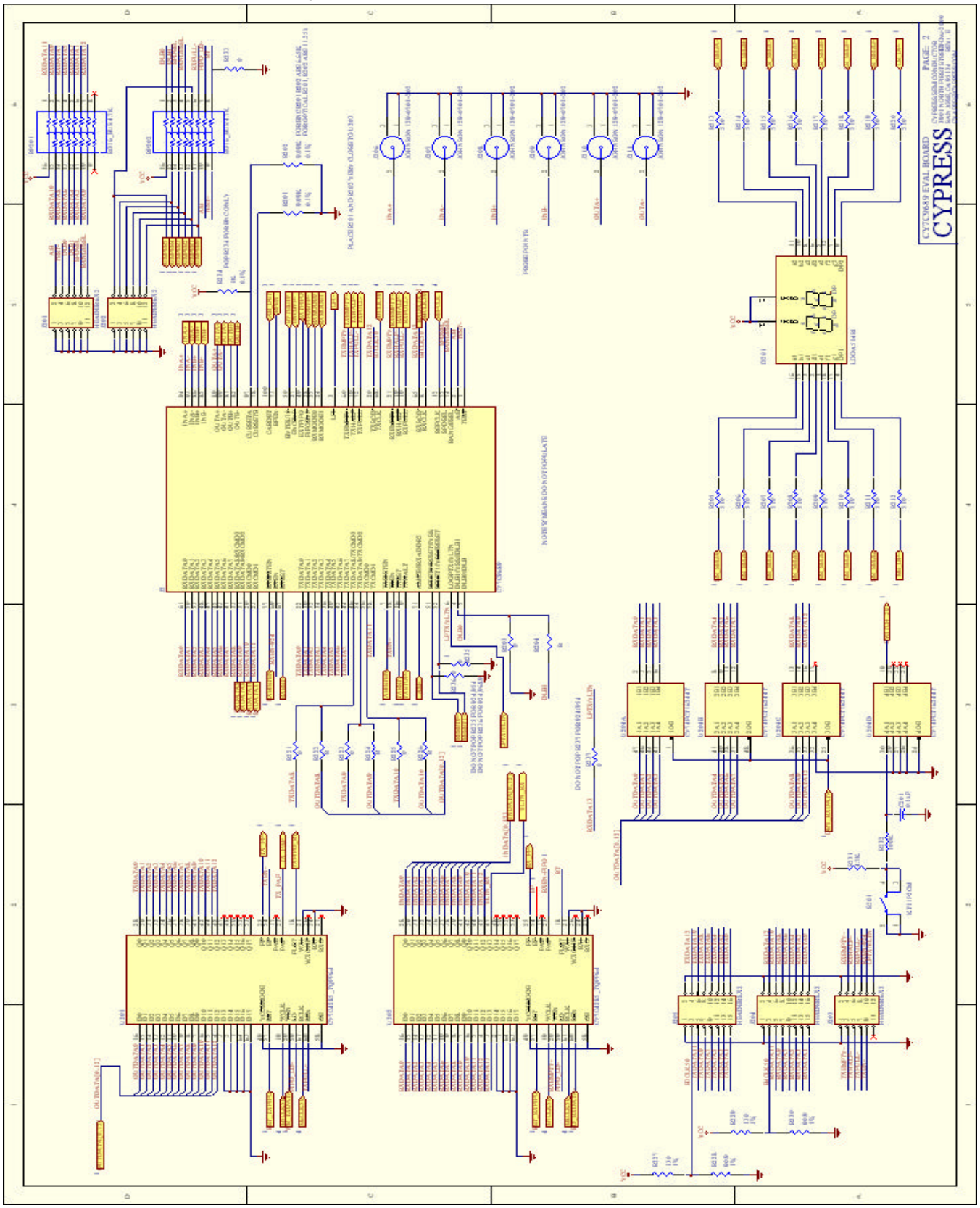
CY7C924DX Evaluation Board Schematic Page 4



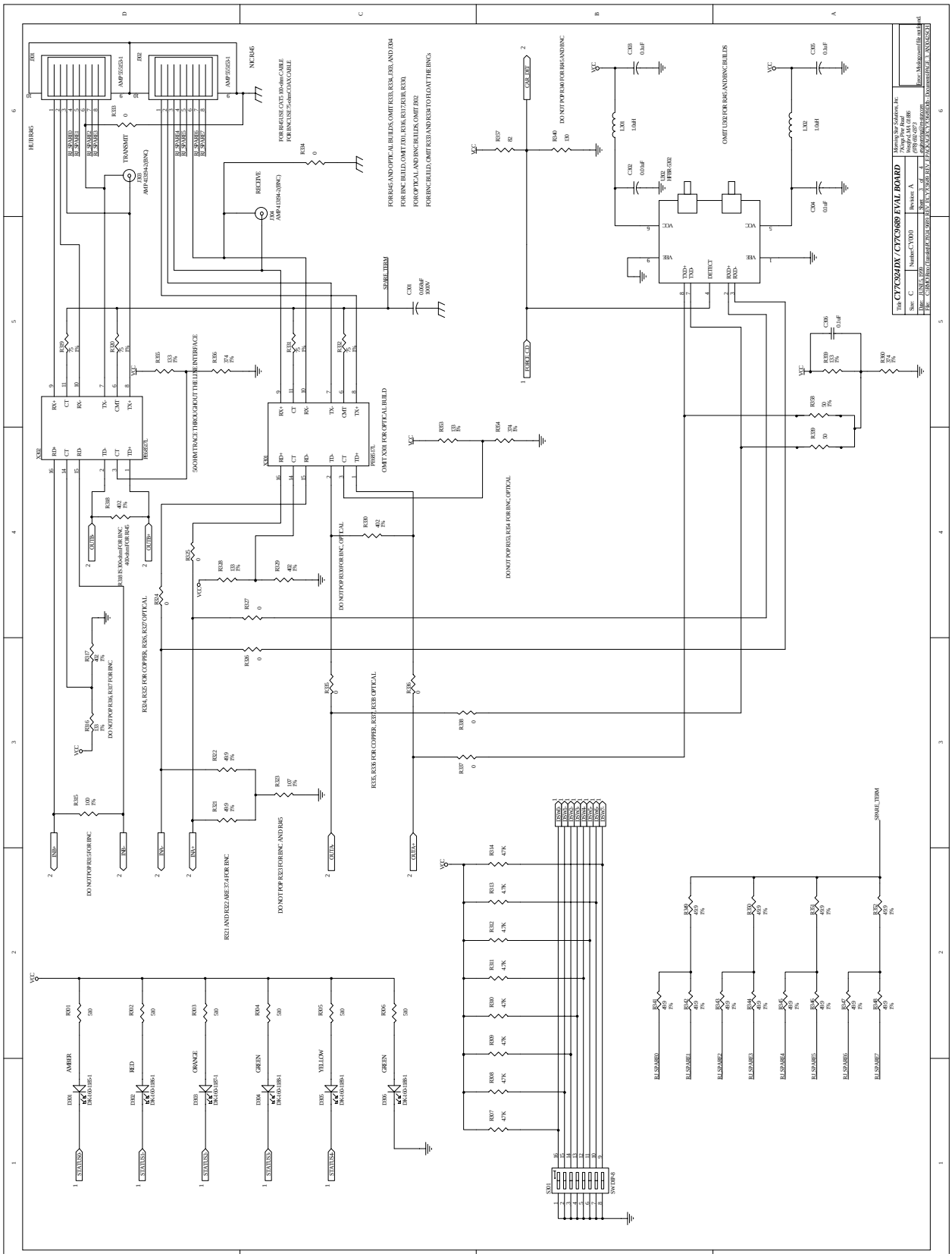
CY7C9689 Evaluation Board Schematic Page 1



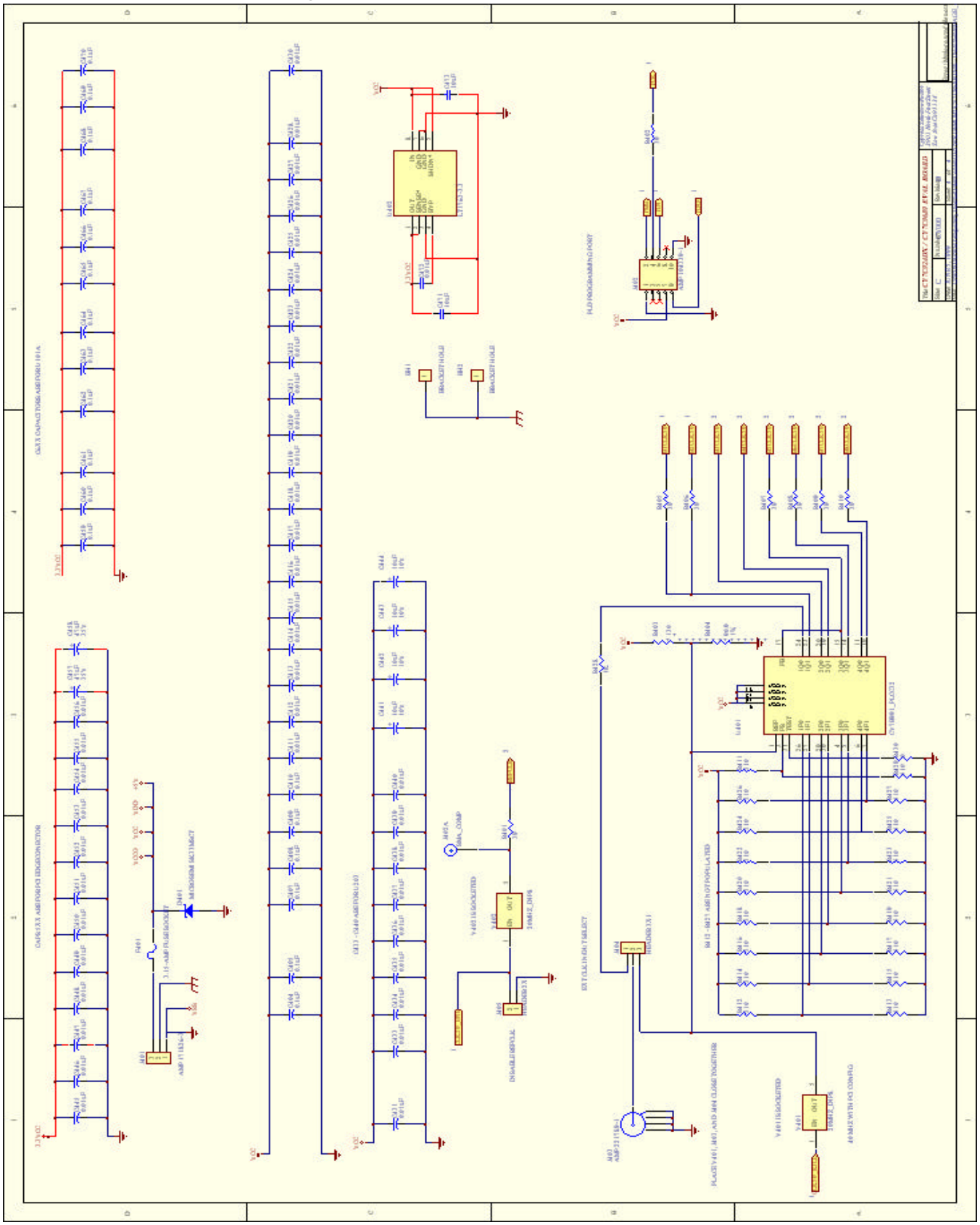
CY7C9689 Evaluation Board Schematic Page 2



CY7C9689 Evaluation Board Schematic Page 3

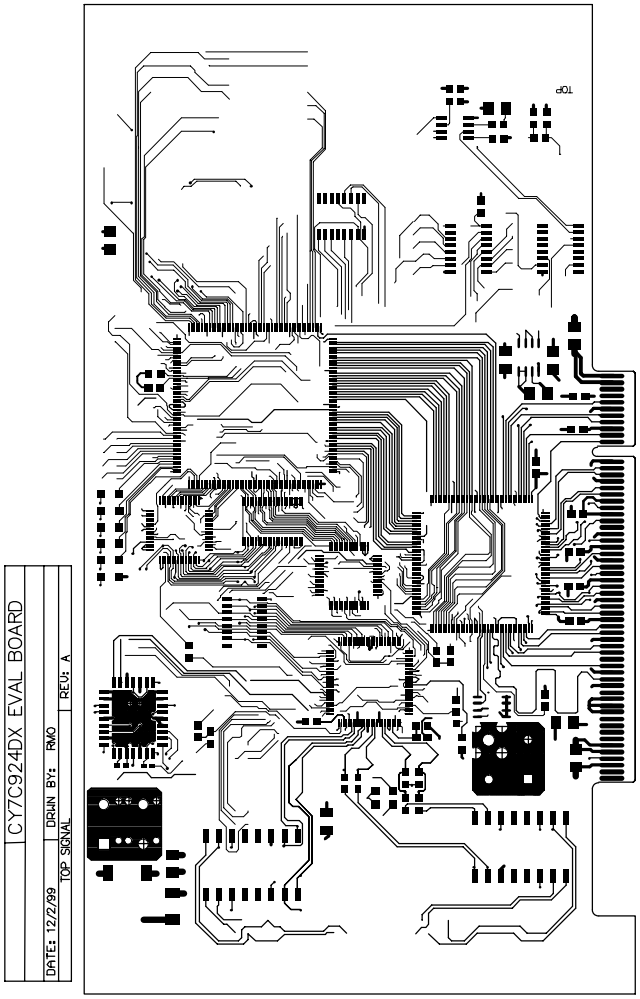


CY7C9689 Evaluation Board Schematic Page 4

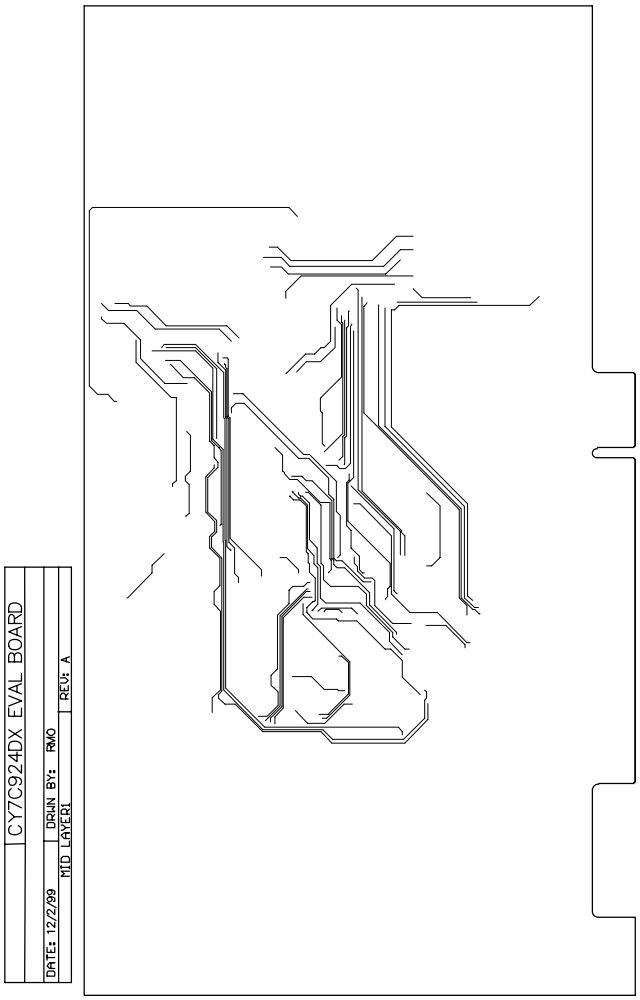


CY7C924DX Evaluation Board Layout Artwork

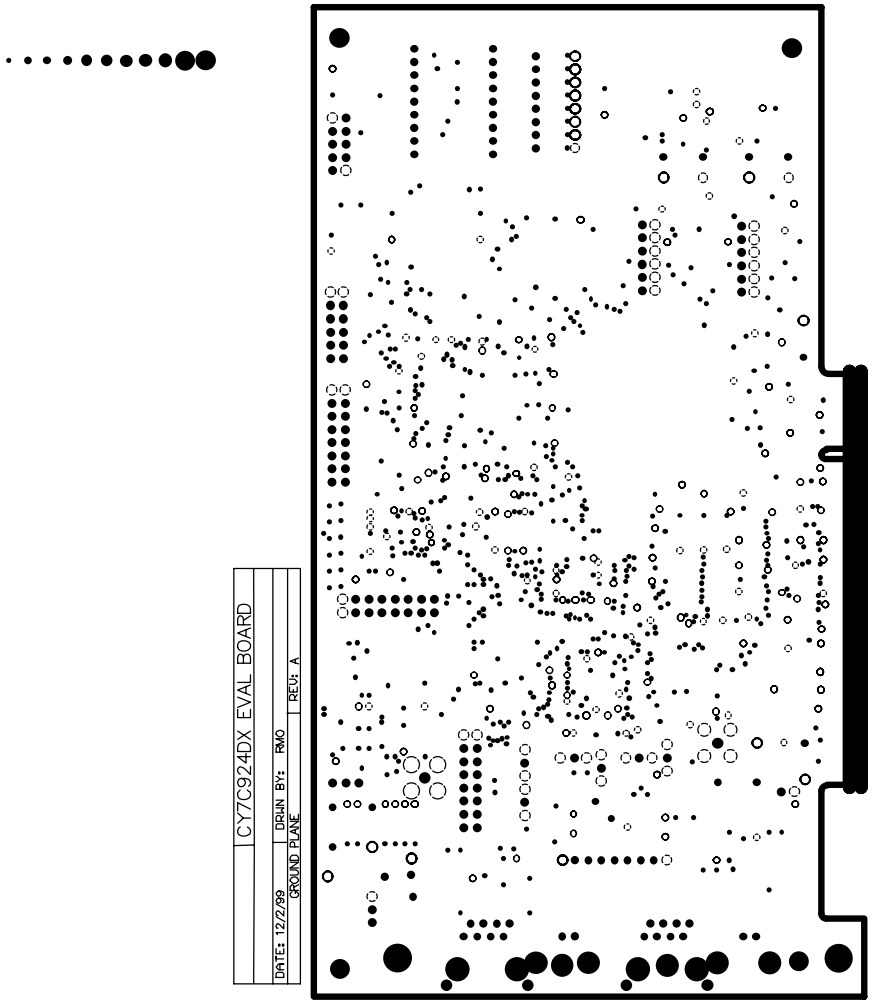
Top Signal Plane



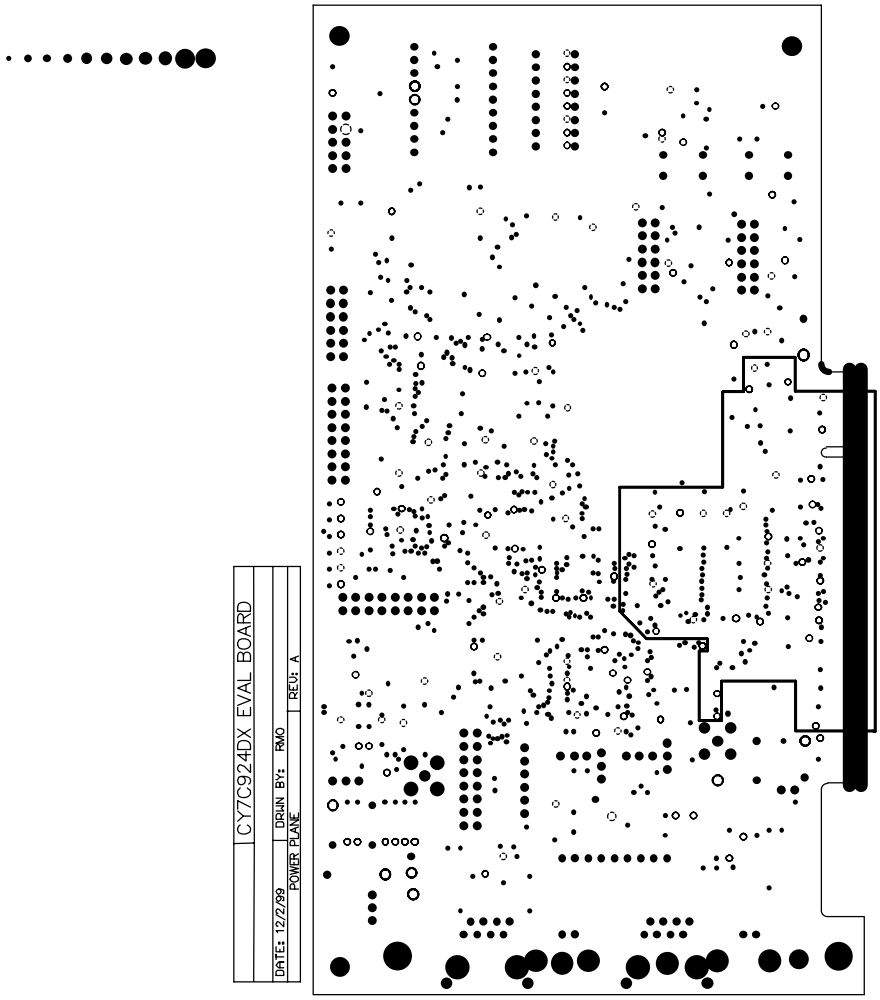
Mid Layer 1



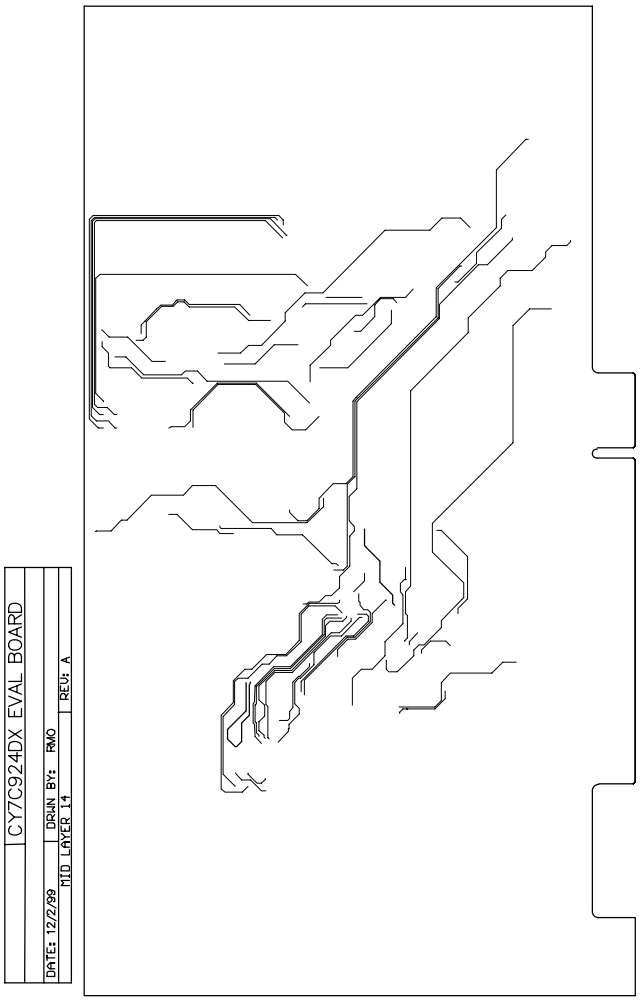
Ground Plane



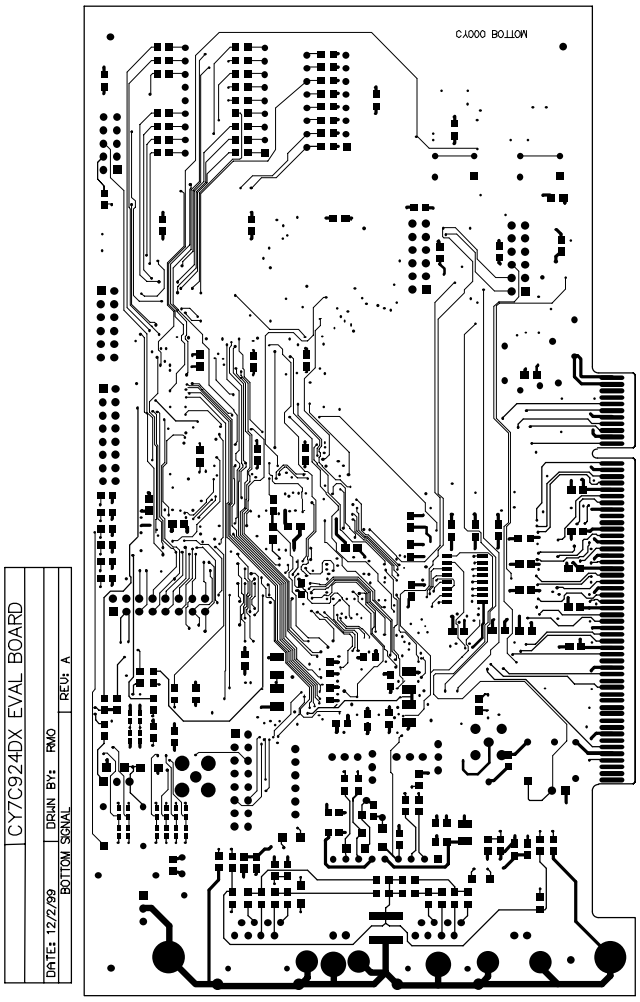
Power Plane


CY7C924DX EVAL BOARD
DRAWN BY: RMO
REV: A
DATE: 12/2/99
POWER PLANE

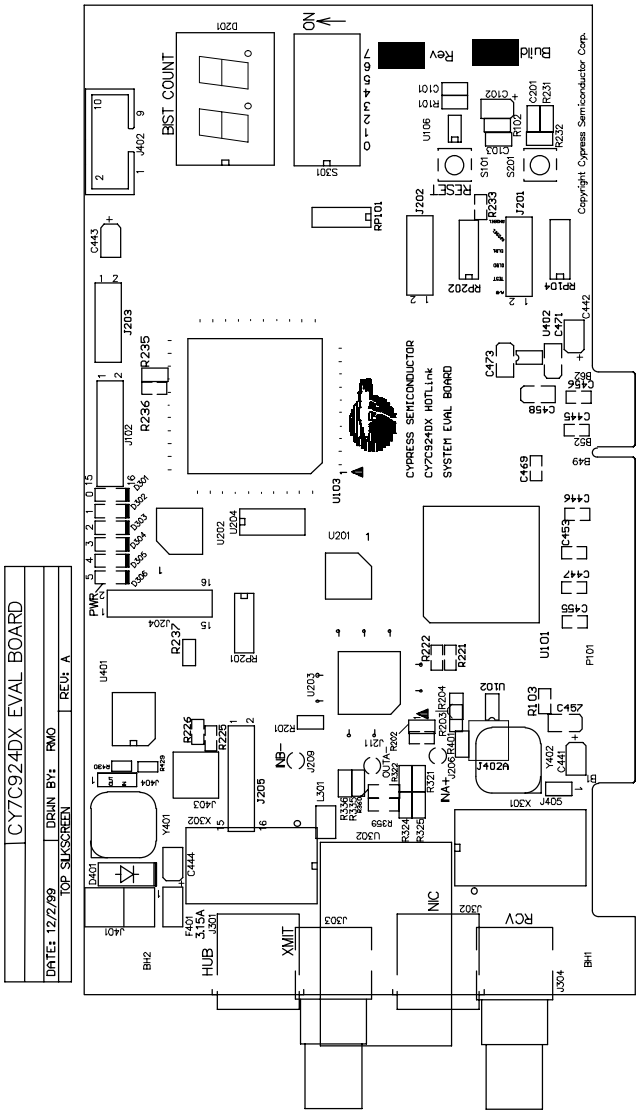
Mid Layer 14


CY7C924DX EVAL BOARD
DATE: 12/2/99
DRWN BY: RMO
REV: A
MID LAYER 14

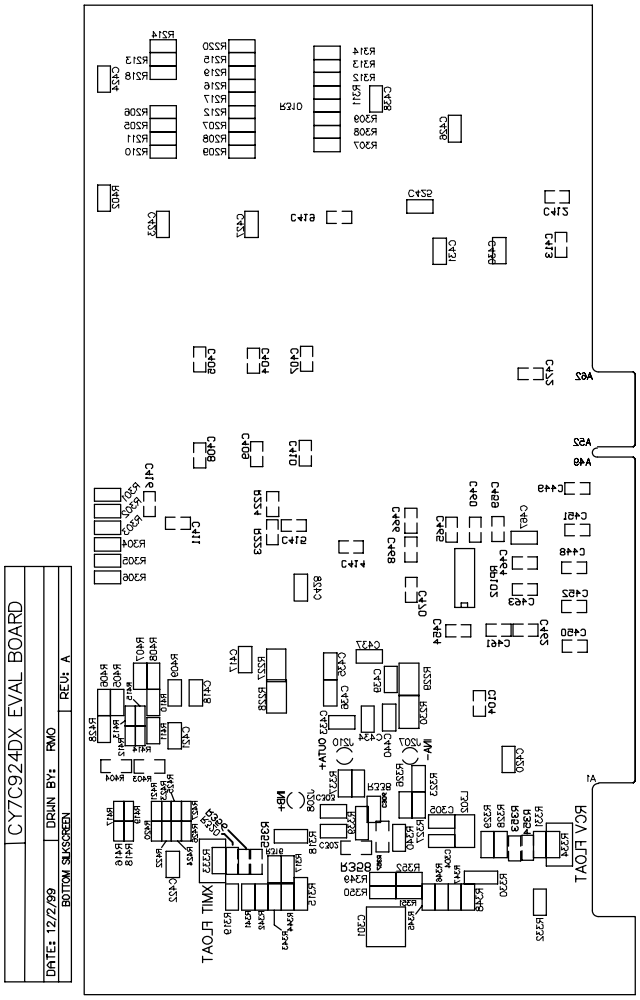
Bottom Layer Signal



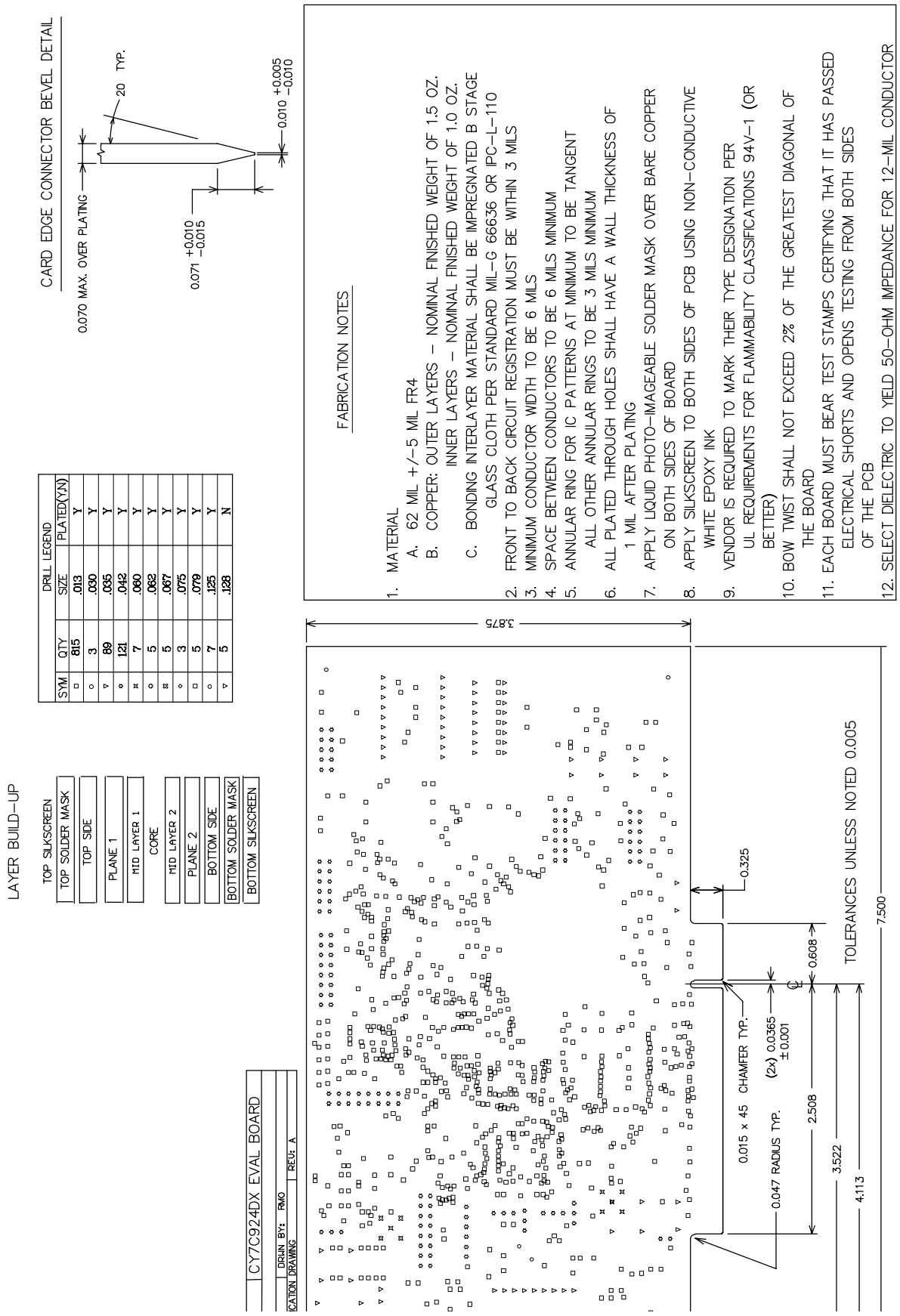
Top Layer Silkscreen



Bottom Layer Silkscreen



Drill Drawing



CPLD Source Code

top_level_pkg.vhd

```
--
--  File: top_level_pkg.vhd
--  Date: 06/05/00 12:05:12
--  Cypress Semiconductor
--  Copyright:  Copyright apply to the following intellectual property.
--The following intellectual property may not be used or duplicated
--without the consent of the author/designer.
--Disclaimer: The following code is for modeling/prove of concept only and it
--is not guaranteed.
--Purpose: This is the top level package which puts all blocks together in a package.


library IEEE;
use IEEE.std_logic_1164.all;

package bridge_blocks is

-- Declare all components that will be used.
component arbitor port (
-- common signals
int_rst: in std_logic;
sysclk: in std_logic;

-- local interrupt input
lirq_l: in std_logic;

-- tx fifo flag
tx_paf_l: in std_logic;

-- 924 TXHALT* signal
tx_halt_l: in std_logic;

-- rx fifo flag

rx_ef_l: in std_logic;

-- flag inputs
tx_buf_full: in std_logic;
tx_buf_empty: in std_logic;
rx_buf_full: in std_logic;
rx_buf_empty: in std_logic;

-- status inputs to arbitor
init_done: in std_logic;
```

```
tx_xfer_done: in std_logic;
rx_xfer_done: in std_logic;
rd_hlmb_done: in std_logic;
wr_lhmb_done: in std_logic;
wr_lint_done: in std_logic;

-- enable outputs from arbitor
arb_tx_buf_empty : out std_logic;
arb_rx_buf_full  : out std_logic;
init_en: out std_logic;
tx_xfer_en: out std_logic;
rx_xfer_en: out std_logic;
rd_hlmb_en: out std_logic;
wr_lhmb_en: out std_logic;
wr_lint_en: out std_logic;
end component;

component flag_reg
  port (
-- common signals
int_rst: in std_logic;
sysclk: in std_logic;
-- flag outputs
tx_buf_full: inout std_logic;
tx_buf_empty: inout std_logic;
rx_buf_full: inout std_logic;
rx_buf_empty: inout std_logic;

-- input from hlmb registers read
reg_tx_buf_full : in std_logic;
reg_rx_buf_empty : in std_logic;
reg_rd_en: in std_logic;

-- input from arbitor
arb_tx_buf_empty : in std_logic;
arb_rx_buf_full  : in std_logic;
end component;

component  init
    Port (
-- common system signal
int_rst: In Std_Logic;
sysclk: In Std_Logic;

-- signal from arbitor
init_en: in std_logic;
```

```
-- signal from 924
lfi_l: in std_logic;

-- signal to the arbitor
init_done: out std_logic;

-- reset signal to the 924
--xcvr_rst0_l : out std_logic;
--xcvr_rst1_l: out std_logic;

-- tx_int to 924
tx_int: out std_logic;

-- reset signal to the 924 internal FIFO
txrst_l: out std_logic;
rxrst_l: out std_logic;

-- reset signal to the external FIFOs.
tx_fifo_rst_l: out std_logic;
rx_fifo_rst_l: out std_logic;

End component;

component Mem_cnt
   Port (
int_rst: In Std_Logic;
SYSCLK: In Std_Logic;
BLK_LENGTH: In Std_Logic_Vector(3 downto 0);
TX_CNT_LOAD: In Std_Logic;
TX_CNT_FULL: OUT Std_Logic;
TX_CNT_INC: In Std_Logic;
RX_CNT_LOAD: In Std_Logic;
RX_CNT_FULL: OUT Std_Logic;

-- For testing:
-- INITIAL_CNT_OUT: OUT Std_Logic_Vector(11 downto 0);
-- ..... TX_CNT_OUT: OUT Std_Logic_Vector(11 downto 0);
-- .....RX_CNT_OUT: OUT Std_Logic_Vector(11 downto 0);
--
RX_CNT_INC: IN Std_Logic
   );

End ..... component;

component  rd_hlmb
```

```
  port (
-- Arbiter signals
rd_hlmb_en: in std_logic;
reg_access_done: out std_logic;
 .....--
reg_tx_buf_full: out std_logic;
reg_rx_buf_empty: out std_logic;
reg_rd_en: out std_logic;

-- Common signals
sysclk: in std_logic;
int_rst: in std_logic;
-- PCI-DP (Co-Mem LITE) signals
-- .............lad: inout std_logic_vector (14 downto 0);
int_lad_out: out std_logic_vector (14 downto 0);-- lad to ext. o/p buffer
int_lad_in: in std_logic_vector (1 downto 0); -- save macrocells only [1:0] are brought in.
 ......lad_en_l: out std_logic; -- for enabling external o/p buffer
-- be3 is active low, but for addressing purposes, it is treated as active high
be3: out std_logic;
strobe_l: out std_logic;
read_l: out std_logic;
blast_l: out std_logic;
rdyout_l: in std_logic;
rdyin: out std_logic);
end component;

component wr_lhmb
  port (
-- Arbiter signals
wr_lhmb_en: in std_logic;
reg_access_done ..............................: out std_logic;
--
reg_tx_buf_empty: in std_logic;
reg_rx_buf_full : in std_logic;
eg_deadlock: in std_logic;

-- Common signals
sysclk: in std_logic;
int_rst: in std_logic;
-- PCI-DP (Co-Mem LITE) signals
-- ..............lad: out std_logic_vector (14 downto 0);
 ..........int_lad: out std_logic_vector (14 downto 0);
lad_en_l: out std_logic;
-- be3 is active low, but for addressing purposes, it is treated as active high
be3: out std_logic;
strobe_l: out std_logic;
```

```
read_l: out std_logic;
blast_l: out std_logic;
rdyout_l: in std_logic;
rdyin: out std_logic);
end component;

component wr_lint
  port (
-- Arbiter signals
wr_lint_en: in std_logic;
reg_access_done: out std_logic;

-- Common signals
sysclk: in std_logic;
int_rst: in std_logic;
- PCI-DP (Co-Mem LITE) signals
-- ..............................lad: out std_logic_vector (14 downto 0);
 ..............................int_lad: out std_logic_vector (14 downto 0);
lad_en_l: out std_logic;
-- be3 is active low, but for addressing purposes, it is treated as active high
be3: out std_logic;
strobe_l: out std_logic;
read_l: out std_logic;
blast_l: out std_logic;
rdyout_l: in std_logic;
rdyin: out std_logic);
end component;

component txbuf
port(-- External FIFO signals
txwen_l: out std_logic;
txpaf_l: in std_logic;
txdata: buffer std_logic_vector (14 downto 0);
-- Signals to/from other FSM blocks
tx_cnt_inc: out std_logic;
tx_cnt_full: in std_logic;
tx_cnt_load: out std_logic;
-- Common signals
sysclk: in std_logic;
int_rst: in std_logic;
-- Arbiter signals
tx_xfer_en: in std_logic;
tx_xfer_done: out std_logic;
-- PCI-DP (Co-Mem LITE) signals that is fed to a MUX and a Tri-State
 ..............................lad_in: in std_logic_vector (14 downto 0);
xlad_out: out std_logic_vector (14 downto 0);--BCL: changed from lad_out to xlad_out
```

```
-- BCL: added to control external buffer
lad_out_en: out std_logic;
..... be3: out std_logic;-- be3 is active low, but for addressing
-- purposes, it is treated as active high
strobe_l: out std_logic;
read_l: out std_logic;
blast_l: out std_logic;
rdyout_l: in std_logic;
rdyin: out std_logic);
end component;

component rxbuf
  port (-- External FIFO signals
  rxen_l: out std_logic;
  rxef_l: in std_logic;
.............................rx_data: in std_logic_vector (14 downto 0);
-- Signals to/from other FSM blocks
rx_cnt_inc: out std_logic;
rx_cnt_full: in std_logic;
rx_cnt_load: out std_logic;
-- Common signals
sysclk: in std_logic;
int_rst: in std_logic;
-- Arbiter signals
rx_xfer_en: in std_logic;
rx_xfer_done: out std_logic;
-- PCI-DP (Co-Mem LITE) signals
--lad: out std_logic_vector (14 downto 0); -- BCL: comment out
dxlad: buffer std_logic_vector (14 downto 0);-- BCL: added to drive ext. buf
-- BCL: added to control external buffer
lad_en: out std_logic;
.. be3: buffer std_logic;-- be3 is active low, but for addressing
-- purposes, it is treated as active high
strobe_l: out std_logic;
read_l: out std_logic;
blast_l: out std_logic;
rdyout_l: in std_logic;
rdyin: out std_logic);
end component;

Component display_decode
   Port (
OVERFLOW_B: In Std_Logic;
MIS_DP_B: In std_logic;
ERR_Cntr: In Std_Logic_Vector(7 DownTo 0); -- Error counter
LS_SEGA_B: Out Std_Logic;
```

```
LS_SEGB_B: Out Std_Logic;
LS_SEGC_B: Out Std_Logic;
LS_SEGD_B: Out Std_Logic;
LS_SEGE_B: Out Std_Logic;
LS_SEGF_B: Out Std_Logic;
LS_SEGG_B: Out Std_Logic;
LS_DP_B: Out Std_Logic;
MS_SEGA_B: Out Std_Logic;
MS_SEGB_B: Out Std_Logic;
MS_SEGC_B: Out Std_Logic;
MS_SEGD_B: Out Std_Logic;
MS_SEGE_B: Out Std_Logic;
MS_SEGF_B: Out Std_Logic;
MS_SEGG_B: Out Std_Logic;
MS_DP_B: Out Std_Logic;

End component;

end bridge_blocks;
```

top_level1.vhd

```
--
--  File: top_level1.vhd
--  Date: 06/05/00 12:05:12
--  Designed by: Benjamin Leung
-- Cypress Semiconductor
--  Copyright:  Copyright apply to the following intellectual property.
--  The following intellectual property may not be used or duplicated
-- ..... without the consent of the author/designer.
--Disclaimer: The following code is for modeling/prove of concept only and it
--is not guaranteed.
--Purpose: This is the top level code which instantiates and ties all the blocks
--          together.
--06/10/2000
--1. assigned tx/rx_buf_empty/full flags to 7 segment displays
-- ....2. assigned blocks enable signals to status_led displays
--  06/12/2000 BCL separated OUTDATA bus to have direct control of HALT*, TXINT
--06/15/2000 BCL connect XCVER_RESET0/1 to LRESETL (PCI bus reset), and changed
--                other state machine's reset to depend on RSTOUTD_B and SYS_RST_B.
--  06/16/2000 BCL Added TX_INT MUX and reroute tx_int from initialization FSM to mux.
--  06/18/2000 BCL Added Flow control: TXHALT* controlled by RXINT
--  06/18/2000 BCL Added Flow Control: RXFIFO PAF controlling TXINT;
--  06/18/2000 BCL assigned TXHALT* to dead_lock input pin of wr_lhmb module
--  06/18/2000 BCL Added TXHALT* signal to arbitor block.
--  26/7/2000 BCL synchronize TXHALT to RXINT...
--  2000-8-16 SIV asynchronous TXHALT to RXINT...
```

```
library IEEE;
use IEEE.std_logic_1164.all;

-- use for simulating in Aldec 4.1
use work.bridge_blocks.all;



entity top_level is
port(
-- Main Clock
 ..............B2CLK50: In Std_Logic;  -- main clock for all blocks

--Delayed PCI bus reset
 ................LRESET_L: In Std_Logic; -- copy of PCI bus reset
 ......................RSTOUTD_B:In Std_Logic; -- software reset in

-- TX external FIFO Control Block Related
OUTDATA: Buffer Std_Logic_Vector(12 DownTo 0); -- Note: Outdata(9) is TXHALT_L
WR_DATA_BOut Std_Logic; -- WREN_L
 ................................ TX_FF: In Std_Logic; -- ext. txfifo full flag
 .....TXFIFO_RT: Out Std_Logic;-- TX retransmit, NOT USED
-- Missing TX FIFO Control signals
 ....................... TX_PAF_L:In Std_Logic;  -- Not Jumpered yet

-- RX external FIFO Control Block Related
INDATA............................ : In Std_logic_Vector(12 DownTo 0);
RX_EF_L: In Std_Logic; -- Empty flag directly from ext. rxfifo
 ............RX_FF: In Std_Logic; -- ext. rxfifo full flag .... No Use
RD_DATA:Out Std_Logic;-- REN_L

-- TX Flow Control
-- RXINT is equal to INDATA(8)
 .......RXDATA_8 : In std_logic; -- RXINT direct input from 924
 ............RXDATA_9 : In std_logic; -- RXCMD(2) for 9689 only
RXDATA_10 : In std_logic; -- RXRVS
-- TXHALT is equal to OUTDATA(9)

-- RX Flow Control
-- TXINT is equal to OUTDATA(8)
-- Missing RX flow control signal
RX_PAF ............................: In Std_Logic; -- Not Jumpered yet

-- Local Bus Related
LAD: Inout Std_logic_vector(14 downto 0);
```

```
BLAST_L: Buffer Std_logic;
.....LREADY: Buffer Std_logic; -- equivalent to rdyin
.....LADS: Buffer Std_logic; -- equivalent to strobe_l
BE_3: Out Std_logic; -- BE3
IBE_2: Out Std_logic; -- Not Used
LBE_1: Out Std_logic; -- Not Used
LBE_0: Out Std_logic; -- Not Used
.....LIRQ_L: In Std_logic; -- local interrupt from PCI-DP
ALE: OUT Std_logic; -- ALE not used
BTERM_B: OUT Std_logic; -- equivalent to IRQIN_B to PCI-DP not used

-- Missing local bus signals
--SELECT_L: Out Std_logic;   --CPLD-127 OR J202_2 JUMPER5
RDYOUT_L: In Std_logic;  --CPLD-29
READ_L: Out Std_logic;   -- No jumper yet
.....RSTOUTD: In Std_logic;   -- May not need....

-- Initialization
LFI_B:In Std_Logic;  -- LFI*
XCR_RST0 ..... :Out Std_logic;-- 924/9689 resets
XCR_RST1:Out Std_logic;
....RXRST_B: Out STd_Logic;-- 924/9689 internal fifo resets
TXRST_B:Out STd_Logic;
.....RST_RXFIFO_B: Out Std_Logic;  -- external fifo resets
RST_TXFIFO_B:Out Std_Logic;

-- 924 Direct Control
.....BYTE8_10_B: Out Std_Logic;-- static control
.....ENCBYP_B: Out Std_Logic;  -- static control
.....TXBISTEN_B: Out Std_Logic;  -- static control
..... RXBISTEN_B: Out Std_Logic;  -- static control
RXMODE0 .....: Out Std_Logic; -- static control
RXMODE1 .....: Out Std_Logic; -- static control
RXHALF_B: In Std_Logic;
RXFULL_B: In Std_Logic;
RXEMPTY_B:In Std_Logic;
TXEMPTY_B:In Std_Logic;
TXHALF_B:In Std_Logic;
TXFULL_B:In Std_Logic;
TXSTOP_B .....: Out Std_Logic; -- static control
RFEN: Out Std_Logic; -- static control
AM_L:Out Std_Logic; -- static control
LOOPTX_VLTN:InOut Std_Logic;
-- special connection from RXFIFO to 924 for RXEN control in address write phase
-- must be directly connection for normal operation
RXEN_924:Out Std_Logic;
```

```
RXEN_FIFO:In..... Std_Logic;

-- PLD Mis. Inputs , Outputs and Display outputs

-- Push button switch input.....
switch_201:INstd_logic;
-- Status LEDs
STATUS0_B: Out Std_Logic;..... -- Amber LED
STATUS1_B:Out Std_Logic;..... -- Red LED
STATUS2_B:Out Std_Logic;.....-- Orange LED
STATUS3_B:Out Std_Logic;.....-- Green LED
STATUS4_B:Out Std_Logic;.....-- Yellow LED
-- Control to Xtl Oscillator
CLK50KILL_B: Out STd_Logic;
-- Mis. outputs/inputs to headers
.......pld_out0:OUTstd_logic;   -- Programmable Probe points
pld_out1:OUTstd_logic;
pld_out2:OUTstd_logic;
pld_out3:OUTstd_logic;
pld_out4:OUTstd_logic;
pld_out5:OUTstd_logic;
pld_out6:OUTstd_logic;
pld_out7  :      OUT  std_logic;

JUMPER0:OUT std_logic;
JUMPER1:OUT std_logic;
JUMPER5:OUT std_logic;
 JUMPER2:IN std_logic;
JUMPER3:IN std_logic;
-- reset from debounced switch
SYS_RESET_B:In Std_Logic;
-- Output to Force CD HIGH but the PLD output is not rail to rail CMOS so it is NOT USED
FORCE_CD: Out Std_Logic;
-- LSB Display
 ..... LS_SEGA_B: Out Std_Logic;-- 7 Segment Display
LS_SEGB_B:Out Std_Logic; .....-- LSB
LS_SEGC_B:Out Std_Logic;
LS_SEGD_B:Out Std_Logic;
LS_SEGE_B:Out Std_Logic;
LS_SEGF_B:Out Std_Logic;
LS_SEGG_B:Out Std_Logic;
LS_DP_B:Out Std_Logic;
-- MSB Display
 ..... MS_SEGA_B: Out Std_Logic;-- 7 Segment Display
MS_SEGB_B:Out Std_Logic; .....-- MSB
MS_SEGC_B:Out Std_Logic;
```

```
MS_SEGD_B:Out Std_Logic;
MS_SEGE_B:Out Std_Logic;
MS_SEGF_B:Out Std_Logic;
MS_SEGG_B:Out Std_Logic;
MS_DP_B:Out Std_Logic;
-- Control for external 16244 to connect OUTDATA to RXDATA
DRV_RXDATA_B :Out Std_Logic);
end top_level;

architecture arch_top_level of top_level is .....

signal int_rst: std_logic; -- internal reset .....

-- internal PCI-DP buffer flags
signal tx_buf_full: std_logic; -- internal flags
signal tx_buf_empty: std_logic;
signal rx_buf_full: std_logic;
signal rx_buf_empty: std_logic;

-- internal F.S.M. done signals
signal init_done: std_logic;
signal tx_xfer_done: std_logic;
signal rx_xfer_done: std_logic;
signal rd_hlmb_done: std_logic;
signal wr_lhmb_done: std_logic;
signal wr_lint_done: std_logic;

-- internal F.S.M. enable signals
signal init_en: std_logic;
signal tx_xfer_en: std_logic;
signal rx_xfer_en: std_logic;
signal rd_hlmb_en: std_logic;
signal wr_lhmb_en: std_logic;
signal wr_lint_en: std_logic;

-- internal flag registers set/reset signals
signal arb_tx_buf_empty: std_logic;
signal arb_rx_buf_full: std_logic;

-- signals from reading the hlmb register
signal reg_tx_buf_full: std_logic;
signal reg_rx_buf_empty: std_logic;
signal reg_rd_en: std_logic;

-- internal tx memory counter control signals
signal tx_cnt_load: std_logic;
```

```
signal tx_cnt_full: std_logic;
signal tx_cnt_inc: std_logic;
signal rx_cnt_load: std_logic;
signal rx_cnt_full: std_logic;
signal rx_cnt_inc: std_logic;

-- internal Local Bus signals
signal int_lad: std_logic_vector (14 downto 0);
--signal int_strobe_l: std_logic; -- These internal signals are not needed.
--signal int_read_l: std_logic;   -- They should go straight out the the pins
--signal int_rdyin: std_logic;
--signal int_blast_l: std_logic;
--signal int_be3: std_logic;
signal int_lad_en_l: std_logic;

-- for unused deadlock signal
signal int_deadlock: std_logic;

-- rd_hlmb local bus outputs
signal rd_hlmb_lad: std_logic_vector (14 downto 0);
signal rd_hlmb_strobe_l: std_logic;
signal rd_hlmb_read_l: std_logic;
signal rd_hlmb_rdyin: std_logic;
signal rd_hlmb_blast_l: std_logic;
signal rd_hlmb_be3: std_logic;
signal rd_hlmb_lad_en_l: std_logic;

-- wr_lhmb local bus outputs
signal wr_lhmb_lad: std_logic_vector (14 downto 0);
signal wr_lhmb_strobe_l: std_logic;
signal wr_lhmb_read_l: std_logic;
signal wr_lhmb_rdyin: std_logic;
signal wr_lhmb_blast_l: std_logic;
signal wr_lhmb_be3: std_logic;
signal wr_lhmb_lad_en_l: std_logic;

-- wr_lint local bus outputs
signal wr_lint_lad: std_logic_vector (14 downto 0);
signal wr_lint_strobe_l: std_logic;
signal wr_lint_read_l: std_logic;
signal wr_lint_rdyin: std_logic;
signal wr_lint_blast_l: std_logic;
signal wr_lint_be3: std_logic;
signal wr_lint_lad_en_l: std_logic;

-- tx_xfer local bus outputs
```

```
signal tx_lad: std_logic_vector (14 downto 0);
signal tx_strobe_l: std_logic;
signal tx_read_l: std_logic;
signal tx_rdyin: std_logic;
signal tx_blast_l: std_logic;
signal tx_be3: std_logic;
signal tx_lad_en_l: std_logic;
signal tx_outdata: std_logic_vector (14 downto 0);

-- rx_xfer local bus outputs
signal rx_lad: std_logic_vector (14 downto 0);
signal rx_strobe_l: std_logic;
signal rx_read_l: std_logic;
signal rx_rdyin: std_logic;
signal rx_blast_l: std_logic;
signal rx_be3: std_logic;
signal rx_lad_en_l: std_logic;
signal rx_indata: std_logic_vector (14 downto 0);

-- internal flow control signal
signal init_tx_int: std_logic;
signal int_tx_int: std_logic;

-- For debug only
signal seg_overflow_b: std_logic;
signal seg_mis_dp_b: std_logic;
signal dual_hex_num: std_logic_vector (7 downto 0);

begin
-- Static Configuration

-- 924 Configuration
BYTE8_10_B <= '1';-- 8 bit
ENCBYP_B <= '1';  -- encoder enabled
TXBISTEN_B <= '1'; -- disable TXBIST
RXBISTEN_B <= '1'; -- disable RXBIST
RXMODE0<= '0'; -- discard policy 2 : discard all sync/K28.5 char
RXMODE1<= '1';
TXSTOP_B<= '1'; -- deassert TXSTOP
RFEN <= '1'; -- Enable framer always
AM_L <= '0'; -- select 924 always
LOOPTX_VLTN <= '0'; -- disable LOOPTX
FORCE_CD <= '1';
-- Connecting RX FIFO PAF to 924 RXEN
--RXEN_924 <= RXEN_FIFO;
RXEN_924 <= RX_FF;
```

```
-- Turn off all LEDs
STATUS0_B<= '1';-- Amber LED
..STATUS1_B<= LFI_B;-- Red LED For Link Fault indication.
STATUS2_B<= '1';-- Orange LED
STATUS3_B<= '1';-- Green LED
STATUS4_B<= '1';-- Yellow LED

-- Turn off all LEDs on DISPLAY
-- LSB Display
--LS_SEGA_B..... <= '1';-- 7 Segment Display
--LS_SEGB_B<= '1';-- LSB
--LS_SEGC_B<= '1';
--LS_SEGD_B<= '1';
--LS_SEGE_B<= '1';
--LS_SEGF_B<= '1';
--LS_SEGG_B<= '1';
--LS_DP_B<= '1';
-- MSB Display
--MS_SEGA_B..... <= '1';-- 7 Segment Display
--MS_SEGB_B<= '1';-- MSB
--MS_SEGC_B<= '1';
--MS_SEGD_B<= '1';
--S_SEGE_B<= '1';
--MS_SEGF_B<= '1';
--MS_SEGG_B<= '1';
--MS_DP_B<= '1';

-- Enable Oscillator
CLK50KILL_B<= '1';

-- Disable 16244 drivers
DRV_RXDATA_B <= '1';

-- Conversion of negative reset to positive reset
.... int_rst <= (not (RSTOUTD_B)) or (not (SYS_RESET_B));
XCR_RST0 <= LRESET_L;
XCR_RST1 <= LRESET_L;

-- Static controls to the PCI-DP
ALE <= '1'; -- deassert ALE
BTERM_B <= '1'; -- deassert local IRQIN
LBE_2 <= '0'; -- Not used just tie to LOW
LBE_1 <= '0'; -- enabling upper byte
LBE_0 <= '0'; -- enabling lower byte
-- .....SELECT_L <= '0'; -- always select the PCI-DP
```

```
-- Static control for TX FIFO
TXFIFO_RT <= '0'; -- disable retransmit



-- Instantiate arbitor
u_arbitor: arbitor port map (int_rst, B2CLK50, LIRQ_L, TX_PAF_L, OUTDATA(9)-- TXHALT*
, RX_EF_L, tx_buf_full,tx_buf_empty, rx_buf_full, rx_buf_empty, init_done,
tx_xfer_done, rx_xfer_done, rd_hlmb_done, wr_lhmb_done, wr_lint_done,
arb_tx_buf_empty, arb_rx_buf_full, init_en, tx_xfer_en, rx_xfer_en,
rd_hlmb_en, wr_lhmb_en, wr_lint_en);

-- Instantiate flag registers
u_flag_reg: flag_reg port map (int_rst, B2CLK50, tx_buf_full, tx_buf_empty, rx_buf_full,
rx_buf_empty, reg_tx_buf_full, reg_rx_buf_empty, reg_rd_en, arb_tx_buf_empty,
arb_rx_buf_full);

-- Instantiate Initialization sequencer
u_init: init port map (int_rst, B2CLK50, init_en, LFI_B, init_done, --XCR_RST0, XCR_RST1,
init_tx_int, TXRST_B, RXRST_B, RST_TXFIFO_B, RST_RXFIFO_B);

-- Instantiate Memory Counters
u_mem_cnt: mem_cnt port map (int_rst, B2CLK50, "1111", tx_cnt_load, tx_cnt_full, tx_cnt_inc,
rx_cnt_load, rx_cnt_full, rx_cnt_inc);

-- Instantiate Read HLMB Module
u_rd_hlmb: rd_hlmb port map (rd_hlmb_en, rd_hlmb_done, reg_tx_buf_full, reg_rx_buf_empty,
reg_rd_en, B2CLK50, int_rst, rd_hlmb_lad, LAD(1 downto 0), rd_hlmb_lad_en_l, rd_hlmb_be3,
rd_hlmb_strobe_l, rd_hlmb_read_l, rd_hlmb_blast_l, RDYOUT_L, rd_hlmb_rdyin);

- assigned TXHALT* to dead_lock input pin of wr_lhmb module
int_deadlock <= OUTDATA(9);
-- Instantiate Write LHMB Module
u_wr_lhmb: wr_lhmb port map (wr_lhmb_en, wr_lhmb_done, tx_buf_empty, rx_buf_full, int_deadlock,
B2CLK50, int_rst, wr_lhmb_lad, wr_lhmb_lad_en_l, wr_lhmb_be3, wr_lhmb_strobe_l,
wr_lhmb_read_l, wr_lhmb_blast_l, RDYOUT_L, wr_lhmb_rdyin);

-- Instantiate Write LINT Module
u_wr_lint: wr_lint port map (wr_lint_en, wr_lint_done, B2CLK50, int_rst, wr_lint_lad,
wr_lint_lad_en_l, wr_lint_be3, wr_lint_strobe_l, wr_lint_read_l, wr_lint_blast_l,
RDYOUT_L, wr_lint_rdyin);

-- Instantiate TX FIFO transfer Module
OUTDATA(7 downto 0) <= tx_outdata(7 downto 0); -- assigning  8 bits of read data to TXFIFO.
-- ..... OUTDATA(8) <= '0'; -- assigning LOW to TXINT
```

```
-- changed by SIV 2000-08-16 to make TXHALT* change asychronously
OUTDATA(9) <= RXDATA_8; -- Flow control: TXHALT* controlled by RXINT
---- synchronize TXHALT to RXINT... BCL 26/7/2000
--sync_TXHALT:..... process (B2CLK50, RXDATA_8)
--Begin
--if rising_edge(B2CLK50) then
--OUTDATA(9) <= RXDATA_8;
--end if;
--end process sync_TXHALT;

OUTDATA(10) <= tx_outdata(10);-- assigning bit 10 to TXSVS
OUTDATA(11) <= tx_outdata(11);-- assigning bit 11 to TXSOC
OUTDATA(12) <= tx_outdata(12);-- assigning bit 12 to TXSC/D*
u_txbuf: txbuf port map (WR_DATA_B, TX_PAF_L, tx_outdata, tx_cnt_inc, tx_cnt_full,
tx_cnt_load, B2CLK50, int_rst, tx_xfer_en, tx_xfer_done, LAD, tx_lad, tx_lad_en_l, tx_be3,
 ..... tx_strobe_l, tx_read_l, tx_blast_l, RDYOUT_L, tx_rdyin);

-- Instantiate RX FIFO transfer Module
rx_indata(12 downto 0) <= INDATA;  -- stuffing two '0's to rx_indata bus.
rx_indata(14 downto 13) <= "00";
u_rxbuf: rxbuf port map (RD_DATA, RX_EF_L, rx_indata, rx_cnt_inc, rx_cnt_full,
rx_cnt_load, B2CLK50, int_rst, rx_xfer_en, rx_xfer_done, rx_lad, rx_lad_en_l, rx_be3,
 .....rx_strobe_l, rx_read_l, rx_blast_l, RDYOUT_L, rx_rdyin);

-- Instantiate 7 segment display control for debug only.
seg_overflow_b <= not(tx_xfer_en);
seg_mis_dp_b <= not (rx_xfer_en);
dual_hex_num (7 downto 4) <= ('0', '0', tx_buf_full, rx_buf_empty);
dual_hex_num (3 downto 0) <= ('0', '0', tx_buf_empty, rx_buf_full);
u_7_segment: display_decode port map(seg_overflow_b, seg_mis_dp_b, dual_hex_num, LS_SEGA_B, LS_SEGB_B,
LS_SEGC_B, LS_SEGD_B, LS_SEGE_B, LS_SEGF_B, LS_SEGG_B, LS_DP_B, MS_SEGA_B, MS_SEGB_B,
MS_SEGC_B, MS_SEGD_B, MS_SEGE_B, MS_SEGF_B, MS_SEGG_B, MS_DP_B);

-- For debug only displaying active blocks
--STATUS0_B.....<= not(tx_xfer_en);-- Amber LED
--STATUS1_B.....<= not(rx_xfer_en);-- Red LED
--STATUS2_B..... <= not(rd_hlmb_en);-- Orange LED
--STATUS3_B.....<= not(wr_lhmb_en);-- Green LED
--STATUS4_B.....<= not(wr_lint_en);-- Yellow LED

-- For debug only display enabled blocks
pld_out0<= tx_xfer_en;
pld_out1<= rx_xfer_en;
pld_out2<= rd_hlmb_en;
pld_out3<= wr_lhmb_en;
pld_out4<= wr_lint_en;
```

```
pld_out5 <= LIRQ_L;
--pld_out6<= RX_EF_L;
--pld_out7<= BLAST_L;
pld_out6<= OUTDATA(8); -- TXINT
pld_out7<= OUTDATA(9); -- TXHALT*

-- For debug only
--pld_out0 <= LAD(0);
--pld_out1 <= LAD(1);
--pld_out2 <= LAD(2);
--pld_out3 <= LAD(3);
--pld_out4 <= LAD(4);
--pld_out5 <= LAD(5);
----pld_out6 <= LAD(6);
--pld_out6 <=BLAST_L;
----pld_out7 <= LAD(7);
--pld_out7<= tx_xfer_en;

--pld_out0 <= LAD(8);
--pld_out1 <= LAD(9);
--pld_out2 <= LAD(10);
--pld_out3 <= LAD(11);
--pld_out4 <= LAD(12);
--pld_out5 <= LAD(13);
--pld_out6 <= LAD(14);

JUMPER0 <= RDYOUT_L;
JUMPER1 <= LREADY;
JUMPER5 <= LADS;

-- TX_INT MUX
TXINT_MUX: process (int_tx_int, init_en, init_tx_int, RXEN_FIFO)
Begin
If init_en = '1' then
OUTDATA(8) <= init_tx_int;
else
OUTDATA(8) <= RXEN_FIFO; -- Flow Control: RXFIFO PAF controlling TXINT;
end if;
End Process TXINT_MUX;

-- LAD TRISTATE OUTPUT BUFFER
-- tri-state buffer for local address data bus
..... HI_Z_OUTPUTS: process (int_lad_en_l, int_lad)
Begin

If int_lad_en_l = '1' then
```

```
LAD <= (others => 'Z');
else
LAD <= int_lad;
end if;

end process HI_Z_OUTPUTS;

-- LAD OUT MUX
LAD_OUT_MUX: process (tx_xfer_en, rx_xfer_en, rd_hlmb_en,
wr_lhmb_en, wr_lint_en, tx_lad, rx_lad, rd_hlmb_lad, wr_lhmb_lad, wr_lint_lad)
begin
if (tx_xfer_en = '1') then
int_lad <= tx_lad;
elsif (rx_xfer_en = '1') then
int_lad <= rx_lad;
elsif (rd_hlmb_en = '1') then
int_lad <= rd_hlmb_lad;
elsif (wr_lhmb_en = '1') then
int_lad <= wr_lhmb_lad;
elsif (wr_lint_en = '1') then
int_lad <= wr_lint_lad;
else
int_lad <= "000000000000000";
end if;
end process LAD_OUT_MUX;

-- BE3 MUX
.... BE3_MUX: process (tx_xfer_en, rx_xfer_en, rd_hlmb_en,
wr_lhmb_en, wr_lint_en, tx_be3, rx_be3, rd_hlmb_be3, wr_lhmb_be3, wr_lint_be3)
begin
if (tx_xfer_en = '1') then
LBE_3 <= tx_be3;
elsif (rx_xfer_en = '1') then
LBE_3 <= rx_be3;
elsif (rd_hlmb_en = '1') then
LBE_3 <= rd_hlmb_be3;
elsif (wr_lhmb_en = '1') then
LBE_3 <= wr_lhmb_be3;
elsif (wr_lint_en = '1') then
LBE_3 <= wr_lint_be3;
else
LBE_3 <= '0';
end if;
end process BE3_MUX;

-- STROBE_L MUX
```

```
STROBE_L_MUX: process (tx_xfer_en, rx_xfer_en, rd_hlmb_en,
wr_lhmb_en, wr_lint_en, tx_strobe_l, rx_strobe_l, rd_hlmb_strobe_l,
wr_lhmb_strobe_l, wr_lint_strobe_l)
begin
if (tx_xfer_en = '1') then
LADS <= tx_strobe_l;
elsif (rx_xfer_en = '1') then
LADS <= rx_strobe_l;
elsif (rd_hlmb_en = '1') then
LADS <= rd_hlmb_strobe_l;
elsif (wr_lhmb_en = '1') then
LADS <= wr_lhmb_strobe_l;
elsif (wr_lint_en = '1') then
LADS <= wr_lint_strobe_l;
else
LADS <= '1';
end if;
end process STROBE_L_MUX;

-- READ_L MUX
READ_L_MUX: process (tx_xfer_en, rx_xfer_en, rd_hlmb_en,
wr_lhmb_en, wr_lint_en, tx_read_l, rx_read_l, rd_hlmb_read_l,
wr_lhmb_read_l, wr_lint_read_l)
begin
if (tx_xfer_en = '1') then
READ_L <= tx_read_l;
elsif (rx_xfer_en = '1') then
READ_L <= rx_read_l;
elsif (rd_hlmb_en = '1') then
READ_L <= rd_hlmb_read_l;
elsif (wr_lhmb_en = '1') then
READ_L <= wr_lhmb_read_l;
elsif (wr_lint_en = '1') then
READ_L <= wr_lint_read_l;
else
READ_L <= '1';
end if;
end process READ_L_MUX;

-- RDYIN MUX
 RDYIN_MUX: process (tx_xfer_en, rx_xfer_en, rd_hlmb_en,
 ..wr_lhmb_en, wr_lint_en, tx_rdyin, rx_rdyin, rd_hlmb_rdyin,
wr_lhmb_rdyin, wr_lint_rdyin)
begin
if (tx_xfer_en = '1') then
LREADY <= tx_rdyin;
```

```
elsif (rx_xfer_en = '1') then
LREADY <= rx_rdyin;
elsif (rd_hlmb_en = '1') then
LREADY <= rd_hlmb_rdyin;
elsif (wr_lhmb_en = '1') then
LREADY <= wr_lhmb_rdyin;
elsif (wr_lint_en = '1') then
LREADY <= wr_lint_rdyin;
else
LREADY <= '0';
end if;
end process RDYIN_MUX;

-- BLAST_L MUX
BLAST_L_MUX: process (tx_xfer_en, rx_xfer_en, rd_hlmb_en,
wr_lhmb_en, wr_lint_en, tx_blast_l, rx_blast_l, rd_hlmb_blast_l,
wr_lhmb_blast_l, wr_lint_blast_l)
begin
if (tx_xfer_en = '1') then
BLAST_L <= tx_blast_l;
elsif (rx_xfer_en = '1') then
BLAST_L <= rx_blast_l;
elsif (rd_hlmb_en = '1') then
BLAST_L <= rd_hlmb_blast_l;
elsif (wr_lhmb_en = '1') then
BLAST_L <= wr_lhmb_blast_l;
elsif (wr_lint_en = '1') then
BLAST_L <= wr_lint_blast_l;
else
BLAST_L <= '1';
end if;
end process BLAST_L_MUX;

-- INT_LAD_EN_L MUX
INT_LAD_EN_L_MUX: process (tx_xfer_en, rx_xfer_en, rd_hlmb_en,
wr_lhmb_en, wr_lint_en, tx_lad_en_l, rx_lad_en_l, rd_hlmb_lad_en_l,
wr_lhmb_lad_en_l, wr_lint_lad_en_l)
begin
if (tx_xfer_en = '1') then
int_lad_en_l <= tx_lad_en_l;
elsif (rx_xfer_en = '1') then
int_lad_en_l <= rx_lad_en_l;
elsif (rd_hlmb_en = '1') then
int_lad_en_l <= rd_hlmb_lad_en_l;
elsif (wr_lhmb_en = '1') then
int_lad_en_l <= wr_lhmb_lad_en_l;
```

```
elsif (wr_lint_en = '1') then
int_lad_en_l <= wr_lint_lad_en_l;
else
int_lad_en_l <= '1';
end if;
end process INT_LAD_EN_L_MUX;

end arch_top_level;
```

init_ro.vhd

```
--
--  File: init_ro.vhd
--  Designed by: Benjamin Leung
-- Cypress Semiconductor
--  Copyright:  Copyright apply to the following intellectual property.
--  The following intellectual property may not be used or duplicated
-- ..... without the consent of the author/designer.
--Disclaimer: The following code is for modelling/prove of concept only and it
--is not guaranteed.
--Purpose: This block will generate resets to the CY7C924/9689, internal TX/RXFIFOs, and
--          external TX and RX FIFOs.
--
--  History List:
--  5/31/2000BCL Created File
--  6/5/2000 BCL fixed reset seq. Ext. FIFO release first then internal FIFOs.
--  06/16/2000 BCL remove XCVR_RST0/1 outputs.
--  06/16/2000 BCL add tx_int reset sequence.
--  06/18/2000 BCL change polarity of TXINT from active HIGH to active LOW
--
Library IEEE;
Use IEEE.Std_Logic_1164.all;
--
Library Cypress;
Use Cypress.Std_Arith.all;
--
Entity init Is
   Port (
-- common system signal
int_rst: In Std_Logic;
sysclk: In Std_Logic;

-- signal from arbitor
init_en: in std_logic;

-- signal from 924
lfi_l: in std_logic;
```

```
-- signal to the arbitor
init_done: out std_logic;

-- reset signal to the 924
--xcvr_rst0_l : out std_logic;
--xcvr_rst1_l: out std_logic;

-- tx_int to 924
tx_int: out std_logic;

-- reset signal to the 924 internal FIFO
txrst_l: out std_logic;
rxrst_l: out std_logic;

-- reset signal to the external FIFOs.
tx_fifo_rst_l: out std_logic;
rx_fifo_rst_l: out std_logic
   );

End init;
--
Architecture init_arch Of init Is
type arch_init_type is (init_idle, init_lfi_check1, init_lfi_wait, init_lfi_check2,
.....init_in_fifo_wait, init_rel_int_fifo, init_ext_fifo_wait,
init_rel_ext_fifo, init_done_st);

signal present_state, next_state: arch_init_type;

constant init_const: std_logic_vector(4 downto 0) := "10000";

signal init_cnt: std_logic_vector(4 downto 0); -- counter value for keeping wait time.
signal init_cnt_load: std_logic;  -- load counter signal
signal init_cnt_inc: std_logic;  -- increase counter signal
signal init_cnt_full: std_logic;  -- counter reached 0 signal

signal init_cnt_full_d: std_logic;  -- internal count full signal for sync
signal init_cnt_load_d: std_logic;  -- internal count load signal for sync
signal init_cnt_inc_d: std_logic;  -- internal count increment signal for sync

signal init_done_d: std_logic; -- internal init_done signal for sync

--signal xcvr_rst0_l_d: std_logic; -- internal transceiver reset signals for sync
--signal xcvr_rst1_l_d: std_logic;

signal txrst_l_d: std_logic; -- internal xcvr FIFO reset signals for sync
```

```
signal rxrst_l_d: std_logic;

signal tx_fifo_rst_l_d: std_logic; -- internal ext. FIFO reset signals for sync
signal rx_fifo_rst_l_d: std_logic;

signal tx_int_d: std_logic; -- internal tx_int

Begin

arch_INIT_machine: Process (int_rst, init_en, lfi_l, init_cnt_full, present_state)
Begin

case present_state is
 ..... when init_idle =>  -- wait for enable signal
if (init_en = '1') then
next_state <= init_lfi_check1;
else
next_state <= init_idle;
end if;
when init_lfi_check1 => -- check for lfi* deassert for the first time
if (lfi_l = '1') then
next_state <= init_lfi_wait;
else
next_state <= init_lfi_check1;
end if;
when init_lfi_wait => -- wait or a while after detected lif* deassertion
if (init_cnt_full = '1') then
next_state <= init_lfi_check2;
else
next_state <= init_lfi_wait;
end if;
 ..... when init_lfi_check2 => -- check lif* deassertion again
if (lfi_l = '1') then
next_state <= init_ext_fifo_wait; -- if lfi* is constantly deasserted then
-- go and release resets
else
next_state <= init_lfi_check1; -- if lif* is asserted then go back and start
 -- check again.
end if;
when init_ext_fifo_wait =>
if (init_cnt_full = '1') then -- wait for N cycles, then release ext. fifos
next_state <= init_rel_ext_fifo;
else
next_state <= init_ext_fifo_wait;
end if;
 ..... when init_rel_ext_fifo =>  -- after releasing ext. fifos go
```

```
-- straight to ext_fifo_wait state.
next_state <= init_in_fifo_wait;.....
when init_in_fifo_wait =>
if (init_cnt_full = '1') then  -- wait for N cycles then release the xcvr internal
-- fifos.
next_state <= init_rel_int_fifo;
else
next_state <= init_in_fifo_wait;
end if;
when init_rel_int_fifo =>
if (init_en = '0') then
next_state <= init_done_st;
else
next_state <= init_rel_int_fifo;
end if;
when init_done_st =>
if (init_en = '1') then
next_state <= init_idle;
else
next_state <= present_state;
end if;
end case;

End Process;

init_cnt_proc: Process (sysclk, init_cnt_load)
Begin
if rising_edge (sysclk) then
If (init_cnt_load = '1') then
init_cnt <= init_const;
elsif (init_cnt_inc = '1') then
init_cnt <= init_cnt - 1;
else
init_cnt <= init_cnt;
end if;
end if;
End Process;



-- assigning output w.r.t. states

with next_state select
tx_int_d <= '0' when init_ext_fifo_wait, -- changed to active low, end up in HIGH state
'1' when others;

with next_state select
```

```
init_cnt_load_d <= '1' when init_idle,
  '1' when init_lfi_check1,
  '1' when init_lfi_check2,
  '1' when init_rel_int_fifo,
  '1' when init_rel_ext_fifo,
  '1' when init_done_st,
  '0' when others;

with next_state select
init_cnt_inc_d <= '0' when init_idle,
    '0' when init_lfi_check1,
    '0' when init_lfi_check2,
    '0' when init_rel_int_fifo,
    '0' when init_rel_ext_fifo,
    '0' when init_done_st,
    '1' when others;

with next_state select
 ..................................init_done_d <= '1' when init_rel_int_fifo,
    '0' when others;

--with next_state select
--xcvr_rst0_l_d <= '0' when init_idle,
--'1' when others;
--
--with next_state select
--xcvr_rst1_l_d <= '0' when init_idle,
--'1' when others;

with next_state select
tx_fifo_rst_l_d <= '0' when init_idle,
 '0' when init_lfi_check1,
 '0' when init_lfi_wait,
 '0' when init_lfi_check2,
 '0' when init_ext_fifo_wait,
 '1' when others;

with next_state select
rx_fifo_rst_l_d <= '0' when init_idle,
 '0' when init_lfi_check1,
 '0' when init_lfi_wait,
 '0' when init_lfi_check2,
 '0' when init_ext_fifo_wait,
 '1' when others;
with next_state select
txrst_l_d <= '0' when init_idle,
```

```
 '0' when init_lfi_check1,
 '0' when init_lfi_wait,
 '0' when init_lfi_check2,
 '0' when init_ext_fifo_wait,
 '0' when init_rel_ext_fifo,
 '0' when init_in_fifo_wait,
 '1' when others;

with next_state select
rxrst_l_d <= '0' when init_idle,
 '0' when init_lfi_check1,
 '0' when init_lfi_wait,
 '0' when init_lfi_check2,
 '0' when init_ext_fifo_wait,
 '0' when init_rel_ext_fifo,
 '0' when init_in_fifo_wait,
 '1' when others;

assign_init_cnt_full: Process (sysclk, init_cnt)
Begin

If rising_edge (sysclk) then
if (init_cnt = "00000") then
init_cnt_full_d <= '1';
else
init_cnt_full_d <= '0';
end if;
end if;
End Process;

init_state: Process (sysclk, int_rst)

Begin

If (int_rst = '1') then
present_state <= init_idle;
init_cnt_full <= '0';
init_cnt_load <= '1';
init_cnt_inc <= '0';
--xcvr_rst0_l <= '0';
--xcvr_rst1_l <= '0';
txrst_l<= '0';
rxrst_l<= '0';
tx_fifo_rst_l<= '0';
rx_fifo_rst_l<= '0';
init_done <= '0';
```

```
tx_int <= '0';
elsif rising_edge (sysclk) then
present_state <= next_state;
init_cnt_full <= init_cnt_full_d;
init_cnt_load <= init_cnt_load_d;
init_cnt_inc <= init_cnt_inc_d;
init_done <= init_done_d;
--xcvr_rst0_l <= xcvr_rst0_l_d;
--xcvr_rst1_l <= xcvr_rst1_l_d;
txrst_l <= txrst_l_d;
rxrst_l <= rxrst_l_d;
tx_fifo_rst_l <= tx_fifo_rst_l_d;
rx_fifo_rst_l <= rx_fifo_rst_l_d;
tx_int <= tx_int_d;
end if;

End Process;

End init_arch;
```

arbitor.vhd

```
--
--  File: arbitor.vhd
--  Designed by: Benjamin Leung
-- Cypress Semiconductor
--  Copyright:  Copyright apply to the following intellectual property.
--  The following intellectual property may not be used or duplicated
-- ..... without the consent of the author/designer.
--Disclaimer: The following code is for modeling/prove of concept only and it
--is not guaranteed.
--Purpose: This block controls the overall operation of the FIFO to PCI-DP function.
--
--          Interrupt Handling
--          ------------------
--          It will handle local interrupt from the PCI-DP (generated by host setting
--          HLMB(24)). After the interrupt is registered, the HLMB lower 16-bits will be
--          read. Then the internal buffer status flags will be updated. The LHMB contents
--          will be updated. Finally, the local interrupt status and control register
--          will be written to to clear the local interrupt. Then returning to IDLE state.
--
--          TX Transfer
--          -----------
--          The TX transfer is triggered by the tx_buf_full flag assertion (asserted by
--          host writing to the HLMB), tx_buf_empty deasserted and TXPAF* deasserted.
--          The TX_XFER engine will be engaged until it is done (i.e. Nk word is read
--          out of the TX_BUF. The the tx_buf_empty flag will be updated. Lastly, the
--          LHMB is written to with the latest buffer statuses.
```

```
--
--          RX Transfer
--          -----------
--          The RX transfer is triggered by the rx_buf_empty flag assertion (asserted by
--          host writing to the HLMB), rx_buf_full deasserted and TXEF* deasserted.
--          The RX_XFER engine will be engaged until it is done (i.e. Nk word is read
--          out of the RXFIFO. The the rx_buf_full flag will be updated. Lastly, the
--          LHMB is written to with the latest buffer statuses.
--
--          Arbitration
--          -----------
--          When both of the TX and RX transfer condition are satisfied at the same time,
--          the arbitor will enter a decision state. Depending on who (TX or RX engine) was
--          activated in the last transfer (state kept by RX_TX_L register). Then the
--          other engine will take control this turn.
--
--
--  History List:
--  5/26/2000 .....................................................BCL Created File
--  6/8/2000BCL Added arb_lirq check state to wait for lirq to deassert
--6/8/2000BCL Change <= present_state to <= "state variable"
--  6/9/2000 ..... BCL added one-hot encoding
--  6/18/2000BCL added condition to tx_xfer_go statement: it will only tx when 924 not
--halted.
--

library IEEE;
use IEEE.std_logic_1164.all;

entity arbitor is
  port (
-- common signals
int_rst: in std_logic;
sysclk: in std_logic;

-- local interrupt input
lirq_l: in std_logic;

-- tx fifo flag
tx_paf_l: in std_logic;

-- 924 TXHALT* signal
tx_halt_l : in std_logic;

-- rx fifo flag
```

```
rx_ef_l: in std_logic;

-- flag inputs
tx_buf_full: in std_logic;
tx_buf_empty in std_logic;
rx_buf_full : in std_logic;
rx_buf_empty: in std_logic;

-- status inputs to arbitor
init_done: in std_logic;
tx_xfer_done: in std_logic;
rx_xfer_done: in std_logic;
rd_hlmb_done: in std_logic;
wr_lhmb_done: in std_logic;
wr_lint_done: in std_logic;

-- enable outputs from arbitor
arb_tx_buf_empty : out std_logic;
arb_rx_buf_full  : out std_logic;
init_en: out std_logic;
tx_xfer_en: out std_logic;
rx_xfer_en: out std_logic;
rd_hlmb_en: out std_logic;
wr_lhmb_en: out std_logic;
wr_lint_en: out std_logic
);
end entity arbitor;

architecture arbitor_arch of arbitor is

type arch_arbitor_type is (arb_init, arb_idle, arb_rd_hlmb, arb_wr_lhmb, arb_wr_lint, arb_tx_xfer,
.. arb_update_txbufempty, arb_rx_xfer, arb_update_rxbuffull,
..... arb_decision, arb_wr_lhmb_1, arb_lirq_check);
-- BCL added for trial
--Attribute state_encoding of arch_arbitor_type: type is one_hot_one;
Attribute state_encoding of arch_arbitor_type: type is gray;

signal present_state, next_state: arch_arbitor_type;

-- For registered outputs
signalint_arb_tx_buf_empty :  std_logic;
signalint_arb_rx_buf_full  :  std_logic;
signalint_init_en: std_logic;
signalint_tx_xfer_en:  std_logic;
signalint_rx_xfer_en:  std_logic;
signalint_rd_hlmb_en:  std_logic;
```

```
signalint_wr_lhmb_en:  std_logic;
signalint_wr_lint_en:  std_logic;

-- combinational status signals
signalint_tx_xfer_go ..... : boolean;
signal int_rx_xfer_go..... : boolean;

-- RX and TX previous state
signalrx_tx_l: std_logic;
signalint_rx_tx_l: std_logic;

begin

-- combinational logic

int_tx_xfer_go <= ((tx_buf_empty = '0') and (tx_buf_full = '1') and (tx_paf_l = '1')
 ..... and (lirq_l = '1') and (tx_halt_l = '1'));

int_rx_xfer_go <= ((rx_buf_empty = '1') and (rx_buf_full = '0') and (rx_ef_l = '1')
and (lirq_l = '1'));


arch_arbitor_machine: process (lirq_l,tx_buf_full, tx_buf_empty, rx_buf_full, rx_buf_empty,
 tx_xfer_done, rx_xfer_done, rd_hlmb_done, wr_lhmb_done,
. wr_lint_done, present_state, int_tx_xfer_go, int_rx_xfer_go,
rx_tx_l, init_done)
begin

case present_state is
when arb_init =>
if (init_done = '1') then
next_state <= arb_idle;
else
next_state <= arb_init;
end if;
when arb_idle =>
if (lirq_l = '0') then
next_state <= arb_rd_hlmb;
 ..... elsif (int_tx_xfer_go and int_rx_xfer_go) then
next_state <= arb_decision;.....
elsif (int_tx_xfer_go) then
next_state <= arb_tx_xfer;
elsif (int_rx_xfer_go) then
next_state <= arb_rx_xfer;
else
next_state <= arb_idle;
```

```
end if;
when arb_rd_hlmb =>
if (rd_hlmb_done = '1')then
next_state <= arb_wr_lhmb;
else
next_state <= arb_rd_hlmb;
end if;
when arb_wr_lhmb =>
if (wr_lhmb_done = '1') then
next_state <= arb_wr_lint;
else
next_state <= arb_wr_lhmb;
end if;
when arb_wr_lint =>
if (wr_lint_done = '1') then
next_state <= arb_lirq_check;
else
next_state <= arb_wr_lint;
end if;
when arb_lirq_check =>
if (lirq_l = '1') then
next_state <= arb_idle;
else
next_state <= arb_lirq_check;
end if;
when arb_tx_xfer =>
if (tx_xfer_done = '1') then
next_state <= arb_update_txbufempty;
else
next_state <= arb_tx_xfer;
end if;
when arb_update_txbufempty =>
next_state <= arb_wr_lhmb_1;
when arb_wr_lhmb_1 =>
if (wr_lhmb_done = '1') then
next_state <= arb_idle;
else
next_state <= arb_wr_lhmb_1;
end if;
when arb_rx_xfer =>
if (rx_xfer_done = '1') then
next_state <= arb_update_rxbuffull;
else
next_state <= arb_rx_xfer;
end if;
when arb_update_rxbuffull =>
```

```
next_state <= arb_wr_lhmb_1;
when arb_decision =>
if (rx_tx_l = '0') then
next_state <= arb_rx_xfer;
else
next_state <= arb_tx_xfer;
end if;
end case;

end process arch_arbitor_machine;

with next_state select
.. int_arb_tx_buf_empty <= '1' when arb_update_txbufempty,
'0' when others;
with next_state select
.....int_arb_rx_buf_full <= '1' when arb_update_rxbuffull,
'0' when others;
with next_state select
int_tx_xfer_en <= '1' when arb_tx_xfer,
'0' when others;
with next_state select
int_rx_xfer_en <= '1' when arb_rx_xfer,
'0' when others;
with next_state select
int_rd_hlmb_en <= '1' when arb_rd_hlmb,
'0' when others;
with next_state select
int_wr_lhmb_en <= '1' when arb_wr_lhmb,
'1' when arb_wr_lhmb_1,
'0' when others;
with next_state select
int_wr_lint_en <= '1' when arb_wr_lint,
'0' when others;
with next_state select
.....int_rx_tx_l <= '1' when arb_update_rxbuffull,
'0' when arb_update_txbufempty,
rx_tx_l when others;

with next_state select
int_init_en <= '1' when arb_init,
'0' when others;

state_clocked:process(int_rst, sysclk)
--variable temp : std_logic;
begin
-- asyn. reset sets outputs to default values
```

```
if (int_rst = '1') then
arb_tx_buf_empty <= '0';
arb_rx_buf_full  <= '0';
tx_xfer_en<= '0';
rx_xfer_en<= '0';
rd_hlmb_en<= '0';
wr_lhmb_en<= '0';
wr_lint_en<= '0';
rx_tx_l<= '0';
present_state<= arb_init;
elsif (rising_edge(sysclk)) then
arb_tx_buf_empty <= int_arb_tx_buf_empty ;
arb_rx_buf_full  <= int_arb_rx_buf_full;
init_en<= int_init_en;
tx_xfer_en<= int_tx_xfer_en;
rx_xfer_en<= int_rx_xfer_en;
rd_hlmb_en<= int_rd_hlmb_en;
wr_lhmb_en<= int_wr_lhmb_en;
wr_lint_en<= int_wr_lint_en;
rx_tx_l<= int_rx_tx_l;
present_state   <= next_state;
end if;
end process state_clocked;

end architecture arbitor_arch;
```

flag_reg.vhd

```
--
--  File: flag_reg.vhd
--  Designed by: Benjamin Leung
-- Cypress Semiconductor
--  Copyright:  Copyright apply to the following intellectual property.
--  The following intellectual property may not be used or duplicated
-- ..... without the consent of the author/designer.
--Disclaimer: The following code is for modelling/prove of concept only and it
--is not guaranteed.
--Purpose: This block will keep the states of the TX_BUF_FULL, TX_BUF_EMPTY,
--          RX_BUF_FULL, RX_BUF_EMPTY flags. TX_BUF_FULL and RX_BUF_EMPTY flags
--          will be updated to reg_tx_buf_full, reg_rx_buf_empty input when rd_en input
--          is high, each flag will be reset to '0' when arb_tx_buf_not_full input or
--          rx_buf_not_empty input are high respectively. The TX_BUF_EMPTY flag will be
--          set high when the arb_tx_buf_empty input is high, this flag will be reset to
--          low when the reg_tx_full input is '1' when rd_en input is '1'. The
--          RX_BUF_FULL flag will be set when the arb_rx_buf_full input is high, this input
--          will be reset when the reg_rx_buf_empty input and rd_en inputs are HIGH.
```

```
--
--  History List:
--  5/25/2000 ..................................................BCL Created File
--  6/13/2000BCL removed reset condition in combinational logic section.
-- ..6/14/2000BCL commented out int_flags reset conditions.
--

library IEEE;
use IEEE.std_logic_1164.all;

entity flag_reg is
  port (
-- common signals
int_rst: in std_logic;
sysclk: in std_logic;
-- flag outputs
tx_buf_full: inout std_logic;
tx_buf_empty: inout std_logic;
rx_buf_full : inout std_logic;
rx_buf_empty: inout std_logic;

-- input from hlmb registers read
reg_tx_buf_full : in std_logic;
reg_rx_buf_empty : in std_logic;
reg_rd_en: in std_logic;

-- input from arbitor
arb_tx_buf_empty : in std_logic;
arb_rx_buf_full  : in std_logic
);
end entity flag_reg;

architecture flag_reg_arch of flag_reg is

-- For registered internal flags register
signal int_tx_buf_full: std_logic;
signal int_tx_buf_empty : std_logic;
signal int_rx_buf_full : std_logic;
signal int_rx_buf_empty : std_logic;

begin
arch_flag_reg_machine: process (reg_tx_buf_full, reg_rx_buf_empty, reg_rd_en,
.......... arb_tx_buf_empty, arb_rx_buf_full, int_rst, tx_buf_full,
.................................rx_buf_empty, rx_buf_full, tx_buf_empty)
begin
-- tx_buf_full flag set and reset
```

```
.................if  (reg_rd_en = '1') and (reg_tx_buf_full = '1') then
int_tx_buf_full <= '1';
elsif (arb_tx_buf_empty = '1') then
   int_tx_buf_full <= '0';
else
int_tx_buf_full <= tx_buf_full;
end if;

-- rx_buf_empty flag set and reset
i ..... f (reg_rd_en = '1') and (reg_rx_buf_empty = '1')then
int_rx_buf_empty <= '1';
elsif (arb_rx_buf_full = '1') then
int_rx_buf_empty <= '0';
else
int_rx_buf_empty <= rx_buf_empty;
end if;

-- rx_buf_full flag set and reset
if (arb_rx_buf_full = '1') then
int_rx_buf_full <= '1';
.... elsif ((reg_rx_buf_empty = '1') and (reg_rd_en = '1')) then
int_rx_buf_full <= '0';
else
int_rx_buf_full <= rx_buf_full;
end if;

-- tx_buf_empty flag set and reset
if (arb_tx_buf_empty = '1') then
int_tx_buf_empty <= '1';
.........elsif ((reg_tx_buf_full = '1') and (reg_rd_en = '1')) then
int_tx_buf_empty <= '0';
else
int_tx_buf_empty <= tx_buf_empty;
end if;

end process arch_flag_reg_machine;


state_clocked:process(int_rst, sysclk)
--variable temp : std_logic;
begin
-- asyn. reset sets outputs to default values
if (int_rst = '1') then
tx_buf_full <= '0';
tx_buf_empty <= '1';
rx_buf_full <= '0';
```

```
rx_buf_empty <= '1';
--int_tx_buf_full <= '0';
--int_tx_buf_empty <= '1';
--int_rx_buf_full <= '0';
--int_rx_buf_empty <= '1';
elsif (rising_edge(sysclk)) then
tx_buf_full <= int_tx_buf_full;
tx_buf_empty <= int_tx_buf_empty;
rx_buf_full <= int_rx_buf_full;
rx_buf_empty <= int_rx_buf_empty; .....
end if;
end process state_clocked;

end architecture flag_reg_arch;
```

mem_cnt_sr.vhd

```
--
--  File: bcl_rx.vhd
--  Designed by: Benjamin Leung
-- Cypress Semiconductor
--  Copyright:  Copyright apply to the following intellectual property.
--  The following intellectual property may not be used or duplicated
-- ..... without the consent of the author/designer.
--Disclaimer: The following code is for modelling/prove of concept only and it
-- is not guaranteed.
--Purpose: This block will keep track of the transmit and receive word count. When the
--          TX/RX_CNT_LOAD signal is asserted the TX/RX counter is loaded with the pre-
--          determined value of 9 bits. When TX_CNT_INC is asserted,
--          the TX counter will be decremented by 1. When the TX count
--          reaches TX_STOP_CNT, then the TX_CNT_FULL will be asserted to '1' for one cycle.
-- RX counter work differently from Tx counter. The HIGH to LOW transistion on
--          RX_CNT_INC will cause the counter to decrement by 1. When the
--
--  History List:
--  5/22/2000 .....BCL Created File
--  06/02/2000 BCL changed tx counter algorithm such that full will assert the cycle
--            after the n-1 increment signal.
--  06/02/2000 BCL changed rx counter bahivour after discussion with RMO. The counter
--            will now decrement on high-to-low transistions. After the n-2 high-to-low
--            transistion, the full signal will be assert the cycle after.
--06/09/2000 BCL changed TX/RX_INIT_CNT to 011111111.
-- 06/09/2000 BCL changed TX/RX_INIT_CNT to 000000111.
--06/16/2000 BCL changed TX/RX_INIT_CNT to 111111111.
--  06/19/2000 BCL changed TX/RX_INIT_CNT to 101011110.
--
Library IEEE;
Use IEEE.Std_Logic_1164.all;
```

```
--
Library Cypress;
Use Cypress.Std_Arith.all;
--
Entity Mem_cnt Is
   Port (
int_rst: In Std_Logic;
SYSCLK: In Std_Logic;
BLK_LENGTH: In Std_Logic_Vector(3 downto 0);
TX_CNT_LOAD: In Std_Logic;
TX_CNT_FULL: OUT Std_Logic;
TX_CNT_INC: In Std_Logic;
RX_CNT_LOAD: In Std_Logic;
RX_CNT_FULL: OUT Std_Logic;

-- For testing:
-- INITIAL_CNT_OUT: OUT Std_Logic_Vector(11 downto 0);
-- ..... TX_CNT_OUT: OUT Std_Logic_Vector(11 downto 0);
-- .....RX_CNT_OUT: OUT Std_Logic_Vector(11 downto 0);
--
RX_CNT_INC: IN Std_Logic
   );

End Mem_cnt;
--
Architecture mem_cnt_arch Of Mem_cnt Is

Signal TXCNT: std_logic_vector(8 downto 0);
Signal RXCNT: std_logic_vector(8 downto 0);
signal d_rx_cnt_inc: std_logic; -- delayed rx count increment
signal int_rx_cnt_inc: std_logic; -- internal rx count increment
--Signal EXT_CNT: std_logic_vector (8 downto 0);
--Signal INITIAL_CNT: std_logic_vector(8 downto 0);

--Signal TX_CNT_FULL_D: std_logic;
--Signal RX_CNT_FULL_D: std_logic;

Constant TX_INIT_CNT: Std_logic_Vector (8 downto 0):= "111111111"; -- this number is N byte -1
Constant RX_INIT_CNT: Std_logic_Vector (8 downto 0):= "111111111";
Constant TX_STOP_CNT: Std_logic_Vector (8 downto 0):= "000000001"; -- do not change end number.
Constant RX_STOP_CNT: Std_logic_Vector (8 downto 0):= "000000010";
Constant RX_STOP_CNT_SUB: Std_logic_Vector (8 downto 0):= "000000001";

Begin

--Form_ext_cnt: Process (BLK_LENGTH)
```

```
--Begin
--EXT_CNT(11) <= BLK_LENGTH(3);
--EXT_CNT(10) <= BLK_LENGTH(2);
--EXT_CNT(9) <= BLK_LENGTH(1);
--EXT_CNT(8) <= BLK_LENGTH(0);
--
--for i in 7 downto 0 loop
--EXT_CNT(i) <= '1';
--end loop;
--End Process;
--
--Form_initial_cnt: Process (EXT_CNT, TXCNT, RXCNT)
--Begin
--INITIAL_CNT_OUT <= EXT_CNT - 5;
--INITIAL_CNT <= EXT_CNT - 5;
--TX_CNT_OUT <= TXCNT;
--RX_CNT_OUT <= RXCNT;
--End Process;

-- TX counter process
TX_CNT: Process (SYSCLK, int_rst)
Begin
if rising_edge(SYSCLK) then
if (int_rst = '1') then
--TXCNT <= INITIAL_CNT;
TXCNT <= TX_INIT_CNT;
elsif TX_CNT_LOAD = '1' then
--TXCNT<= INITIAL_CNT;
TXCNT<= TX_INIT_CNT;
elsif TX_CNT_INC = '1' then
TXCNT <= TXCNT - 1;
end if;
end if;
End Process;

-- Register to delay rx_inc by one cycle
delay_rx_inc: process (int_rst, sysclk)
Begin
if (int_rst = '1') then
d_rx_cnt_inc <= '0';
elsif rising_edge (sysclk) then
d_rx_cnt_inc <= rx_cnt_inc;
end if;
End Process;

int_rx_cnt_inc <= d_rx_cnt_inc and not(rx_cnt_inc); -- combining the delayed and incoming signal
```

```
-- to form an internal inc signal

RX_CNT: Process (int_rst, SYSCLK)
Begin
if rising_edge(SYSCLK) then
if (int_rst = '1') then
--RXCNT <= INITIAL_CNT;
RXCNT <= RX_INIT_CNT;
elsif RX_CNT_LOAD = '1' then
--RXCNT<= INITIAL_CNT;
RXCNT <= RX_INIT_CNT;
elsif int_rx_cnt_inc = '1' then
RXCNT <= RXCNT - 1;
end if;
end if;
End Process;

FULL_OUT: Process (int_rst, SYSCLK)
Begin
if rising_edge (SYSCLK) then
if (int_rst = '1') then
TX_CNT_FULL <= '0';
RX_CNT_FULL <= '0';
elsif (TXCNT = TX_STOP_CNT) and ( TX_CNT_INC = '1') then
TX_CNT_FULL <= '1';
else
TX_CNT_FULL <= '0';
end if;

if (RXCNT = RX_STOP_CNT) and (int_rx_cnt_inc = '1') then
RX_CNT_FULL <= '1';
elsif (RXCNT = RX_STOP_CNT_SUB) then
RX_CNT_FULL <= '1';
else
RX_CNT_FULL <= '0';
end if;
end if;
End Process;

--Reg_out: Process (int_rst, sysclk)
--Begin
--If (int_rst = '1') then.....
--TX_CNT_FULL <= '0';
--RX_CNT_FULL <= '0';
--elsif rising_edge (sysclk) then
--tx_cnt_full <= tx_cnt_full_d;
```

```
--rx_cnt_full <= rx_cnt_full_d;.....
--end if;
--End Process;


End mem_cnt_arch;
```

rd_hlmb_iop.vhd

```
--
--  File: rd_hlmb.vhd
--  Designed by: Benjamin Leung
-- Cypress Semiconductor
--  Copyright:  Copyright apply to the following intellectual property.
--  The following intellectual property may not be used or duplicated
-- ..... without the consent of the author/designer.
--Disclaimer: The following code is for modelling/prove of concept only and it
-- is not guaranteed.
--Purpose: This block will read the content of the Host to Local Mailbox of the
--          PCI-DP. The data content of the lower 16-bit of register 0x04E8 will be
--          read and presenting the last two bits to the outputs REG_TX_BUF_FULL (HLMB[1])
--          and REG_RX_BUF_EMPTY (HLMB[0]). All outputs are registered.
--
--  History List:
--  5/23/2000 .....BCL Created File
--  6/2/2000BCL made modification to make reg_tx_full and reg_rx_empty to latch the input
--              data and keep it.
--
--

library IEEE;
use IEEE.std_logic_1164.all;

entity rd_hlmb is
  port (
-- Arbiter signals
rd_hlmb_en..... : in std_logic;
reg_access_done ..... : out std_logic;
--
reg_tx_buf_full ..... : out std_logic;
reg_rx_buf_empty..... : out std_logic;
reg_rd_en ..... : out std_logic;

-- Common signals
sysclk: in std_logic;
int_rst: in std_logic;
-- PCI-DP (Co-Mem LITE) signals
-- .....lad: inout std_logic_vector (14 downto 0);
```

```
int_lad_out: out std_logic_vector (14 downto 0);-- lad to ext. o/p buffer
int_lad_in: in std_logic_vector (1 downto 0); -- save macrocells only [1:0] are brought in.
.....lad_en_l: out std_logic; -- for enabling external o/p buffer
--be3 is active low, but for addressing purposes, it is treated as active high
be3: out std_logic;
strobe_l: out std_logic;
read_l: out std_logic;
blast_l: out std_logic;
rdyout_l: in std_logic;
rdyin: out std_logic);
end entity rd_hlmb;

architecture rd_hlmb_arch of rd_hlmb is

type arch_reg_read_type is (idle, reg_ad_phase, reg_data_wait, reg_data_read,
                            reg_rd_access_done);
signal present_state, next_state: arch_reg_read_type;

-- For registered internal lad bus output enable
--signal lad_en_l: std_logic;
signal int_lad_en_l: std_logic;

-- For registered outputs
signal int_tx_buf_full, int_rx_buf_empty: std_logic;
signal int_reg_access_done: std_logic;
--signal int_lad: std_logic_vector (14 downto 0);
signal int_int_lad: std_logic_vector (14 downto 0);
signal int_be3: std_logic;
signal int_strobe_l: std_logic;
signal int_read_l: std_logic;
signal int_rdyin: std_logic;
signal int_blast_l: std_logic;
signal int_reg_rd_en: std_logic;

begin
arch_reg_read_machine: process (present_state, rd_hlmb_en, int_lad_in, rdyout_l)
begin

case present_state is
when idle =>
if (rd_hlmb_en = '1') then
next_state <= reg_ad_phase;
else
next_state <= idle;
end if;
when reg_ad_phase =>
```

```
..... next_state <= reg_data_wait; -- always transition
when reg_data_wait =>
if (rdyout_l = '0') then
next_state <= reg_data_read;
else
next_state <= present_state;
end if;
when reg_data_read =>
next_state <= reg_rd_access_done;
when reg_rd_access_done =>
if (rd_hlmb_en = '0') then
next_state <= idle;
else
next_state <= present_state;
end if;
when others =>
next_state <= idle;
end case;
end process arch_reg_read_machine;

with next_state select
int_reg_access_done <=  .....'0' when idle,
'1' when reg_data_read,
'0' when others;
with next_state select
..... int_tx_buf_full <= int_lad_in(1) when reg_data_read,
'0' when others;

with next_state select
.....int_rx_buf_empty <= int_lad_in(0) when reg_data_read,
'0' when others;
with next_state select
int_reg_rd_en <= '1' when reg_data_read,
'0' when others;

with next_state select
int_strobe_l <= ..... '0' when reg_ad_phase,
‘1' when others;

with next_state select
int_read_l <= '0' when reg_ad_phase,
'1' when others;

with next_state select
int_blast_l <= '0' when reg_data_read,
'1' when others;
```

```
with next_state select
int_rdyin <= '1' when reg_data_read,
'0' when others;

with next_state select
int_be3 <= '0' when reg_ad_phase,
'0' when others;

with next_state select
int_lad_en_l <= '1' when idle,
'0' when reg_ad_phase,
'1' when others;

with next_state select
..... int_int_lad <= "000010011101000" when reg_ad_phase,
"000000000000000" when others;

state_clocked:process(int_rst, sysclk)
--variable temp : std_logic;
begin
.....-- asyn. reset sets outputs to default values
if (int_rst = '1') then
-- Common local bus interface
strobe_l <= '1';
read_l <= '1';
blast_l <= '1';
rdyin <= '0';
be3 <= '0';
-- common state variable
present_state <= idle;
-- arch_reg_rd specific
reg_tx_buf_full <= '0';
reg_rx_buf_empty <= '0';
reg_access_done <= '0';
elsif (rising_edge(sysclk)) then
present_state <= next_state;
reg_access_done <= int_reg_access_done;

--if (next_state = reg_data_read) then
--reg_tx_buf_full <= lad(1);
--reg_rx_buf_empty <= lad(0); .....
--reg_rd_en <= '1';
-- ..... elsif (not(next_state = reg_data_read)) then
--reg_tx_buf_full <= '0';
--reg_rx_buf_empty <= '0';
```

```
--reg_rd_en <= '0';
--end if;
reg_tx_buf_full <= int_tx_buf_full;
reg_rx_buf_empty <= int_rx_buf_empty; .....
reg_rd_en <= int_reg_rd_en;
strobe_l <= int_strobe_l;
read_l <= int_read_l;
blast_l <= int_blast_l;
rdyin <= int_rdyin;
be3 <= int_be3;
lad_en_l <= int_lad_en_l;
int_lad_out <= int_int_lad;

end if;
end process state_clocked;

--HI_Z_OUTPUTS: process (lad_en_l, int_lad)
--Begin
--
--If lad_en_l = '1' then
--lad <= (others => 'Z');
--else
--lad <= int_lad;
--end if;
--
--end process HI_Z_OUTPUTS;

end architecture rd_hlmb_arch;
```

rx_buffer_op_f.vhd

```
--======================= Template 1.0 ========================================
-- Design Units: Control unit for the receive block. Data is transferred
-- from the receive FIFO to the PCI-DP.
--
-- File Name: rxbuf.vhd
-- Purpose

-- Limitations: [none]
-- Errors: [none]
-- Libraries: ieee
-- Author: Robert O'Leary
--Cypress Semiconductors
-- ..... 3901 N. First Street, San Jose, CA, 95134
--408/432-7084
-- Environment ..... : Warp 5.2 /w Service Pack 1
--Windows 98
```

```
--------------------------------------------------------------------------------
--                          Cypress Semiconductor
-- Copyright:  Copyright apply to the following intellectual property.
-- The following intellectual property may not be used or duplicated
-- without the consent of the author/designer.
-- Disclaimer: The following code is for modelling/proof of concept only and it
-- is not guaranteed.
--------------------------------------------------------------------------------
-- Change History:
-- ____________________________________________________________________________
--|Version|Date|Author ini.|Changes
--|1.0|2000-05-23|siv|New version
--|
--|
--------------------------------------------------------------------------------
-- BCL 8/3/2000 added initial state for dxlad


library IEEE;
use IEEE.std_logic_1164.all;

entity rxbuf is
  port (-- External FIFO signals
  n_l: out std_logic;
  rxef_l: in std_logic;
 ..........................rx_data: in std_logic_vector (14 downto 0);
-- Signals to/from other FSM blocks
rx_cnt_inc: out std_logic;
rx_cnt_full: in std_logic;
rx_cnt_load ..........................................................: out std_logic;
-- Common signals
sysclk: in std_logic;
int_rst: in std_logic;
-- Arbiter signals
rx_xfer_en: in std_logic;
rx_xfer_done: out std_logic;
-- PCI-DP (Co-Mem LITE) signals
-- ..........................lad: inout std_logic_vector (14 downto 0);
-- BCL added for connecting to external buffer
.........................dxlad: buffer std_logic_vector (14 downto 0);
-- BCL added to control exteranl buffer
lad_en: out std_logic;
.. be3: buffer std_logic;-- be3 is active low, but for addressing
-- purposes, it is treated as active high
strobe_l: out std_logic;
read_l: out std_logic;
```

```
blast_l: out std_logic;
rdyout_l: in std_logic;
rdyin: out std_logic);
end entity rxbuf;

architecture rxbuf_arch of rxbuf is

-- SYMBOLIC ENCODED state machine: rxbuf_arch
-- State code: -----  21 states

type arch_rxbuf_type is (idle, address_phase, wait_for_disable, read_no_ack, bb1, bb2, bb3,
bb4,bb5, write1,write1A, write2, write3, write4, write5, write6, write7, write8, write9,write10, ackread1,
ackread2,ackread3,Read_no_Ack_full,ackread_full);
-- BCL Added to try improve speed / less passes
attribute state_encoding of arch_rxbuf_type: type is gray;

signal present_state, next_state: arch_rxbuf_type;

-- REGISTERED OUTPUT Signals
-- For registered internal lad bus output enable
-- BCL: commented out, brought to top level .....

--signal lad_en: std_logic;
signal int_lad_en: std_logic;

-- D flip flops used to generate output based on next_state
signal drxren_l: std_logic;
signal drxdata ..... : std_logic_vector (14 downto 0);

signal drx_cnt_inc: std_logic;
signal drx_cnt_load: std_logic;

signal drx_xfer_done: std_logic;

signal dlad..... : std_logic_vector (14 downto 0);
-- BCL commented out, brought to top level
--signal dxlad  : std_logic_vector (14 downto 0); -- siri check?
signal dbe3: std_logic;

signal dstrobe_l: std_logic;
signal dread_l: std_logic;
signal dblast_l: std_logic;
signal drdyin: std_logic;

begin
arch_rxbuf_machine: process (present_state, rx_cnt_full, rx_xfer_en, rdyout_l, rxef_l)
```

```
begin

-- code to goto idle state if rx_xfer_en = '0'.
if (not(present_state = idle) and rx_xfer_en = '0') then
next_state <= idle;
else
case present_state is
-- ==================== Initial states ============================================
when idle =>
if (rx_xfer_en = '1') then
next_state <= address_phase;
else
next_state <= idle;
end if;
when address_phase =>
if (rxef_l = '1') then
next_state <= read_no_ack;
else
   next_state <= address_phase;
end if;

-- ==================== First part of states ======================================
when read_no_ack =>if (rxef_l = '1') then
next_state <= bb1;
   elsif(rxef_l = '0' and rx_cnt_full = '1')then
next_state <= read_no_ack_full;
else
   next_state <= read_no_ack;
end if;.....
   when read_no_ack_full =>if (rxef_l = '1') then
next_state <=bb5;
else
   next_state <= read_no_ack_full;
end if;

   when ackread_full=>
if (rxef_l = '0') then
next_state <= ackread_full;
else
next_state <= bb5;
end if

   when write1 =>
.....if (rx_cnt_full = '0'and rdyout_l = '1') then
next_state <= write1;
..... elsif (rx_cnt_full = '0'and rdyout_l = '0') then
```

```
next_state <= bb2;
..... elsif (rx_cnt_full = '1'and rdyout_l = '0') then
next_state <= bb3;
..... elsif (rx_cnt_full = '1'and rdyout_l = '1') then
next_state <= write6;
end if;
when write1A =>
.....if (rx_cnt_full = '0'and rdyout_l = '1') then
next_state <= write1;
..... elsif (rx_cnt_full = '0'and rdyout_l = '0') then
next_state <= bb2;
..... elsif (rx_cnt_full = '1'and rdyout_l = '0') then
next_state <= bb3;
..... elsif (rx_cnt_full = '1'and rdyout_l = '1') then
next_state <= write6;
end if;
when write2=>
.....if (rdyout_l = '1' and rxef_l = '0' and rx_cnt_full= '0') then
next_state <= write2;
..... elsif (rx_cnt_full = '1'and rdyout_l = '0') then
next_state <= ackread2;
..... elsif (rx_cnt_full = '1'and rdyout_l = '1') then
next_state <= write7;
.....elsif (rxef_l = '0'and rdyout_l = '0') then
next_state <= ackread1;
.. elsif (rxef_l = '1'and rdyout_l = '1' and rx_cnt_full = '0') then
next_state <= write10;
... elsif (rxef_l = '1'and rdyout_l = '0' and rx_cnt_full ='0') then
next_state <= read_no_ack;
end if;
when write3=>
if (rdyout_l = '1') then
next_state <= write3;
elsif (rdyout_l = '0') then
next_state <= bb2;
end if;
when write4=>
.....if (rx_cnt_full = '0'and rdyout_l = '1') then
next_state <= write4;
..... elsif (rx_cnt_full = '0'and rdyout_l = '0') then
next_state <= read_no_ack;
..... elsif (rx_cnt_full = '1'and rdyout_l = '0') then
next_state <= ackread3;
..... elsif (rx_cnt_full = '1'and rdyout_l = '1') then
next_state <= write8;
end if;
```

```
when write5=>
if (rdyout_l = '1') then
next_state <= write5;
elsif (rdyout_l = '0') then
next_state <= wait_for_disable; .....
end if;
when write6=>
if (rdyout_l = '1') then
next_state <= write6;
else
next_state <= bb3;
end if;
when write7=>
if (rdyout_l = '1') then
next_state <= write7;
else
next_state <= ackread2;
end if;
when write8=>
if (rdyout_l = '1') then
next_state <= write8;
else
next_state <= ackread3;
end if;
when write9=>
if (rdyout_l = '1') then
next_state <= write9;
else
next_state <= wait_for_disable; .....
end if;
when write10=>
if (rdyout_l = '1') then
next_state <= write3;
elsif (rdyout_l = '0') then
next_state <= bb2;
end if;
when ackread1=>
 ..... if (rxef_l = '0' and rx_cnt_full = '0') then
next_state <= ackread1;
 ..... elsif(rxef_l = '0' and rx_cnt_full = '1')then
next_state <= ackread_full;
  else
next_state <= bb2;
end if;

when ackread2=>
```

```
if (rxef_l = '0') then
next_state <= ackread2;
 ..... elsif(rxef_l = '0' and rx_cnt_full = '1')then
next_state <= ackread_full;
else
next_state <= bb3;
end if;
when ackread3=>
if (rxef_l = '0') then
next_state <= ackread3;
else
next_state <= bb4;
  end if;
when bb1 =>
if (rxef_l = '1') then
next_state <= write1A;
else
   next_state <= write2;
end if;
when bb2 =>
next_state <= write4;
   when bb3 =>
next_state <= write5;
when bb4 =>
next_state <= write9;
  when bb5 =>
next_state <= write7;.....


-- ==================== Final states ================================================
when wait_for_disable =>
if (rx_xfer_en = '0') then
next_state <= idle;
else
next_state <= wait_for_disable;
end if;
end case;
end if;
end process arch_rxbuf_machine;

state_clocked:process(int_rst, sysclk)
begin
-- asyn. reset sets outputs to default values
if (int_rst = '1') then
rxen_l <= '1';
rx_cnt_inc <= '0';
```

```
rx_cnt_load <= '0';
rx_xfer_done <= '0';
strobe_l <= '1';
read_l <= '1';
blast_l <= '1';
rdyin <= '0';
be3 <= '0';
--lad <= "000000000000000";
lad_en <= '1';..... -- For the tri-state
. dxlad <= (others => '0');  -- SIV 2000-08-04 for proper reset
-- BCL 8/3/2000 added initial state for dxlad
--dxlad <= "000000000000000";
present_state <= idle;
elsif (rising_edge(sysclk)) then
rxen_l <= drxren_l;
rx_cnt_inc <= drx_cnt_inc;
rx_cnt_load <= drx_cnt_load;
rx_xfer_done <= drx_xfer_done;
strobe_l <= dstrobe_l;
read_l <= dread_l;
blast_l <= dblast_l;
rdyin <= drdyin;
dxlad <= dlad;
be3 <= dbe3;
--lad <= "000000000000000";
lad_en <= int_lad_en; ..... -- For the tri-state

present_state <= next_state;
..... end if;
end process state_clocked;

-- === Ben's Tri-State Code .....

-- Commented out to use external buffer
--HI_Z_OUTPUTS: process (lad_en, dxlad)
--Begin
--
--If lad_en = '1' then
--lad <= (others => 'Z');
--else
--lad <= dxlad;
--end if;
--
--end process HI_Z_OUTPUTS;
  --
```

```
-- ====================== Output Assignment based on the next state ===================

with next_state select -- siri note that wr_last1/2 states not here.
drxren_l <= '0' when read_no_ack |ackread_full | bb1  | bb5  | write10 |ackread1 |ackread2 |ackread3|read_no_ack_full,
'1' when others;

with next_state select -- only increment count when writing data
drx_cnt_inc <= '1' when write1 | write1A | write2 |  write3 | write4 | write5 | write6 | write7 | write8 | write9 | write10,
'0' when others;

with next_state select
drx_cnt_load <= '1' when address_phase,
'0' when others;

with next_state select
-- BCL changed enable state to wait_for_disable from write5|write9
 ..... drx_xfer_done <= '1' when wait_for_disable,
'0' when others;

with next_state select
dstrobe_l <= '0' when address_phase,
'1' when others;

with next_state select
dread_l <= '1' when others;

with next_state select
dblast_l <= '0' when write5 |write9 ,
'1' when others;

with next_state select
drdyin <= '1' when  write1 | write1A | write2 |  write3 | write4 | write5 | write6 | write7 | write8 | write9 | write10,
'0' when others;

with next_state select
dlad <= rx_data when write1A |write2 |write4 |write5 | write8 |write9,
"110000000000000" when address_phase,
 ..... dxlad when others;-- previous value when in other states

with next_state select
 .....dbe3 <= '1' when write4 | write5 | write8 |write9 ,
'0' when write1 | write1A | write2 | write3 | write6 | write7 |write10|idle|address_phase,
be3 when others;
with next_state select
   int_lad_en <= '1' when idle | wait_for_disable,
      '0' when others;
```

```
end architecture rxbuf_arch;
```

txbuf2_iop_temp.vhd

```
--======================= Template 1.0 ======================================
-- Design Units: Control unit for the transmit block. Data is transferred
-- from the PCI-DP to the external FIFO.
-- 9689 support to be added later.
--
-- File Name: txbuf2_iop_temp.vhd
-- Purpose:
-- Notes: Naming convention: used the names presented in the file 1_Drawing1.vsd
-- 1.  The block transfer is off by two. So, a transfer of 2000 words will require 2002
-- reads from the PCI-DP and 2000 writes to the external FIFO. The last 2 words read from
-- the PCI-DP are discarded. Note that the number of count increments (tx_cnt_inc) will be
-- 1999. There is no actual writes in the states last1 and last2.
--
-- 2. If the present_state is doing a write and the tx_cnt_full signal arrives, then
-- go to the last1 or last2 state. If the present_state is not doing a write and the
-- tx_cnt_full signal arrives, then go to the last_write1 or last_write2 states followed
-- by the last1 or last2 states respectively.
--
-- Limitations: [none]
-- Errors: [none]
-- Libraries: ieee
-- Author: Siri Velauthapillai
--Cypress Semiconductors
--3901 N. First Street, San Jose, CA, 95134
--408/432-7084
-- Environment ..... : Warp 5.2 /w Service Pack 1
--Windows NT 4.0 Service Pack 4
---------------------------------------------------------------------------
--                          Cypress Semiconductor
-- Copyright:  Copyright apply to the following intellectual property.
-- The following intellectual property may not be used or duplicated
-- without the consent of the author/designer.
-- Disclaimer: The following code is for modelling/prove of concept only and it
-- is not guaranteed.
---------------------------------------------------------------------------
-- Change History:
-- ___________________________________________________________________________
--VersionDateAuthor ini.|Changes
--1.02000-05-23 ..... |siv|New version
--1.1 .....|2000-05-24|siv|Modified to have registered outputs
--1.2 ..... |2000-06-01|siv|More bug fixes
```

```
--|.....1.3|2000-06-07|siv|Split bi-dir lad into two buses
--| 1.4|2000-06-20 |bcl|delay done signal till last cycle has finished.
----------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity txbuf is
  port (-- External FIFO signals
  txwen_l: out std_logic;
  txpaf_l: in std_logic;
 .....txdata: buffer std_logic_vector (14 downto 0);
-- Signals to/from other FSM blocks
tx_cnt_inc: out std_logic;
tx_cnt_full: in std_logic;
tx_cnt_load: out std_logic;
-- Common signals
sysclk: in std_logic;
int_rst: in std_logic;
-- Arbiter signals
tx_xfer_en: in std_logic;
tx_xfer_done: out std_logic;
-- PCI-DP (Co-Mem LITE) signals that is fed to a MUX and a Tri-State
lad_in .....: in std_logic_vector (14 downto 0);
xlad_out: out std_logic_vector (14 downto 0);--BCL: changed from lad_out to xlad_out
-- BCL: added to control external buffer
lad_out_en: out std_logic;
 ..... be3: out std_logic;-- be3 is active low, but for addressing
-- purposes, it is treated as active high
strobe_l: out std_logic;
read_l: out std_logic;
blast_l: out std_logic;
rdyout_l: in std_logic;
rdyin: out std_logic);
-- .....Attribute sum_split Of txdata : Signal Is cascaded;
end entity txbuf;

architecture txbuf_arch of txbuf is

-- SYMBOLIC ENCODED state machine: txbuf_arch
-- State code: -----  14 states
--            12345 where 12=ya(acknowledge data),na(do not acknowledge data)
--                        34=yw(write data), nw(do not write data)
--                        5 = 1 or 0( used to address the lower or upper memory[be3])

type arch_txbuf_type is (idle, address_phase, wait_for_disable, data_wait1,
```

```
                wait_last1, write_last1, last1, yanw1,nayw1,yayw1,nanw1,
                wait_last2, write_last2, last2, yanw2,nayw2,yayw2,nanw2);
-- BCL added to solve fitting problem at logic block AA
--Attribute state_encoding of arch_txbuf_type: type is one_hot_one;
Attribute state_encoding of arch_txbuf_type: type is gray;


signal present_state, next_state: arch_txbuf_type;

-- REGISTERED OUTPUT Signals
-- For registered internal lad_out bus output enable
--signal lad_out_en: std_logic;  -- BCL: moved to external signal
signal dlad_out_en: std_logic;

-- D flip flops used to generate output based on next_state
signal dtxwen_l ..................................................... : std_logic;
signal dtxdata..... : std_logic_vector (14 downto 0);

signal dtx_cnt_inc: std_logic;
signal dtx_cnt_load: std_logic;

signal dtx_xfer_done: std_logic;

signal dlad_out..... : std_logic_vector (14 downto 0);
-- BCL: commented out brought to the top.
--signal xlad_out: std_logic_vector (14 downto 0); -- siri can remove this after combining with other blocks
signal dbe3: std_logic;

signal dstrobe_l: std_logic;
signal dread_l: std_logic;
signal dblast_l: std_logic;
signal drdyin: std_logic;

begin
arch_txbuf_machine: process (present_state, txpaf_l, tx_cnt_full, tx_xfer_en, lad_in, rdyout_l)
begin

-- code to goto idle state if tx_xfer_en = '0'.
if (not(present_state = idle) and tx_xfer_en = '0') then
next_state <= idle;
else
case present_state is
-- ==================== Initial states ================================================
when idle =>
if (tx_xfer_en = '1') then
next_state <= address_phase;
```

```
else
next_state <= idle;
end if;
when address_phase =>
..... next_state <= data_wait1; -- always transition
when data_wait1 =>
.... if (rdyout_l = '0' and txpaf_l = '1' and tx_cnt_full = '0') then
next_state <= yanw1;
elsif (tx_cnt_full = '1') then
next_state <= last1;
else
next_state <= data_wait1;
end if;
-- ==================== First part of states ========================================
when yayw1 =>
.... if (rdyout_l = '0' and txpaf_l = '1' and tx_cnt_full = '0') then
next_state <= yayw2;
elsif (tx_cnt_full = '1') then
next_state <= last2;
elsif (rdyout_l = '0' and txpaf_l = '0' and tx_cnt_full = '0') then
next_state <= nayw2;
.....elsif (rdyout_l = '1' and tx_cnt_full = '0') then
next_state <= nanw2;
end if;
when nayw1 =>
.... if (rdyout_l = '0' and txpaf_l = '1' and tx_cnt_full = '0') then
next_state <= yanw2;
elsif (tx_cnt_full = '1') then
next_state <= last2;
elsif ((rdyout_l = '1' and txpaf_l = '1') or (txpaf_l = '0' and tx_cnt_full = '0') ) then
next_state <= nanw2;
end if;
when nanw1 =>
.... if (rdyout_l = '0' and txpaf_l = '1' and tx_cnt_full = '0') then
next_state <= yanw1;
elsif (tx_cnt_full = '1') then
next_state <= wait_last1;
elsif ((rdyout_l = '1' and txpaf_l = '1') or (txpaf_l = '0' and tx_cnt_full = '0') ) then
next_state <= nanw1;
end if;
when yanw1 =>
.... if (rdyout_l = '0' and txpaf_l = '1' and tx_cnt_full = '0') then
next_state <= yayw1;
elsif (tx_cnt_full = '1') then
next_state <= wait_last1;
elsif (rdyout_l = '0' and txpaf_l = '0' and tx_cnt_full = '0') then
```

```
next_state <= nayw1;
.....elsif (rdyout_l = '1' and tx_cnt_full = '0') then
next_state <= nanw1;
end if;
when wait_last1 => -- copy from lad_in to txdata but don't write;
if (rdyout_l = '0') then
next_state <= write_last1;
else
next_state <= present_state;
end if;
..... when write_last1 => -- do the actual write but don't copy
next_state <= last2;
when last1 =>
if (rdyout_l = '0') then
next_state <= wait_for_disable;
else
next_state <= last1;
end if;
-- ==================== Second part of states ========================================
when yayw2 =>
.... if (rdyout_l = '0' and txpaf_l = '1' and tx_cnt_full = '0') then
next_state <= yayw1;
elsif (tx_cnt_full = '1') then
next_state <= last1;
elsif (rdyout_l = '0' and txpaf_l = '0' and tx_cnt_full = '0') then
next_state <= nayw1;
.....elsif (rdyout_l = '1' and tx_cnt_full = '0') then
next_state <= nanw1;
end if;
when nayw2 =>
.... if (rdyout_l = '0' and txpaf_l = '1' and tx_cnt_full = '0') then
next_state <= yanw1;
elsif (tx_cnt_full = '1') then
next_state <= last1;
elsif ((rdyout_l = '1' and txpaf_l = '1') or (txpaf_l = '0' and tx_cnt_full = '0') ) then
next_state <= nanw1;
end if;
when nanw2 =>
.... if (rdyout_l = '0' and txpaf_l = '1' and tx_cnt_full = '0') then
next_state <= yanw2;
elsif (tx_cnt_full = '1') then
next_state <= wait_last2;
elsif ((rdyout_l = '1' and txpaf_l = '1') or (txpaf_l = '0' and tx_cnt_full = '0') ) then
next_state <= nanw2;
end if;
when yanw2 =>
```

```
.... if (rdyout_l = '0' and txpaf_l = '1' and tx_cnt_full = '0') then
next_state <= yayw2;
elsif (tx_cnt_full = '1') then
next_state <= wait_last2;
 elsif (rdyout_l = '0' and txpaf_l = '0' and tx_cnt_full = '0') then
next_state <= nayw2;
 .....elsif (rdyout_l = '1' and tx_cnt_full = '0') then
next_state <= nanw2;
end if;
when wait_last2 => -- copy from lad_in to txdata but don't write;
if (rdyout_l = '0')then
next_state <= write_last2;
else
next_state <= wait_last2;
end if;
 ..... when write_last2 => -- do the actual write but don't copy
next_state <= last1;
when last2 =>
if (rdyout_l = '0') then
next_state <= wait_for_disable;
else
next_state <= last2;
end if;
-- ==================== Final states ================================================
when wait_for_disable =>
if (tx_xfer_en = '0') then
next_state <= idle;
else
next_state <= wait_for_disable;
end if;
end case;
end if;
end process arch_txbuf_machine;

state_clocked:process(int_rst, sysclk)
begin
-- asyn. reset sets outputs to default values
if (int_rst = '1') then
txwen_l <= '1';
tx_cnt_inc <= '0';
tx_cnt_load <= '0';
tx_xfer_done <= '0';
strobe_l <= '1';
read_l <= '1';
blast_l <= '1';
rdyin <= '0';
```

```
txdata <= "0000000000000000";
be3 <= '0';
lad_out_en <= '1';..... -- For the tri-state

present_state <= idle;
elsif (rising_edge(sysclk)) then
txwen_l <= dtxwen_l;
tx_cnt_inc <= dtx_cnt_inc;
tx_cnt_load <= dtx_cnt_load;
tx_xfer_done <= dtx_xfer_done;
strobe_l <= dstrobe_l;
read_l <= dread_l;
blast_l <= dblast_l;
rdyin <= drdyin;
txdata <= dtxdata;
xlad_out <= dlad_out;
be3 <= dbe3;
 .....lad_out_en <= dlad_out_en;-- For the tri-state

present_state <= next_state;
end if;
end process state_clocked;

-- === Ben's Tri-State Code
--HI_Z_OUTPUTS: process (lad_out_en, xlad_out)
--Begin
--
--If lad_out_en = '1' then
--lad_out <= (others => 'Z');
--else
--lad_out <= xlad_out;
--end if;
--
--end process HI_Z_OUTPUTS;
--

-- ====================== Output Assignment based on the next state ===================
with next_state select
dlad_out_en <= '0' when address_phase ,
'1' when others;

with next_state select -- only increment count when writing data;
dtx_cnt_inc <= '1' when yayw1 | nayw1 | yayw2 | nayw2 | write_last1 | write_last2,
'0' when others;

with next_state select
```

```
dtx_cnt_load <= '1' when address_phase,
'0' when others;

with next_state select
-- ..... dtx_xfer_done <= '1' when last1 | last2,
-- BCL changed to delay done signal till last cycle has finished.
dtx_xfer_done <= '1' when wait_for_disable,
'0' when others;

with next_state select
dstrobe_l <= '0' when address_phase,
'1' when others;

with next_state select
dread_l <= '0' when address_phase,
'1' when others;

with next_state select
dblast_l <= '0' when last1 | last2,
'1' when others;

with next_state select
drdyin <= '1' when yayw1 | yanw1 | last1 | yayw2 | yanw2 | last2 | wait_last1 | wait_last2,
'0' when others;

with next_state select
dtxwen_l <= '0' when yayw1 | nayw1 | yayw2 | nayw2 |write_last1 | write_last2,
'1' when others;

with next_state select -- Siri check this?
dtxdata <= lad_in when yayw1 | nayw1 | yayw2 | nayw2 |data_wait1 |wait_last1 | wait_last2,
......"000000000000000" when idle | address_phase,
txdata when nanw1 | nanw2 | wait_for_disable | yanw1 | yanw2 |last1 |last2
| write_last1 | write_last2;

with next_state select
..... dlad_out <= "100000000000000" when address_phase,
"000000000000000" when others;

with next_state select
dbe3 <= '0' when idle | address_phase | data_wait1 | nanw1 | yanw1 | yayw2 |
.....nayw2 | last2 | wait_for_disable | wait_last1 | write_last1,
'1' when yayw1 | nayw1 | last1 | nanw2 | yanw2 |wait_last2 |write_last2;
end architecture txbuf_arch;
```

wr_lhmb_ro_op.vhd

```
--
--  File: rd_hlmb.vhd
--  Designed by: Benjamin Leung
-- Cypress Semiconductor
--  Copyright:  Copyright apply to the following intellectual property.
--  The following intellectual property may not be used or duplicated
-- ..... without the consent of the author/designer.
--Disclaimer: The following code is for modelling/prove of concept only and it
--is not guaranteed.
--Purpose: This block will write the flags status to the Local to Host Mailbox of the
--          PCI-DP. The will be written into the LHMB register in the following order
--          LHMB(2) <= DEADLOCK
--          LHMB(1) <= TX_BUF_EMPTY
--          LHMB(0) <= RX_BUF_FULL
--          LHMB(24) <= '1' -- to cause interrupt
--          This process will write to the lower 16-bits and upper 16-bits of the LHMB
--          register.
--
--  History List:
--  5/24/2000 .....BCL Created File
--6/3/2000BCL added register stage for internal flag inputs: tx_buf_empty, rx_buf_full and deadlock.
--

library IEEE;
use IEEE.std_logic_1164.all;

entity wr_lhmb is
  port (
-- Arbiter signals
wr_lhmb_en: in std_logic;
reg_access_done: out std_logic;
--
reg_tx_buf_empty: in std_logic;
reg_rx_buf_full : in std_logic;
reg_deadlock: in std_logic;

-- Common signals
sysclk: in std_logic;
int_rst: in std_logic;
-- PCI-DP (Co-Mem LITE) signals
-- .....lad: out std_logic_vector (14 downto 0);
 .....int_lad: out std_logic_vector (14 downto 0);
lad_en_l: out std_logic;
-- be3 is active low, but for addressing purposes, it is treated as active high
be3: out std_logic;
```

```
strobe_l: out std_logic;
read_l: out std_logic;
blast_l: out std_logic;
rdyout_l: in std_logic;
rdyin: out std_logic);
end entity wr_lhmb;

architecture wr_lhmb_arch of wr_lhmb is

type arch_reg_write_type is (idle, reg_wr_ad_phase, reg_wr_data_write1,
                    reg_wr_data_write2, reg_wr_access_done, reg_wr_disable_wait);
signal present_state, next_state: arch_reg_write_type;

-- Constants
--constant lhmb_lower: std_logic_vector (11 downto 0) := "000000000000";

-- For registered internal lad bus output enable
--signal lad_en_l: std_logic;
signal int_lad_en_l: std_logic;

-- For registered outputs
signal int_reg_access_done: std_logic;
--signal int_lad: std_logic_vector (14 downto 0);
signal int_int_lad: std_logic_vector (14 downto 0);
signal int_be3: std_logic;
signal int_strobe_l: std_logic;
signal int_read_l: std_logic;
signal int_rdyin: std_logic;
signal int_blast_l: std_logic;
signal int_reg_deadlock: std_logic;
signal int_reg_tx_buf_empty: std_logic;
signal int_reg_rx_buf_full: std_logic;

begin
arch_reg_wr_lhmb_machine: process (present_state, wr_lhmb_en, rdyout_l)
begin

case present_state is
when idle =>
if (wr_lhmb_en = '1') then
next_state <= reg_wr_ad_phase;
else
next_state <= idle;
end if;
when reg_wr_ad_phase =>
 ..... next_state <= reg_wr_data_write1; -- always transition
```

```
when reg_wr_data_write1 =>
if (rdyout_l = '0') then
..... next_state <= reg_wr_data_write2; -- lower 16-bit written
else
next_state <= present_state; -- waiting for RDYOUT# to go LOW
end if;
when reg_wr_data_write2 =>
if (rdyout_l = '0') then
.. next_state <= reg_wr_access_done; -- upper 16-bit written
else
next_state <= present_state; -- waiting for RDYOUT# to go LOW
end if;
when reg_wr_access_done =>
next_state <= reg_wr_disable_wait; -- done writing assert reg_access_done.
when reg_wr_disable_wait =>
if (wr_lhmb_en = '0') then
next_state <= idle; -- wait for N cycle so that enable signal will go low before going back to idle.
else
next_state <= present_state;
end if;
when others =>
next_state <= idle;
end case;
end process arch_reg_wr_lhmb_machine;

with next_state select
int_reg_access_done <=  .....'0' when idle,
'1' when reg_wr_access_done,
'0' when others;

with next_state select
int_strobe_l <= ..... '0' when reg_wr_ad_phase,
'1' when others;

with next_state select
int_read_l <= '1' when reg_wr_ad_phase,
'1' when others;

with next_state select
int_blast_l <= '0' when reg_wr_data_write2,
'1' when others;

with next_state select
int_rdyin <= '1' when reg_wr_data_write1 | reg_wr_data_write2,
'0' when others;
```

```
with next_state select
int_be3 <= '0' when reg_wr_ad_phase,
'0' when reg_wr_data_write1,
'1' when reg_wr_data_write2,
'0' when others;

with next_state select
int_lad_en_l <= '1' when idle,
‘0' when reg_wr_ad_phase | reg_wr_data_write1 | reg_wr_data_write2,
'1' when others;

with next_state select
 int_int_lad <= "000010011111000" when reg_wr_ad_phase,
 ..... ('0','0','0','0','0','0','0','0','0','0','0','0','0',
int_reg_deadlock, int_reg_tx_buf_empty, int_reg_rx_buf_full)
when reg_wr_data_write1,
"000000100000000" when reg_wr_data_write2,
"000000000000000" when others;

state_clocked:process(int_rst, sysclk)
--variable temp : std_logic;
begin
-- asyn. reset sets outputs to default values
if (int_rst = '1') then
-- Common local bus interface
strobe_l <= '1';
read_l <= '1';
blast_l <= '1';
rdyin <= '0';
be3 <= '0';
-- common state variable
present_state <= idle;
-- arch_reg_rd specific
reg_access_done <= '0';
 int_lad <= (others => '0');  -- SIV 2000-08-04 for proper reset
elsif (rising_edge(sysclk)) then
present_state <= next_state;
reg_access_done <= int_reg_access_done;
strobe_l <= int_strobe_l;
read_l <= int_read_l;
blast_l <= int_blast_l;
rdyin <= int_rdyin;
be3 <= int_be3;
lad_en_l <= int_lad_en_l;
int_lad <= int_int_lad;
int_reg_deadlock <= reg_deadlock;
```

```
int_reg_tx_buf_empty <= reg_tx_buf_empty;
int_reg_rx_buf_full <= reg_rx_buf_full;
end if;
end process state_clocked;

---- tri-state buffer for local address data bus
--HI_Z_OUTPUTS: process (lad_en_l, int_lad)
--Begin
--
--If lad_en_l = '1' then
--lad <= (others => 'Z');
--else
--lad <= int_lad;
--end if;
--
--end process HI_Z_OUTPUTS;

end architecture wr_lhmb_arch;
```

wr_lint_ro_op.vhd

```
--
--  File: wr_lint_ro.vhd
--  Designed by: Benjamin Leung
-- Cypress Semiconductor
--  Copyright:  Copyright apply to the following intellectual property.
--  The following intellectual property may not be used or duplicated
-- ..... without the consent of the author/designer.
--Disclaimer: The following code is for modelling/prove of concept only and it
--is not guaranteed.
--Purpose: This block will write the flags status to the Local Interrupt Control Status
--          of the PCI-DP. LINT(3) will be set to clear the local interrupt.
--          This process will write to the lower 16-bits of the LINT register.
--
--  History List:
--  5/24/2000BCL Created File
--
--

library IEEE;
use IEEE.std_logic_1164.all;

entity wr_lint is
  port (
-- Arbiter signals
wr_lint_en: in std_logic;
reg_access_done: out std_logic;
```

```
-- Common signals
sysclk: in std_logic;
int_rst: in std_logic;
-- PCI-DP (Co-Mem LITE) signals
-- .....lad: out std_logic_vector (14 downto 0);
.....int_lad: out std_logic_vector (14 downto 0);
lad_en_l: out std_logic;
-- be3 is active low, but for addressing purposes, it is treated as active high
be3: out std_logic;
strobe_l: out std_logic;
read_l: out std_logic;
blast_l: out std_logic;
rdyout_l: in std_logic;
rdyin: out std_logic);
end entity wr_lint;

architecture wr_lint_arch of wr_lint is

type arch_reg_lint_write_type is (idle, reg_wr_lint_ad_phase, reg_wr_lint_data_write1,
                    reg_wr_lint_access_done, reg_wr_lint_disable_wait);
signal present_state, next_state: arch_reg_lint_write_type;

-- Constants
--constant lhmb_lower: std_logic_vector (11 downto 0) := "000000000000";

-- For registered internal lad bus output enable
--signal lad_en_l: std_logic;
signal int_lad_en_l: std_logic;

-- For registered outputs
signal int_reg_access_done: std_logic;
--signal int_lad: std_logic_vector (14 downto 0);
signal int_int_lad: std_logic_vector (14 downto 0);
signal int_be3: std_logic;
signal int_strobe_l: std_logic;
signal int_read_l: std_logic;
signal int_rdyin: std_logic;
signal int_blast_l: std_logic;

begin
arch_reg_wr_lint_machine: process (present_state, wr_lint_en, rdyout_l)
begin

case present_state is
when idle =>
```

```
if (wr_lint_en = '1') then
next_state <= reg_wr_lint_ad_phase;
else
next_state <= idle;
end if;
when reg_wr_lint_ad_phase =>
...next_state <= reg_wr_lint_data_write1; -- always transition
when reg_wr_lint_data_write1 =>
if (rdyout_l = '0') then
next_state <= reg_wr_lint_access_done; -- lower 16-bit written
else
next_state <= present_state; -- waiting for RDYOUT# to go LOW
end if;
when reg_wr_lint_access_done =>
next_state <= reg_wr_lint_disable_wait ; -- done writing assert reg_access_done.
when reg_wr_lint_disable_wait =>
if (wr_lint_en = '0') then
next_state <= idle; -- wait for N cycle so that enable signal will go
-- low before going back to idle.
else
next_state <= present_state;
end if;
when others =>
next_state <= idle;
end case;
end process arch_reg_wr_lint_machine;

with next_state select
int_reg_access_done <=  .....'0' when idle,
'1' when reg_wr_lint_access_done,
'0' when others;

with next_state select
..... int_strobe_l <='0' when reg_wr_lint_ad_phase,
'1' when others;

with next_state select
int_read_l <= '1' when reg_wr_lint_ad_phase,
'1' when others;

with next_state select
.....int_blast_l <= '0' when reg_wr_lint_data_write1,
'1' when others;

with next_state select
..... int_rdyin <= '1' when reg_wr_lint_data_write1,
```

```
'0' when others;

with next_state select
int_be3 <= '0' when reg_wr_lint_ad_phase,
'0' when reg_wr_lint_data_write1,
'0' when others;

with next_state select
int_lad_en_l <= '1' when idle,
....'0' when reg_wr_lint_ad_phase | reg_wr_lint_data_write1,
'1' when others;

with next_state select
int_int_lad <= "000010011110100" when reg_wr_lint_ad_phase,
"000000011111111"when reg_wr_lint_data_write1, -- clear all local interrupts
"000000000000000" when others;

state_clocked:process(int_rst, sysclk)
--variable temp : std_logic;
begin
-- asyn. reset sets outputs to default values
if (int_rst = '1') then
-- Common local bus interface
strobe_l <= '1';
read_l <= '1';
blast_l <= '1';
rdyin <= '0';
be3 <= '0';
-- common state variable
present_state <= idle;
-- arch_reg_rd specific
reg_access_done <= '0';
int_lad <= (others => '0');  -- SIV 2000-08-04 for proper reset
elsif (rising_edge(sysclk)) then
present_state <= next_state;
reg_access_done <= int_reg_access_done;
strobe_l <= int_strobe_l;
read_l <= int_read_l;
blast_l <= int_blast_l;
rdyin <= int_rdyin;
be3 <= int_be3;
lad_en_l <= int_lad_en_l;
int_lad <= int_int_lad;
end if;
end process state_clocked;
```

```
-- tri-state buffer for local address data bus
--HI_Z_OUTPUTS: process (lad_en_l, int_lad)
--Begin
--
--If lad_en_l = '1' then
--lad <= (others => 'Z');
--else
--lad <= int_lad;
--end if;
--
--end process HI_Z_OUTPUTS;

end architecture wr_lint_arch;
```

7segment.vhd

```
--
--  File: 7_seg_decode.vhd
--  Designed by: Benjamin Leung
-- Cypress Semiconductor
--  Copyright:  Copyright apply to the following intellectual property.
--  The following intellectual property may not be used or duplicated
-- ..... without the consent of the author/designer.
--Disclaimer: The following code is for modelling/prove of concept only and it
-- is not guaranteed.
--Purpose: Designed for decoding an 8 bit word to a two digit 7-segment display.
-- using the left-hand side decimal point for overflow indication.
--
--  History List:
--  6/24/1999 .....BCL Created File
--
--
Library IEEE;
Use IEEE.Std_Logic_1164.all;
--
Library Cypress;
Use Cypress.Std_Arith.all;
--
Entity display_decode Is
   Port (
OVERFLOW_B: In Std_Logic;
MIS_DP_B: In std_logic;
ERR_Cntr: In Std_Logic_Vector(7 DownTo 0); -- Error counter
LS_SEGA_B: Out Std_Logic;
LS_SEGB_B: Out Std_Logic;
LS_SEGC_B: Out Std_Logic;
LS_SEGD_B: Out Std_Logic;
```

```
LS_SEGE_B: Out Std_Logic;
LS_SEGF_B: Out Std_Logic;
LS_SEGG_B: Out Std_Logic;
LS_DP_B: Out Std_Logic;
MS_SEGA_B: Out Std_Logic;
MS_SEGB_B: Out Std_Logic;
MS_SEGC_B: Out Std_Logic;
MS_SEGD_B: Out Std_Logic;
MS_SEGE_B: Out Std_Logic;
MS_SEGF_B: Out Std_Logic;
MS_SEGG_B: Out Std_Logic;
MS_DP_B: Out Std_Logic
   );

End display_decode;
--
Architecture display_decode_arch Of display_decode Is
--
Signal LSB_Cntr: Std_Logic_vector (3 downto 0);
Signal MSB_Cntr: Std_Logic_vector (3 downto 0);

Begin
LS_DP_B <= MIS_DP_B;
MS_DP_B <= OVERFLOW_B;
bus_split: Process(ERR_Cntr)
Begin
for i in 7 downto 4 loop
MSB_Cntr(i-4) <= ERR_Cntr(i);
end loop;
for i in 3 downto 0 loop
LSB_Cntr(i) <= ERR_Cntr(i);
end loop;
End Process;


Decode: Process (LSB_Cntr, MSB_Cntr, OVERFLOW_B)
Begin
Case (LSB_Cntr) Is
when x"0" =>
LS_SEGA_B<= '0';
LS_SEGB_B<= '0';
LS_SEGC_B<= '0';
LS_SEGD_B<= '0';
LS_SEGE_B<= '0';
LS_SEGF_B<= '0';
LS_SEGG_B<= '1';
```

```
when x"1" =>
LS_SEGA_B<= '1';
LS_SEGB_B<= '0';
LS_SEGC_B<= '0';
LS_SEGD_B<= '1';
LS_SEGE_B<= '1';
LS_SEGF_B<= '1';
LS_SEGG_B<= '1';

when x"2" =>
LS_SEGA_B<= '0';
LS_SEGB_B<= '0';
LS_SEGC_B<= '1';
LS_SEGD_B<= '0';
LS_SEGE_B<= '0';
LS_SEGF_B<= '1';
LS_SEGG_B<= '0';

when x"3" =>
LS_SEGA_B<= '0';
LS_SEGB_B<= '0';
LS_SEGC_B<= '0';
LS_SEGD_B<= '0';
LS_SEGE_B<= '1';
LS_SEGF_B<= '1';
LS_SEGG_B<= '0';

when x"4" =>
LS_SEGA_B<= '1';
LS_SEGB_B<= '0';
LS_SEGC_B<= '0';
LS_SEGD_B<= '1';
LS_SEGE_B<= '1';
LS_SEGF_B<= '0';
LS_SEGG_B<= '0';

when x"5" =>
LS_SEGA_B<= '0';
LS_SEGB_B<= '1';
LS_SEGC_B<= '0';
LS_SEGD_B<= '0';
LS_SEGE_B<= '1';
LS_SEGF_B<= '0';
LS_SEGG_B<= '0';

when x"6" =>
```

```
LS_SEGA_B<= '0';
LS_SEGB_B<= '1';
LS_SEGC_B<= '0';
LS_SEGD_B<= '0';
LS_SEGE_B<= '0';
LS_SEGF_B<= '0';
LS_SEGG_B<= '0';

when x"7" =>
LS_SEGA_B<= '0';
LS_SEGB_B<= '0';
LS_SEGC_B<= '0';
LS_SEGD_B<= '1';
LS_SEGE_B<= '1';
LS_SEGF_B<= '1';
LS_SEGG_B<= '1';

when x"8" =>
LS_SEGA_B<= '0';
LS_SEGB_B<= '0';
LS_SEGC_B<= '0';
LS_SEGD_B<= '0';
LS_SEGE_B<= '0';
LS_SEGF_B<= '0';
LS_SEGG_B<= '0';

when x"9" =>
LS_SEGA_B<= '0';
LS_SEGB_B<= '0';
LS_SEGC_B<= '0';
LS_SEGD_B<= '0';
LS_SEGE_B<= '1';
LS_SEGF_B<= '0';
LS_SEGG_B<= '0';

when x"A" =>
LS_SEGA_B<= '0';
LS_SEGB_B<= '0';
LS_SEGC_B<= '0';
LS_SEGD_B<= '1';
LS_SEGE_B<= '0';
LS_SEGF_B<= '0';
LS_SEGG_B<= '0';

when x"B" =>
LS_SEGA_B<= '1';
```

```
LS_SEGB_B<= '1';
LS_SEGC_B<= '0';
LS_SEGD_B<= '0';
LS_SEGE_B<= '0';
LS_SEGF_B<= '0';
LS_SEGG_B<= '0';

when x"C" =>
LS_SEGA_B<= '0';
LS_SEGB_B<= '1';
LS_SEGC_B<= '1';
LS_SEGD_B<= '0';
LS_SEGE_B<= '0';
LS_SEGF_B<= '0';
LS_SEGG_B<= '1';

when x"D" =>
LS_SEGA_B<= '1';
LS_SEGB_B<= '0';
LS_SEGC_B<= '0';
LS_SEGD_B<= '0';
LS_SEGE_B<= '0';
LS_SEGF_B<= '1';
LS_SEGG_B<= '0';

when x"E" =>
LS_SEGA_B<= '0';
LS_SEGB_B<= '1';
LS_SEGC_B<= '1';
LS_SEGD_B<= '0';
LS_SEGE_B<= '0';
LS_SEGF_B<= '0';
LS_SEGG_B<= '0';

when x"F" =>
LS_SEGA_B<= '0';
LS_SEGB_B<= '1';
LS_SEGC_B<= '1';
LS_SEGD_B<= '1';
LS_SEGE_B<= '0';
LS_SEGF_B<= '0';
LS_SEGG_B<= '0';

when others =>
End case;
Case (MSB_Cntr) Is
```

```
when x"0" =>
MS_SEGA_B<= '0';
MS_SEGB_B<= '0';
MS_SEGC_B<= '0';
MS_SEGD_B<= '0';
MS_SEGE_B<= '0';
MS_SEGF_B<= '0';
MS_SEGG_B<= '1';

when x"1" =>
MS_SEGA_B<= '1';
MS_SEGB_B<= '0';
MS_SEGC_B<= '0';
MS_SEGD_B<= '1';
MS_SEGE_B<= '1';
MS_SEGF_B<= '1';
MS_SEGG_B<= '1';

when x"2" =>
MS_SEGA_B<= '0';
MS_SEGB_B<= '0';
MS_SEGC_B<= '1';
MS_SEGD_B<= '0';
MS_SEGE_B<= '0';
MS_SEGF_B<= '1';
MS_SEGG_B<= '0';

when x"3" =>
MS_SEGA_B<= '0';
MS_SEGB_B<= '0';
MS_SEGC_B<= '0';
MS_SEGD_B<= '0';
MS_SEGE_B<= '1';
MS_SEGF_B<= '1';
MS_SEGG_B<= '0';

when x"4" =>
MS_SEGA_B<= '1';
MS_SEGB_B<= '0';
MS_SEGC_B<= '0';
MS_SEGD_B<= '1';
MS_SEGE_B<= '1';
MS_SEGF_B<= '0';
MS_SEGG_B<= '0';

when x"5" =>
```

```
MS_SEGA_B<= '0';
MS_SEGB_B<= '1';
MS_SEGC_B<= '0';
MS_SEGD_B<= '0';
MS_SEGE_B<= '1';
MS_SEGF_B<= '0';
MS_SEGG_B<= '0';

when x"6" =>
MS_SEGA_B<= '0';
MS_SEGB_B<= '1';
MS_SEGC_B<= '0';
MS_SEGD_B<= '0';
MS_SEGE_B<= '0';
MS_SEGF_B<= '0';
MS_SEGG_B<= '0';

when x"7" =>
MS_SEGA_B<= '0';
MS_SEGB_B<= '0';
MS_SEGC_B<= '0';
MS_SEGD_B<= '1';
MS_SEGE_B<= '1';
MS_SEGF_B<= '1';
MS_SEGG_B<= '1';

when x"8" =>
MS_SEGA_B<= '0';
MS_SEGB_B<= '0';
MS_SEGC_B<= '0';
MS_SEGD_B<= '0';
MS_SEGE_B<= '0';
MS_SEGF_B<= '0';
MS_SEGG_B<= '0';

when x"9" =>
MS_SEGA_B<= '0';
MS_SEGB_B<= '0';
MS_SEGC_B<= '0';
MS_SEGD_B<= '0';
MS_SEGE_B<= '1';
MS_SEGF_B<= '0';
MS_SEGG_B<= '0';

when x"A" =>
MS_SEGA_B<= '0';
```

```
MS_SEGB_B<= '0';
MS_SEGC_B<= '0';
MS_SEGD_B<= '1';
MS_SEGE_B<= '0';
MS_SEGF_B<= '0';
MS_SEGG_B<= '0';

when x"B" =>
MS_SEGA_B<= '1';
MS_SEGB_B<= '1';
MS_SEGC_B<= '0';
MS_SEGD_B<= '0';
MS_SEGE_B<= '0';
MS_SEGF_B<= '0';
MS_SEGG_B<= '0';

when x"C" =>
MS_SEGA_B<= '0';
MS_SEGB_B<= '1';
MS_SEGC_B<= '1';
MS_SEGD_B<= '0';
MS_SEGE_B<= '0';
MS_SEGF_B<= '0';
MS_SEGG_B<= '1';

when x"D" =>
MS_SEGA_B<= '1';
MS_SEGB_B<= '0';
MS_SEGC_B<= '0';
MS_SEGD_B<= '0';
MS_SEGE_B<= '0';
MS_SEGF_B<= '1';
MS_SEGG_B<= '0';

when x"E" =>
MS_SEGA_B<= '0';
MS_SEGB_B<= '1';
MS_SEGC_B<= '1';
MS_SEGD_B<= '0';
MS_SEGE_B<= '0';
MS_SEGF_B<= '0';
MS_SEGG_B<= '0';

when x"F" =>
MS_SEGA_B<= '0';
MS_SEGB_B<= '1';
```

```
MS_SEGC_B<= '1';
MS_SEGD_B<= '1';
MS_SEGE_B<= '0';
MS_SEGF_B<= '0';
MS_SEGG_B<= '0';
when others =>
End case;
End Process;


End display_decode_arch;
```