To the Graduate Council:

I am submitting herewith a thesis written by Robert Earl Leugers entitled "The Development of a TMS320C4x Serial Communications Board." I have examined the final copy for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Bruce W. Bomar

Bruce W. Bomar, Major Professor

We have read this thesis and

recommend its acceptance:

L Montgomery Smith

Roy S Joseph

Accepted for the Council:

Associate Vice Chancellor and Dean
of the Graduate School

The Development of a TMS320C4x Serial Communications Board

A Thesis Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Robert Earl Leugers

December 1996

ACKNOWLEDGMENTS

I thank Dr. Bruce Bomar, Dr. Roy D. Joseph, and Dr. L. Montgomery Smith for their advice and assistance in the preparation of this thesis. I also wish to thank Sverdrup Corporation for the loan of the parts and some of the equipment I used for this project. I wish to express my appreciation to the Altera corporation for donating the software and programming hardware I used. The equipment was donated through their University Support Program. I especially thank the United States Air Force PALACE Acquire Program for funding my studies at UTSI. I would not have been able to work on this thesis at UTSI if it were not for the program.

ABSTRACT

This thesis presents the development of a circuit board which converts the bi-directional bit-parallel communications port data bus of the Texas Instruments TMS320C4x family of digital signal processors into a high speed bit-serial data bus. The usual method of interconnecting Texas Instruments TMS320C4x digital signal processors is by ribbon cables connected to parallel communications ports. The cables are cumbersome and do not allow reliable data communications over distances greater than one meter.

The board overcomes this limitation by using a Cypress CY7B923 Hotlink transmitter to convert from eight bit parallel to ten bits serial using an internal 8B/10B data encoder. The transmitter sends the data at bit rates from 160 to 330 megabits per second either through a category-5 unshielded twisted pair cable or through multimode fiber optics. A Cypress CY7B933 Hotlink receiver decodes and converts the data from ten bit serial back to eight bit parallel. The maximum length for a category-5 cable connected to the Hotlink transmitter/receiver pair is 150 feet at a serial data rate of 250 megabits/second.

The board uses three state machines compiled into an Altera EPM7129ELC84-15 programmable logic device. One state machine is responsible for communicating with and transferring data to the TMS320C4x digital signal processor. Another state machine

is responsible for transmitter control and data transmission. The last state machine controls the receiver. Each of the state machines was implemented using VHDL hardware description language in the Altera Max Plus II development environment.

The TMS320C4x communications board has been built and its capabilities tested. The board has allowed a TMS320C40 processor to send data across a 15 foot category-5 unshielded twisted pair cable at 4.9 megabytes per second. This data rate should be the same using a 150 foot cable.

Two suggestions are made for improving the data rate. First, the speed at which the state machines are operating can be improved by compiling the VHDL code into a faster programmable logic device. Second, the 'C4x interface can more quickly detect an attempted read to or write from a TMS320C4x processor if the synchronization flip flops used by the 'C4x state machine are removed.

TABLE OF CONTENTS

CHAPTER PAGE

List of Figures

Chapter 1

Introduction

The Texas Instruments TMS320C4x ('C4x) family of digital signal processors is designed for parallel processing and other real-time embedded applications. Each 'C4x processor has six 16 megabyte per second communications ports for a processor operating at 40MHz [1]. These ports allow the processor to be connected in either simple or complex parallel processing geometries.

The speed at which the 'C4x processors communicate depends on the distance between the processors. The processors can communicate with one another at 16 megabytes per second if the processors are less than an inch apart. This data rate decreases with distance to about 11.5 MBytes per second at three feet. Communications are unreliable at distances longer than three feet using ribbon cables [2].

These processors are used in large scale digital signal processing systems [3]. Some applications include communications systems, medical imaging systems, SONAR, RADAR, and turbine engine signal processing systems. Systems used in these applications must have a very large throughput and meet real-time processing requirements. These requirements are best met by the parallel/distributed hardware

configurations made possible by digital signal processors such as the TMS320C4x family.

Each communications port has an eight-bit data bus and a four-bit control bus. The current method for connecting the processors is with ribbon cable. The cable plugs into a keyed 26-pin header that is connected to a communications port. Every other conductor in the cable is grounded for noise immunity.

The TMS320C4x serial communications board was designed around the Cypress Semiconductor CY7B923 Hotlink transmitter and CY7B933 Hotlink receiver integrated circuits, which convert eight bit parallel data into encoded ten bit serial data. The data rates at which the circuits can operate serially are from 160 Mbits per second to 330 Mbits per second. Other integrated circuits were selected for this project considering which devices would work best with the parts and equipment already selected or available.

Category-5 unshielded twisted pair cable (Cat-5 UTP) was the serial cable used in this design. Cat-5 UTP is inexpensive, easy to use, and is capable of carrying a signal of 330 Mbits per second for distances up to 38 meters, a great improvement over the capabilities of ribbon cable.

A printed circuit board (PCB) was designed for this project using HIWIRE II computer aided design software. The PCB was designed as a dual board which would fit a VME chassis; two communications boards are available on each PCB. The PCB was

built, and the integrated circuits and discrete components were placed on the board. The controlling state machines were built and placed in a programmable logic device (PLD). The board was functionally tested, and performance parameters were measured.

The average data rate of the board was measured to be 4.9 MBytes per second, for a serial data rate of 250 Mbits per second over a 15 foot Cat-5 UTP cable. The rate at which data is written into the board is 5.4 MBytes per second. The speed for data to be written out of the board is 5.4 MBytes per second.

The board was also checked for proper operation. All control functions passed the tests. Data was written into the board, transmitted through the serial link, and read from the receiver.

Chapter 2 will provide background information useful for understanding the design of the system. Chapter 3 will give an overview of the design. Chapter 4 will discuss the details of the design. Chapter 5 presents the results obtained after the design and development of the system, and Chapter 6 concludes the thesis with a brief summary and recommendations for improving the design.

Chapter 2

Background

Overview

The 'C4x family of Digital Signal Processors (DSPs) communicates through bi-directional communication links connected to communication ports on the processor. A communication link consists of a bi-directional asynchronous byte parallel data bus and a four line control bus. The communications protocol uses four bi-directional control lines, nCREQ, nCACK, nCSTRB, and nCRDY, to ensure that only one of the two processors connected by a link can send data at a time and that the data has been received. The lower case n in front of the signal name signifies an active low signal [1].

TMS320C4x Data Communications

The two 'C4x DSPs connected by a link arbitrate ownership of the data bus using a token/no token system. A processor with the token has bus ownership. At power-up, communication ports one, two and three, the default output ports, have the token, and ports four, five and six, the default input ports, do not. When connecting processors, a default output port is connected to a default input port.

The 'C4x is a 32-bit processor. A word transmitted through a communications port must be broken up into four bytes. The most significant byte is sent first.

Ownership of the data bus can be transferred only after a whole word has been sent. Consider Figure 2-1, and assume that DSP A has the token. 'C4x A has ownership of the data bus, and 'C4x B can only receive data from A.

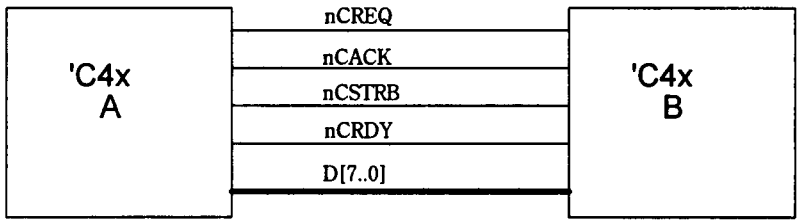


Figure 2-1: Connection of Two 'C4xs

The initial state of the control lines at processor A is that nCREQ and nCRDY are at high impedance, and nCACK , nCSTRB, and D[7..0] are driven. Processor B is driving nCREQ and nCRDY and has nCACK, nCSTRB and D[7.0] at high impedance.

In Figure 2-2, DSP A has data to transmit to B. First, A drives the data onto D[7..0] (#1). After a short delay, A brings nCSTRB low (#2) . Processor B sees nCSTRB go low, signifying there is valid data on the bus. DSP B, after reading the data, then brings nCRDY low (#3), telling A that it has read the data. DSP A brings nCSTRB high, and B drives nCRDY high (#4), signaling that it is ready for a new byte of data.

The process runs four bytes before a token transfer can occur. These actions transfer a 32 bit word.

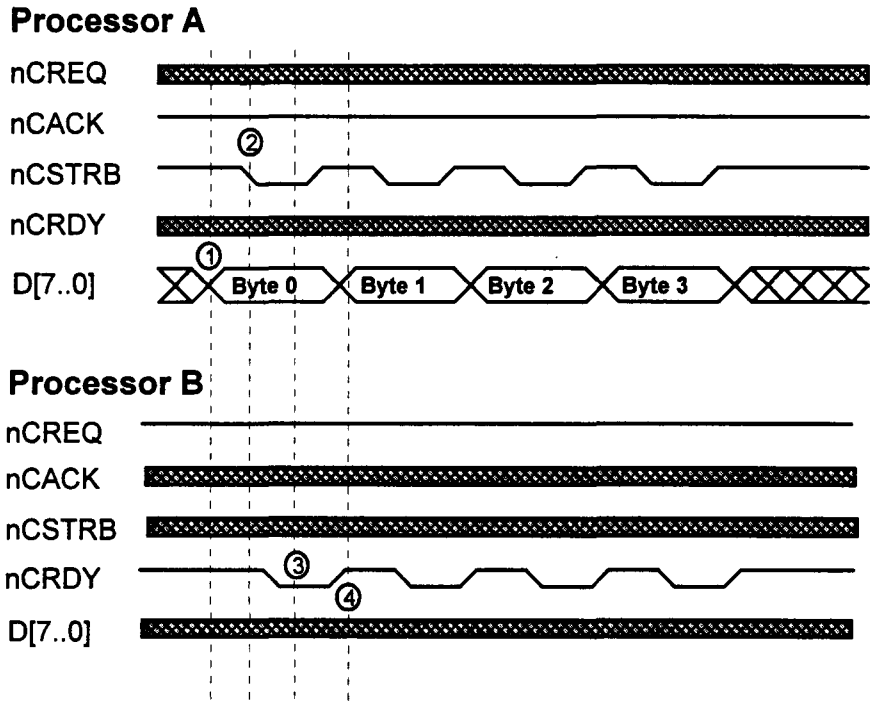


Figure 2-2: TMS320C4x Word Transfer Operation

When DSP B has data to send to DSP A, the token can be transferred. The process is started (see Figure 2-3) when DSP B drives nCREQ low (#1). nCREQ can go low at any time during the data transfer operation. After a word transfer is completed, processor A brings nCACK low (#2). The token is now exchanged. The control lines and data bus must be turned around. Following a short delay after nCACK goes low, DSP A changes nCRDY from high impedance to high (#3). Then processor A stops driving data on the bus by bringing the data lines to a high impedance state (#4).

Processor B changes nCSTRB from high impedance to high. A short delay later, B brings nCREQ high (#5). 'C4X A brings nCACK high and then puts it and nCSTRB in high impedance (#6).

DSP B places nCREQ and nCRDY in high impedance and nCACK high after bringing nCREQ high (#7). The bus and control lines are now turned around and DSP B can send data to DSP A.

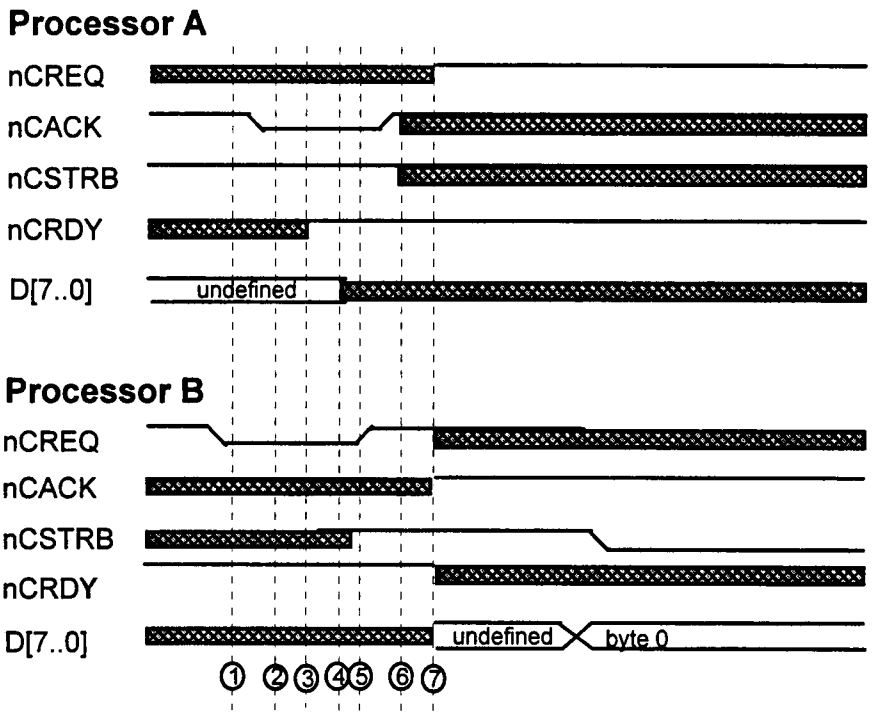


Figure 2-3: TMS320C4x Token Transfer Operation

Serial Data Communications

The TMS320C4x digital signal processor communicates at data rates of up to 20 megabytes per second for a 50MHz processor. Thus, a serial protocol that can attain this and allow real time communications must have a data rate of 160 megabits per second.

The Cypress CY7B923 Hotlink transmitter and the CY7B933 Hotlink receiver were chosen as the chipset to realize the serial communications. The chips can communicate at data rates from 160 Mbits per second to 330 Mbits per second with an intrinsic bit error rate of less than $3 \bullet 10^{-12}$ bits per second, operating at 330 Mbits/sec for a 125 foot category-5 unshielded twisted pair cable [4]. At the time of selection, these chips were the only integrated circuits capable of handling the data speed requirements.

The chipset uses 8B/10B encoding [4]. For every byte loaded in parallel to the transmitter, ten bits are encoded and transmitted serially. The bits are received by the receiver, error-detected, and decoded. Eight bits are then written out of the receiver in parallel. When data is not being sent, the transmitter chip sends a special fill character. This action ensures that there are always transitions on the serial line for synchronization purposes.

Because the serial line has continuous transitions, a transformer can be used to couple the cables onto a long copper line. The communications sources are isolated from one another in this manner, and the transformer helps to filter line noise.

The encoding scheme also allows for special characters to be defined. Through these characters, control information can easily be passed from one side of the data link to the other .

The 8B/10B encoder is constructed from a 5B/6B encoder and a 3B/4B encoder operating in parallel. This feature is evident when the notation for the data characters and special characters is examined. The format for the data characters is Dxx.yy, where "xx" and "yy" are decimal values. "xx" is the character name for the data encoded by the 5B/6B encoder, and "yy" is the character encoded by the 3B/4B encoder. An example is D5.2, where 5 is encoded by the 5B/6B encoder and 2 by the 3B/4B encoder.

The format for a special character is Kxx.yy, where xx and yy are decimal values. As in the case of the data, "xx" is the character name for the data encoded by the 5B/6B encoder, and "yy" is the character encoded by the 3B/4B encoder. K25.1 , for example, refers to a special character.

Data characters run from D0.0 to D31.7. All 256 eight bit data combinations are covered by this code. Twelve special characters are available. The special characters are K28.0, K28.1, K28.2, K28.3, K28.4, K28.5, K28.6, K28.7, K23.7, K27.7, K29.7 and K30.7. Since K28.5 is used by the system as a fill sequence to keep the transmitter/receiver pair in synchronization, only eleven characters are available to a system designer. When more control options are needed, two or more special characters can be combined to make extra control sequences or symbols.

Figure 2-4 shows a Hotlink transmitter and receiver connected by a serial cable. To transmit data, data must first be placed on the data bus D[7..0]. nENA, transmitter enable, and SC/nD, Special Character not Data, are brought low simultaneously to load the data. The data is encoded and sent over the serial cable. Special characters are transmitted in the same manner, except that SC/nD remains high through the sequence.

The serial transmission rate is ten times the CLK, clock, rate. The transmitter sends data to the receiver, where a phase locked loop keeps the receiver in synchronization with the data. The signal CKR, read clock, is an output signal which is derived from the incoming serial data. The signal frequency is the serial data rate divided by ten and is in phase with the incoming serial data.

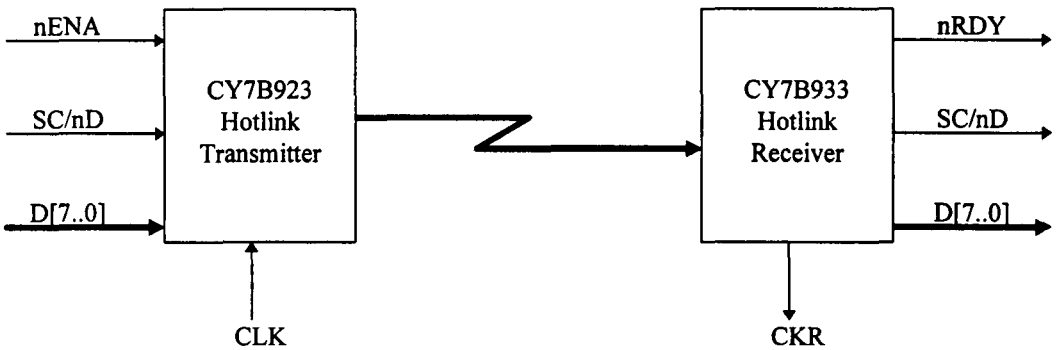


Figure 2-4: Interconnection of Hotlink Transmitter and Receiver

Data Storage

Data which is to be sent through the serial link must be stored until the DSP can download it. Also, in case data transmission through the data link is temporarily stopped,

data must be stored until it can be transmitted. Thus, data should be adequately buffered to reduce losses.

One of the simplest types of memory chips available for data buffering which is first in first out memory (FIFO). When data is written into a FIFO, the internal logic keeps track of the location of the data within the memory. When data is read, the first data which was stored in the chip is the first to appear. No external address bits are necessary to retrieve the data.

FIFOs come in two types, synchronous and asynchronous. Asynchronous FIFOs do not need a clock to synchronize their functions. Although synchronous FIFOs require a clock for internal logic functions to operate, the read and write clocks do not need to run at the same frequency.

The memory chosen for this project was the Cypress Semiconductor CY7C453-14 2K x 9 Synchronous FIFO. Some features of this product are a tri-state buffer on the output data lines and a set of status flags, nEF, nHF, and nPAFE. The flags indicate how much data is stored in memory. Table 2-1 shows the flag truth table [5].

Table 1: FIFO Flag Truth Table

nEF	nPAFE	nHF	STATE	Number of Words in FIFO.
0	0	1	Empty	0
1	0	1	Almost Empty	1 to 16
1	1	1	≤ Half Full	17 to 1024
1	1	0	> Half Full	1025 to 2031
1	0	0	Almost Full	2032 to 2047
0	0	0	Full	2048

Communications Control

The Hotlink chipset and the TMS320C4x communications must be controlled and sequenced to interface the Hotlink with the 'C4x. State machines must be provided to control all the chips involved. Because constructing large state machines to control each device cannot be easily constructed with discrete TTL or CMOS chips, programmable logic devices were used.

In the past, PLDs offered more flexibility then discrete components connected on a board. However, erasing, programming, and placing the code inside the parts was difficult. Currently, PLDs are available which are simple to use. They can be easily erased and programmed. Higher level hardware description languages such as the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) allow the design of code which is self documenting and easy to understand and maintain. Altera, Cypress Semiconductor, Advanced Micro Devices, and other companies offer devices which can hold many logic functions and high level compilers to help program them.

Summary

The TMS320C4x digital signal processor communicates from six ports using 8 bi-directional data lines and 4 bi-directional control lines. It uses a token passing scheme to arbitrate bus ownership. It can communicate at a data rate of 160 megabits per second.

The Cypress Semiconductor CY7B923 Hotlink transmitter and the CY7B933 Hotlink receiver chipsets can communicate from 160 to 330 megabits per second. The transmitter/receiver pair uses 8B/10B encoding which can send data or special characters to be used for control. When data is not being sent, the encoder sends synchronization characters. Because the transmission rate is constant, transformer coupling can be used to isolate the communications line from the boards.

Data which is to be transmitted through or received from the Hotlink chipset should be stored in memory until it is needed. First in first out memories offer a simple alternative for the required buffering.

Controlling integrated circuits can be accomplished using programmable logic devices. High level hardware description languages such as VHDL are available for programming the programmable logic devices. A programmer can use VHDL to write self-documenting code.

Chapter 3

Design Overview

Overview

A communications board which allows 'C4x DSPs to communicate at full speed must communicate at speeds of up to 160 Mbits per second. Incoming and outgoing data must be buffered to ensure the best possible throughput of data. The integrated circuits on the board should be controlled with a programmable logic device large enough to hold the required logic, and it must be fast enough to avoid creating a bottleneck during the communications process. Figure 3-1 is a block diagram of the system.

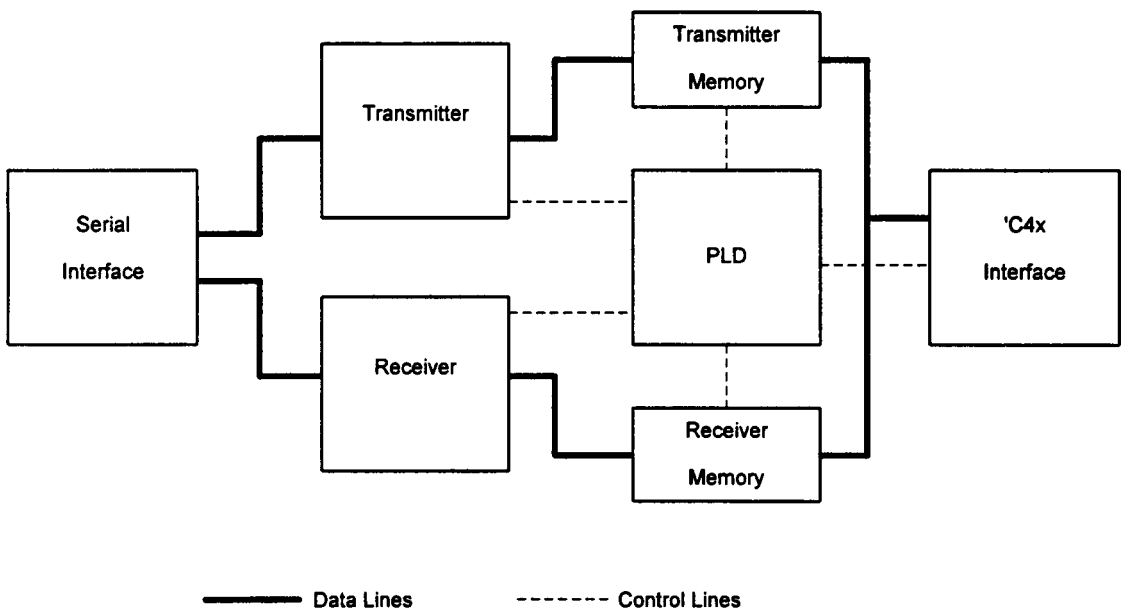


Figure 3-1: General Block Diagram of System

Block Diagram of System

To use the Cypress Hotlink transmitter/receiver chipset, the communications board must connect to other communications boards via a serial interface. The interface must be capable of transmitting the data stream at the 160 to 330 Mbits per second rates specified for the Hotlink transmitter and receiver chipsets. The interface should allow reliable connections for distances of at least 100 feet.

The transmitter and receiver memory should store enough data to provide a substantial buffer in case the data link should be temporarily halted at the receiving end. The buffer should be fast and simple to use so that the logic required for control is minimized.

The transmitter and receiver used in this project were the Cypress Semiconductor CY7B923 and 'B933 Hotlink chipset. At the time of selection, these chips were the only chips capable of providing the speed and ease of operation desired, so they were the only ones really considered.

Altera 7000 series programmable logic devices were used because of the ready availability of equipment and software to support these devices. Various devices within the series could have been used. The type of device which was used was determined during the design process. The PLD used must be fast enough to provide the best possible throughput and large enough to hold logic generated from well documented and understandable VHDL code.

VHDL was the language used to program the PLD. The language is a Department of Defense and IEEE standard. VHDL is flexible and allows for writing self-documenting code. A weakness with VHDL is that the compilers used for generating the logic do not necessarily provide the optimum logic reduction and fitting into a device. Reasons for using VHDL included following the standards, providing maintainable code, and potentially allowing the design to be ported from one make of PLD to another.

The TMS320C4x interface must communicate and transfer data on the bi-directional data bus via a 26-wire ribbon cable. Except for external signal conditioning devices and physical connectors, the interface can be implemented in a programmable logic device. The interface mainly affects the selection of the type of PLD.

Chapter 4

Detailed Design Description

Overview

The design of the TMS320C4x communications board can be divided into four parts: serial cabling and connector, receiver, transmitter, and 'C4x communications. Three state machines, residing in an Altera EPM7128ELC84-15, PLCC package, are needed to control the integrated circuits used on the board. The state machines control the 'C4x communications, receiver, and transmitter sections of the design.

Cabling and Connector

The serial cabling and connectors were selected based on their ability to handle the required data rates, ease of use, costs, and useable length of the cable. Cables considered were Category-3 (Cat-3) Unshielded Twisted Pair (UTP), Cat-5 UTP, Shielded Twisted Pair-1 (STP-1) and RG-59 coaxial cable.

Cat-3 and Cat-5 UTP are cables used in local area networks for ethernet, fast ethernet, or asynchronous transfer mode computer networking. Both types of cable use RJ-45 connectors and sockets. Each cable has four cable pairs. The cables and connectors are available through computer network suppliers and are inexpensive. Cat-3 UTP has a lower bandwidth than Cat-5.

Shielded Twisted Pair is a shielded version of the above cables. The cable uses a DB-9 connector. STP cable costs more than UTP cable. STP connectors are also larger than UTP connectors.

RG-59 coaxial cable requires two cables and connectors for each transmitter and receiver pair. This option would take up too much board space for connectors. Connecting several boards using this option would result in a much more complex arrangement than using any of the twisted pair cables.

Category-5 UTP was selected for use with this project. Table 2 shows maximum line length for several data rates [4].

Table 2: Category 5 Unshielded Twisted Pair Maximum Line Length for Data Rates

Data Rate (Mbits/sec)	Maximum Line Length (m)
160	80
200	65
250	52
280	45
330	38

Because longer distances or connection in a very noisy environment may be required, the 'C4x communications board was also designed for use with a fiber optics transceiver. The device intended for use with this design is the Hewlett Packard HFBR - 5302. This device is a 1300 nm multimode fiber optic transceiver. The connector is an SC-duplex type connector, which allows one cable to be used for each transmitter/receiver pair.

The HFBR-5302 was chosen for three reasons. First, the multimode fiber optic cable was less expensive than single mode cable. Second, no OSHA safety regulations cover the light emitting diodes used by this module, which is not the case for single mode units using laser diodes. Third, although the cost for similar modules from other manufacturers is approximately the same, the HFBR-5302 was recommended by technical support engineers from Cypress Semiconductor.

Table 3 presents the distances available when using the Hotlink receiver and transmitter chips with the HFBR-5302 and stepped-index multimode fiber [6].

Table 3: Stepped-Index Multimode Fiber Data Rate vs. Maximum Line Length

Data Rate (Mbits/sec)	Maximum Line Length (m)
160	125
200	100
250	80
280	71
330	61

If graded-index fiber optic cable is used, the bandwidth-distance product can be as high as 2.5 GHz • km. Table 4 shows the possible distances for this type of cable [6].

Table 4: Graded-Index Multimode Fiber Data Rate vs. Maximum Line Length

Data Rate (Mbits/sec)	Maximum Line Length (km)
160	15.6
200	12.5
250	10.0
280	8.9
330	7.6

Transmitter Section

The transmitter section, the upper half of Figure 3-1, consists of a CY7C453 synchronous first in first out (FIFO) memory circuit, CY7B923 Hotlink Transmitter, and controlling state machine, TxLnk (the VHDL code listing is in the Appendix), contained within the EPM7128ELC84-15 PLD. Figure 4-1 is a block diagram of the transmitter section.

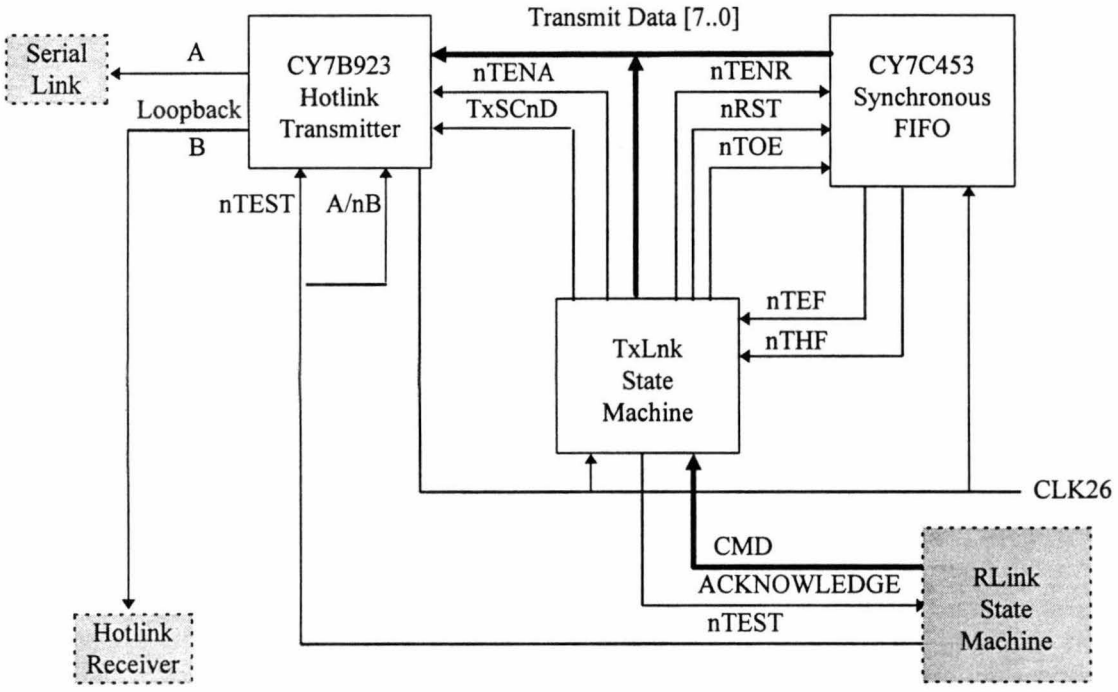


Figure 4-1: Block Diagram of Transmitter Section

A typical data transmission begins when the TxLnk, by examining the nTHF and nTEF flags, detects that data has been written into the FIFO by the 'C4x interface (see Table 1 for the decoding of the nTEF and nTHF flags). When data is detected in the

FIFO, the state machine reads one byte of data and latches it at the output terminal of the FIFO by bringing nTENR, the FIFO read line, and nTOE, output enable, low on one clock cycle and high on the next. The state machine brings transmit enable, nTENA, low and send transmit or special control character, TxSCnD, low on the next clock cycle, writing the data into the Hotlink transmitter. The transmitter encodes the data, serializes it, and sends it to the opposite receiver at ten times the clock rate, CLK26, the main transmitter clock running at 25 MHz.

TxLnk can also send control symbols by placing the desired special character on the data bus, and bringing TxSCnD high and nTENA low. nTENR and nTOE remain high for this operation.

The control symbols consist of two special characters sent one after another. The system is designed to send the following symbols:

1. LSTAT -- link status check. K28.0 followed by K28.0. This symbol is used to check periodically that the serial link is still operational.

2. HALT -- halt symbol. K28.0 followed by K28.1. This symbol is used to request a stop to the data being transmitted by the opposite transmitter. This symbol is generated on a command from the receiver state machine when the receive FIFO becomes full.

3. ENABLE -- enable symbol. K28.0 followed by K28.2. This symbol is used to notify the opposite transmitter that it can send data again after

it has been halted by #2. This symbol is generated on a command from the receiver state machine when the receive FIFO becomes less than half full after a HALT has been requested.

This method allows for up to 121 control messages to be generated.

The receiver state machine, RLink, controls TxLnk through a four bit command bus, CMD_LINES[3..0]. RLink can disable or enable the transmitter. The receiver state machine can also command each control symbol except LSTAT to be generated. LSTAT is generated after the built in self test (BIST) sequence is passed, and every 1024 clock cycles afterwards.

The loopback line is used to check the functionality of a transmitter/receiver pair. During the built in self test, the nTEST line becomes low. This line is also connected to A/nB, a pin on the Hotlink transmitter. When A/nB is high, data is directed through serial line A; when low, data is directed through B.

Serial line B is connected to the serial input on the Hotlink receiver. nTEST is used not only to initiate the BIST but also to direct the test sequence to be transmitted on serial line B. The test sequence is checked by the Hotlink receiver, which signals when the test sequence is complete.

If the transmitter is disabled by the receiver state machine, the transmitter stops sending data from the transmit FIFO. The ability to send control symbols is not disabled.

This gives the transmitter/receiver pair the capacity to check and control the data link at all times.

Receiver Section

The receiver section (the lower half of Fig. 3-1) is made up of a Cypress Semiconductor CY7C453 synchronous FIFO memory, Cypress Semiconductor CY7B933 Hotlink receiver, and controlling state machine. The controlling state machine, RLink (the VHDL code listing is in the Appendix), is one of three state machines placed into an Altera EPM 7128 ELC-15 programmable logic device. Figure 4-2 is a block diagram of the receiver section.

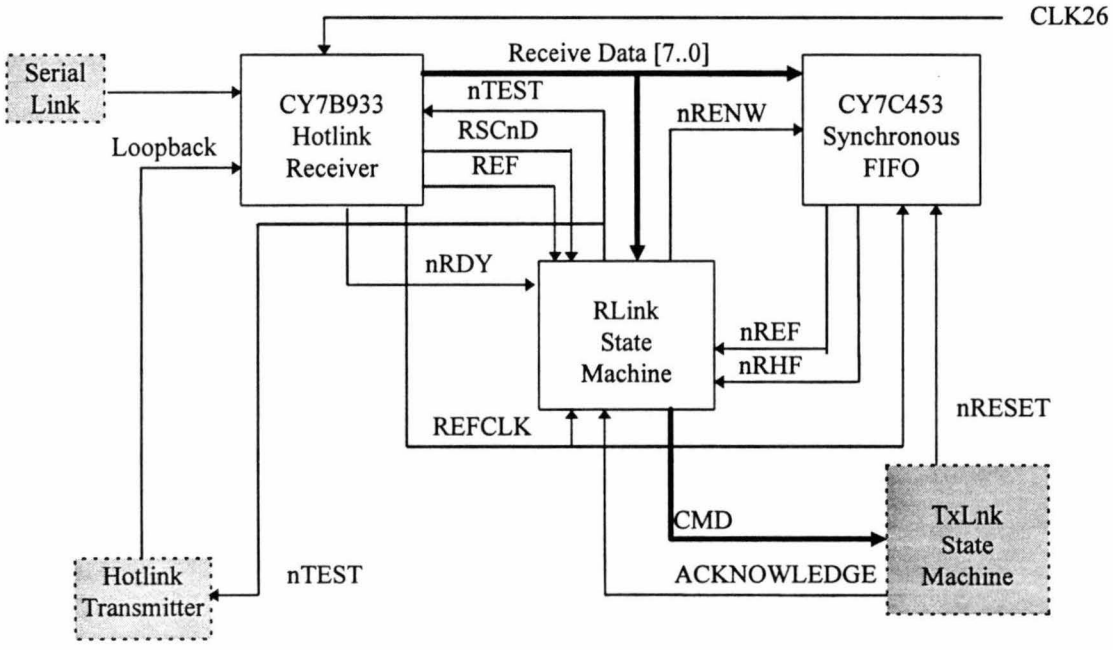


Figure 4-2: Block Diagram of Receiver Section

The normal data read operation begins when the Hotlink receiver gets data, decodes and error detects it, places the data on the eight-line output bus, and signals that the data is ready by bringing nRDY low. The receiver also signals whether the data is a control character or valid data, using a high or low signal, respectively, on RSCnD.

The RLink state machine monitors RSCnD and drops nRENW if valid data appears at the terminal. If a special character, K28.0, appears, the state machine moves to a state in which it watches for the next special character. If no special character appears, the RLink moves to a state in which it waits for data or a special character to appear at the output of the receiver. If a defined special character is output, the receiver state machine takes appropriate action for that code. Control symbols and the actions taken are:

1. LSTAT: The action taken is to light a light emitting diode to signal that the data link is functional. LSTAT is also required after the built in self test is initiated to enable the receiver section.

2. HALT: This code signals the receiver to halt transmitter operation until the receiver receives an enable symbol.

3. ENABLE: This code tells the receiver to enable the transmitter to send data again and is valid only after a HALT symbol is received.

As mentioned above, the receiving state machine is the link controller. When one of the control symbols is received, the state machine places the appropriate command on

CMD_LINES and waits for the transmitter state machine to acknowledge the action using the ACKNOWLEDGE line.

RLink also monitors how much data is present in the receive FIFO. If the FIFO becomes full, the state machine requests TxLnk to send a HALT symbol by placing the appropriate command on CMD_LINES. When the FIFO becomes less than half full, the receiving state machine requests the ENABLE symbol to be sent, signaling that there is enough free memory to store data again.

RLink is responsible for initiating the built-in self test. At power up or at system reset, it brings the nTEST line low, initiating a loopback test. The transmitter sends a test pattern of all possible codes on the loopback line. When a full pattern has been received correctly, the Hotlink receiver chip brings nRDY high, signaling the end of a test sequence.

The state machine waits for eight test sequences to pass before ending the loopback test. Next, the point-to-point data link is tested. The receiver state machine waits for an LSTAT to be sent before writing to the receive FIFO. This verifies a valid connection between a transmitter and a receiver and keeps any data from the test sequence from being written into the receive FIFO.

A final function of the receiver state machine is to reframe the data. An unresolveable error in the received code will cause RVIOL to go high, signaling a

decoding error. Because the bit error rate (less than 3×10^{-12}) is so low, it can be assumed that the most common cause of a coding violation is misframed data [2].

Misframed data results when the receiver loses bit synchronization due to noise on the serial communications channel. Misframed data occurs at the transmitter-receiver level and is not controllable at the state machine level, except that subsequent data can be realigned by bringing REF high, reframing the data.

Any time RVIOL is high, the receiver state machine responds by bringing REF high. This action ensures that the data is realigned immediately, and that data loss is reduced.

TMS320C4x Interface Section

Figure 4-3 is a block diagram of the 'C4x interface. This section consists of the 'C4x communications protocol state machine, PROTOSM (VHDL code listing is in the Appendix), connected to the transmit and receive section FIFOs, along with some Schmidt triggers and the 26-pin header for connecting to a ribbon cable.

The 'C4x state machine, PROTOSM, uses the jumper at power up or reset to determine if it has the token, or initial bus ownership. If the jumper is set, the state machine has the token and can transfer data to the digital signal processor. Otherwise, the state machine is set up to receive data. Data and token transfer are accomplished using the TMS320C4x protocol described earlier.

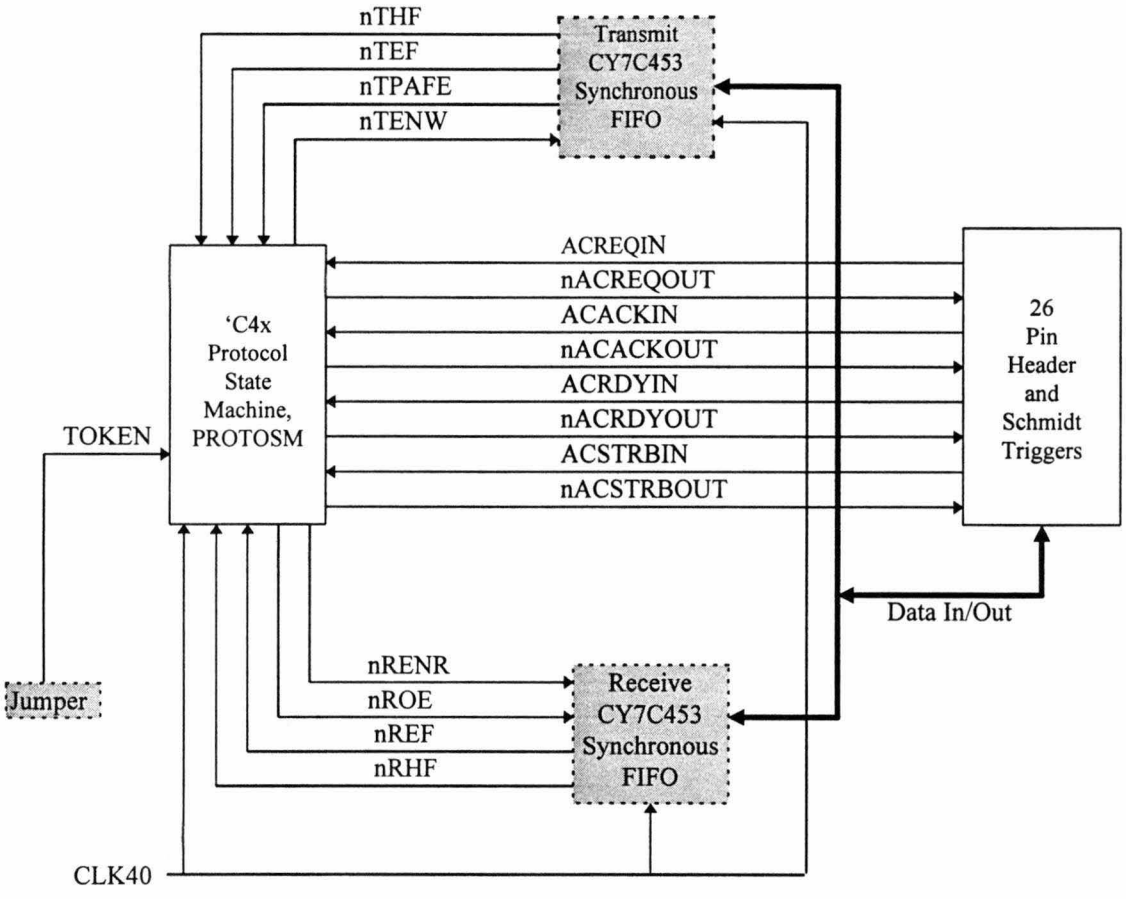


Figure 4-3: The TMS320C4x Communications Section

If the state machine owns the data bus and data is received into the receive FIFO, the FIFO flags, nREF and nRHF, will reflect that data is present. The state machine then transfers a word of data into the DSP. After this word transfer, the 'C4x can receive the token so that it can send data across the data link. If the receive FIFO reflects that data is present and the state machine no longer has the token, it requests and regains possession of the bus and transmits a word of data to the DSP. This operation is defined as two-way

communication and operates similarly if the state machine starts up without the token as determined by the jumper in Fig. 4-3.

One-way communication occurs when the 'C4x continuously sends data into the data link with no token transfer. This operation maximizes the speed at which the data can be transferred from one processor to another, since the data bus does not have to be turned around.

Whether one-way or two-way communications is in process for a data link, data transfer into the transmit FIFO halts when the FIFO flags, nTEF, nTHF, and nTPAFE, reflect that the FIFO is full. Data can be transferred again when the flags reflect a condition other than full.

The signals ACSTRBIN and nACSTRBOUT are tied together at the 'C4x interface. ACSTBIN is the signal nCSTRB as it comes from the DSP through a 74HC14 Schmidt trigger inverter. The Schmidt trigger inverts the signal coming from the DSP and provides some noise immunity for the communications line. nACSTRBOUT bypasses the inverter and is used only when the conditions are appropriate for the state machine to drive that line. ACRDYIN and nACRDYOUT, ACREQIN and nACREQOUT, and ACACKIN and nACACKOUT are connected at the interface in the same manner. Refer to the circuit schematic in the Appendix for connection details.

Summary

The TMS320C4x serial communications board can communicate using either Category-5 unshielded twisted pair cables and connectors or a 1300 nm multimode fiber optic transmitter/receiver pair. The fiber optic transceiver uses a duplex SC-type connector.

The board consists of several passive and active components requiring controlling logic. The controlling logic consists of three state machines: TxLnk, to control the transmitter section; RLINK, for controlling the receiver section; and PROTOSM, the TMS320C4x communications protocol controller. The state machines coexist in an ALTERA EPM7128ELC84-15 programmable logic device.

Chapter 4

Checkout and Results

Overview

The TMS320C4x serial communications printed circuit board was designed and electrically checked using a computer assisted drafting and design tool. The board was then built and subjected to several tests to check and correct operation and to benchmark the one-way data transfer speed. A data dependent encoding error was encountered for three of the 256 data bytes which could be transmitted through the system. Although several attempts were made to find the source of the error, it could not be resolved. A work around substituting three control symbols for these three data characters is proposed.

Design Implementation

The 'C4x serial communications printed circuit board was designed using HIWIRE II CAD software. This tool provides a schematic entry option for creating electrical schematics, PCB routing and autorouting utilities, netlist extraction, and electrical design checking.

The design was entered into the software using the schematic editor tool. The schematic was checked with the netlist extraction tool, which reports errors in the

schematic. The design was corrected and rechecked until the netlist extractor reported that a successful netlist was created. The finished schematic is located in the Appendix.

Next, the PCB was drawn. This was done by first drawing an outline of the board. The outline of the board was divided to make the TMS320C4x serial communications board a dual board. After the outline was done, the footprints of each of the components were placed inside the outline of each section of the board. The footprints either were found in the drawing libraries of HIWIRE II or had to be created. The footprints were placed in a position conducive to running signal traces to the required pins on the PCB.

Some of the signal traces were entered manually at this point. These traces were the serial communications from the CAT-5 RJ-45 connector to the transformer, the traces from the transformer to the Hotlink serial input and output, and the system clocks. Keeping these traces short and direct was desirable to ensure that they were quiet.

The autorouter utility was then used to connect the signals over the rest of the board. After the autorouting, the signal traces were checked for erroneous and meandering connections, which were corrected by hand. Finally, because the autorouter draws corners using 90° angles, each autorouted trace was rounded to 45° angles.

The routed board is six layers. The top two layers and the bottom two layers are signal traces with the inner two layers being a power and ground plane. Wherever possible, the traces on the top layer run at a right angle to the traces on the inner top layer.

The bottom two layers were laid out using a similar method. This layout scheme helps to reduce crosstalk.

After the PCB was drawn, the drawing was checked and rechecked with the design rule check utility, until no more errors were reported. The artwork was produced with another utility, and the board was constructed by a contractor.

All three controlling state machines were written in VHDL using Altera Maxplus II version 5.4 development tools. Some features of this environment are schematic entry, ability to program most varieties of Altera PLDs, ability to design for speed or minimum space used in the part, design simulation, and timing analyzer.

The initial versions of the state machines were designed and programmed. They were compiled to fit into an EPM 7128ELC84-15 part. The simulation and timing analyzer were used to check performance and initial functionality of the design to ensure the part selected was correct for the design needs. Once this was verified, the VHDL compiler was allowed to select the pinout of the PLD to be used in the design and layout of the printed circuit board.

Checkout Method

The board was built and checked for short circuits. Using a digital multimeter, first the power and ground planes were checked. Then many signal traces were checked at random. The results were that no short circuits existed either between power and ground or among the checked traces.

The state machines were compiled into the PLD and placed in their sockets. Then the board was powered up, as the final and ultimate test for short circuits. No short circuits existed anywhere on the board.

Overall functionality was the next step. The steps in this checkout plan and methods for accomplishing it were:

1. Check that the self test and link test pass. This was done by watching for a light emitting diode to light when the self test passes and another to light when the serial link is good.

2. Transfer data into the transmit first in first out memory. This step checks the handshaking between the TMS320C4x processor and the 'C4x communications state machine. This test was verified with a logic analyzer.

3. See that data is being written from the FIFO into the Hotlink transmitter. This verifies the operation of the transmitter state machine. This step was checked with a logic analyzer.

4. Check that data is received by the Hotlink receiver and written into the receive FIFO. The operation of the receiver state machine is verified in this step. This operation was checked by a logic analyzer.

5. Verify that data is written from the receive FIFO into the communications port on the TMS320C4x. This step completes the check of the ability for one-way data communications. This was seen with a logic analyzer.

6. Calculate performance parameters. Parameters which were calculated at this time are average link speed (the time for several words to be written into the transmitter and out of the receiver), speed for data to be written into the board, and the speed at which data is written out of the board. These parameters were measured using a logic analyzer.

7. Check the bi-directional operation of the TMS320C4x data bus. This is the final step in verification of the 'C4x controlling state machine. This test was verified using a logic analyzer.

8. Check the operation of the halt/resume states. This is the final check of the transmitter and receiver state machines. This step was accomplished using an oscilloscope.

During the overall functionality check, when a design error in the hardware or the state machines was found, it was corrected. The next functionality check was not started until the error was corrected and the test was passed.

Results

The printed circuit board did not have any shorts either between power and ground or between signal traces. The system passed both the built in self test and link status check by lighting the appropriate light emitting diodes.

For all tests, the clock used to run the transmitter and receiver pair at both ends of the link was 25Mhz. The data was transmitted across the serial data link at 250 Mbits/sec under this condition. The length of the CAT-5 UTP used for these tests was 15 feet. The length of the ribbon cable used to connect the processor to the board was about 2.5 feet. The best data rate which can be expected for transferring data into and out of the serial communications board is for this cable length is about 11.43 MBytes per second [2]. The Figure 5-1 is a diagram of the test setup.

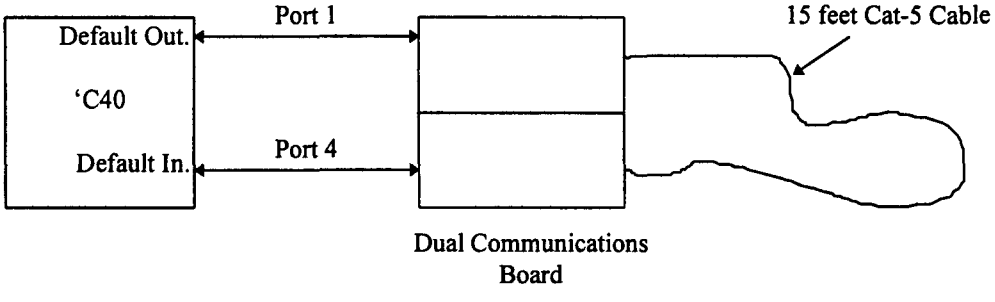


Figure 5-1: Test Setup

The test processor was a single TMS320C40 digital signal processor with a system clock operating at 40MHz. The processor was connected to the serial

communications board from port one, which will power up with the token, and port four, which will power up without the token.

The logic analyzer used for the subsequent steps was a Boulder Creek Engineering, Incorporated, Pod-A-Lyzer™ 8020. Features of this system include an ability to capture data either synchronously, at rates greater than 50 MHz, or asynchronously from 1ksamples to 100 Msamples per second. The analyzer has 18 input channels with a 64 kilobit trace buffer for each channel. It can trigger on word recognition, logic low or logic high and detect rising or falling signal edges. The logic analyzer plugs into a communications port on a personal computer and was controlled using software based in Microsoft Windows 3.11.

The TMS320C4x communications board passed the first test quickly. Both the light emitting diodes lighted, signaling the completion of the built in self test and link status. The board was ready to be checked to see if data was being written into the transmit FIFO.

Figure 5-2 shows the data as it was being written from a TMS320C40 digital signal processor into the FIFO. The DSP lowered the signal on nCSTRB b, signaling that eight bits of data were ready. The state machine lowered nTENW, writing the data into the FIFO memory, and then brought nCRDY b low. The DSP then raised the signal level on nCSTRB b to logic 1, and the 'C4x communications state machine brought nCRDY b

high. This process continued for one full 32 bit word. The FIFO flag nTEF changed its signal level, showing that the memory was storing data.

When nTENR was brought low, data appeared at the output pins of the FIFO memory. Then nTENA was lowered, writing data into the Hotlink transmitter. The result of this action was seen in the signals nRENW and nREF. When the data was received at the output of the receiver, the state machine detected it and wrote the data into the receive FIFO. The FIFO flag nREF changed state to reflect that the memory was storing data. Tests three and four were passed.

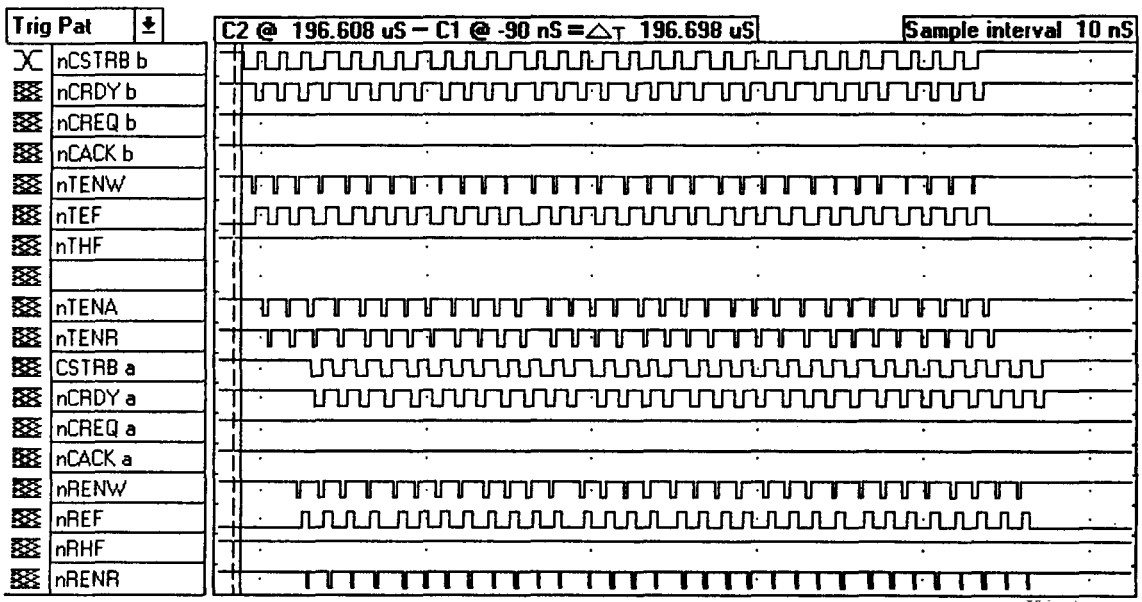


Figure 5-2: Data Transfer Through the TMS320C4x Serial Communications Board

Step five can also be seen in Figure 5-2. The data transfer operation to the receiving port of the DSP began with bringing nRENR low. Eight bits of data were

latched at the output of the receive FIFO. The 'C4x communications state machine lowered nCSTRB a, signaling the processor that data was ready to be read. The DSP read the data and lowered nCRDY to signal the state machine that the data had been read. The process continued for one 32 bit word.

At this point the data was examined as it came out of the DSP. Data written into the board was written out of the board correctly with three exceptions to be discussed later.

Figure 5-3 shows the measurement for average link speed. For this measurement, eight four-byte words were written out of port 1 on the TMS320C40 DSP and then received at port 4 of the same processor. The time for eight words to be written into and out of the serial communications board was 6560 nsec.

$$\frac{8 words}{6560 n \sec} = 1.22 Mwords / \sec = 4.9 Mbytes / \sec$$

Figure 5-4 shows the measurement for the speed at which data was written into the board. The cursors measure 5970 nsec for eight 32 byte words.

$$\frac{8 words}{5970 n \sec} = 1.34 Mwords / \sec = 5.4 Mbytes / \sec$$

This measurement shows that data was being written into the board at a faster data rate than the average data transfer rate. A potential exists for the transmit FIFO to fill up,

(depending on how much data the 'C4x DSP sends) since the data was being loaded into the memory faster than it was being transmitted..

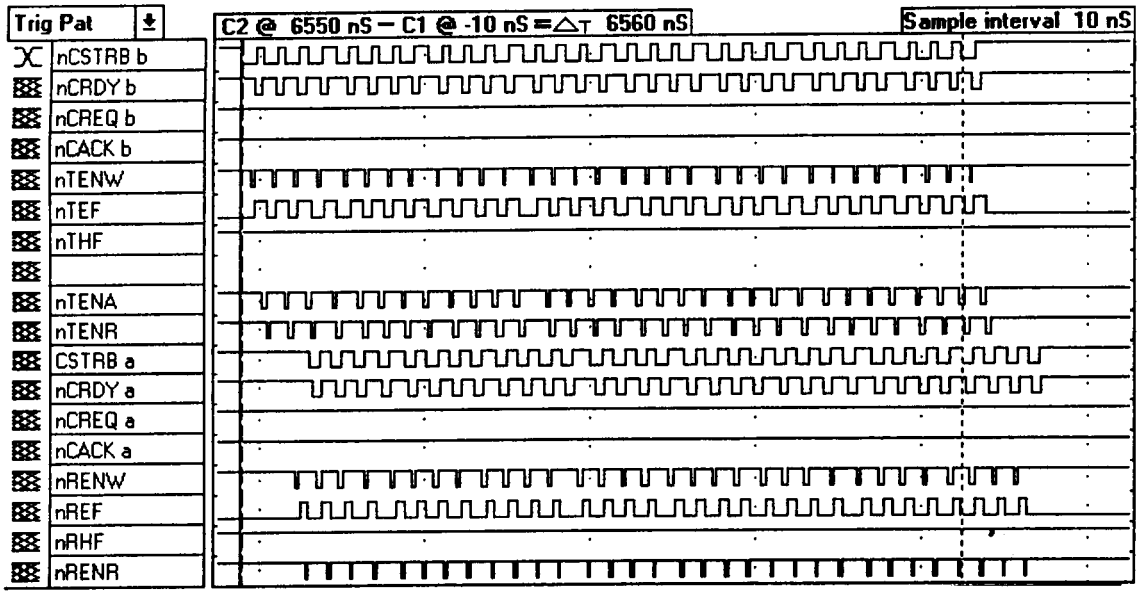


Figure 5-3: Time for Data to be Sent Through System

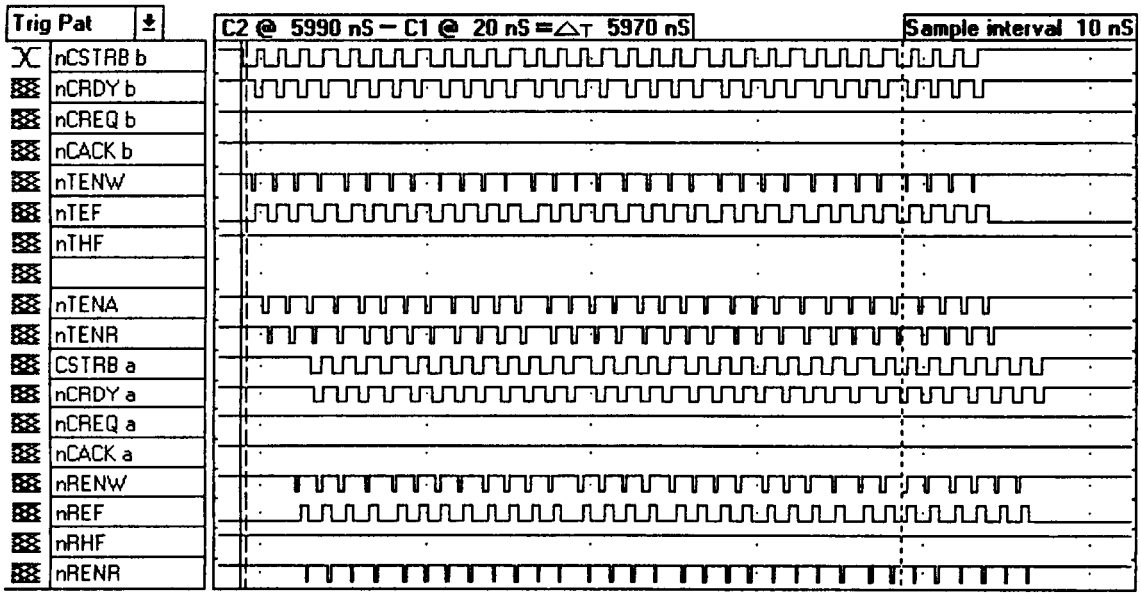


Figure 5-4: Time for Data to be Written Into the Board

Figure 5-5 shows the rate at which data was being written out of the receive FIFO and into the TMS320C40 DSP.

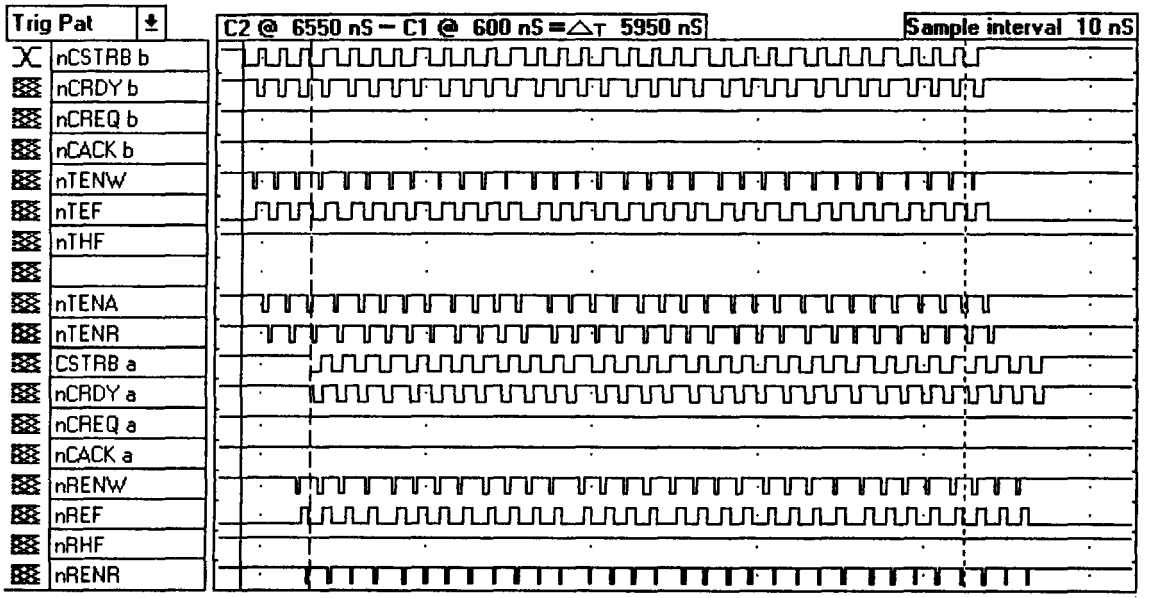


Figure 5-5: Time to Write Data Out of Board

Eight words were written out of the receive FIFO in 5950 nsec.

$$\frac{8 words}{5950 n \sec} = 1.34 Mwords / \sec = 5.4 Mbytes / \sec$$

Thus the 'C40 was reading data from the board at approximately the same rate as it was writing the data to the board.

Checkout step seven involves changing the ownership of the data bus. Figure 5-6 shows the results for this test.

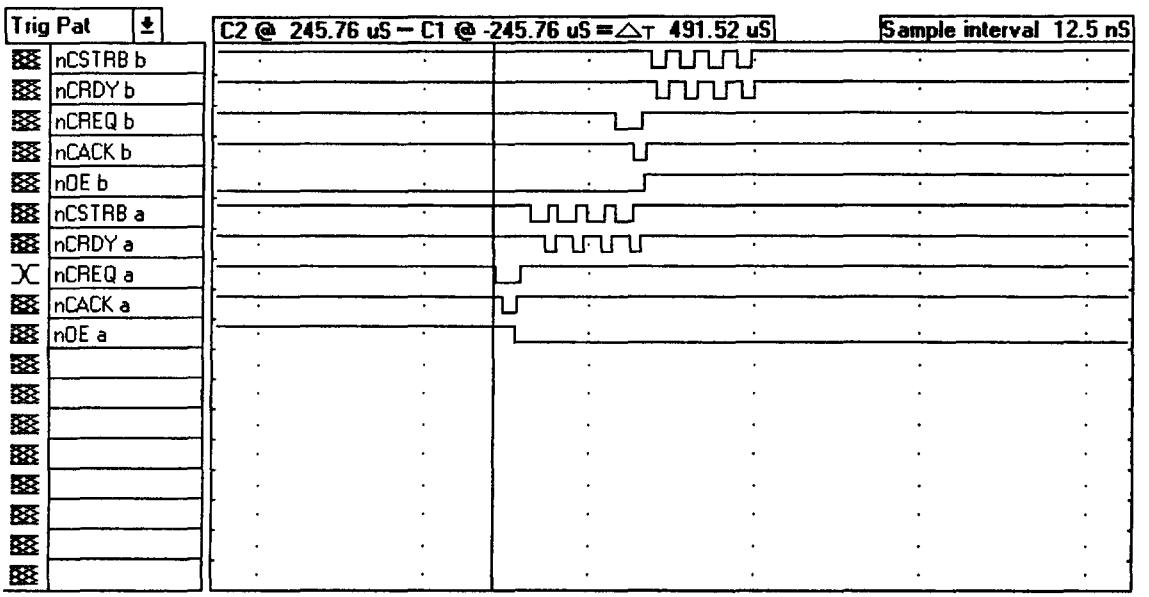


Figure 5-6: Token Transfer

For this test, data was written into 'C40 port 4, which by default does not have bus ownership, causing a token transfer. nCREQ a went low, requesting the token. The token transfer request was acknowledged when the PROTOSM, the 'C4x state machine, lowered nCACK a. PROTOSM drove nCACK a high, and both the control and data bus changed ownership. The signal nOE a showed a change in bus ownership by a change in logic level. Data was then written from 'C40 port 4 into the board.

The data which was written into the board was transmitted through the data link and written into the receive FIFO at the other end of the data link. This link was

connected to 'C40 port 1, which by default had the token. To transfer the data, PROTOSM must gain bus ownership. The token was requested by lowering the signal nCREQ b. The 'C40 DSP acknowledged the request by lowering nCACK b. PROTOSM saw this signal change and raised nCREQ b. The DSP raised nCACK b when nCREQ b went high. The bus changed ownership, shown by a change in logic level on nOE b, and a word was written into the TMS320C40 processor.

Test number eight, the check of the HALT and ENABLE states, was accomplished by first writing two kilobytes plus 32 bytes of data into the TMS320C4x serial communications board. This action filled up the receive FIFO and placed 32 bytes of data into the transmit FIFO.

The flags for both the transmit and receive FIFOs were checked with an oscilloscope to verify this condition. nTENA was checked next to see if any data was being written into the Hotlink transmitter. The FIFO flags reported that the receive FIFO was full and the transmit FIFO was less than or equal to half full. nTENA was not being asserted, demonstrating that data was not passing through the serial link. The HALT state passed this test.

The next action involved reading 1 kilobyte plus 48 bytes of data from the receive FIFO. The FIFO flags reported the receive FIFO was less than half full after this action, and the transmit FIFO was empty. The board entered the ENABLE state after the HALT state was engaged. Checkout step number eight was passed.

One problem was encountered during checkout which could not be resolved. A work around has been specified to correct this problem. A data encoding error occurs when any one byte of data containing the integer 32, 33, or 34 is transmitted through the Hotlink chipset. The RVS line on the receiver goes high, signaling that an error has occurred, and the violation code for a data coding error appears at the output of the receiver. All other eight bit combinations pass through the system without error.

Technical support at Cypress Semiconductor indicated this problem could be caused by improper termination or bias voltage on the transmission lines. Further investigation of this problem involved trying different termination schemes as well as a direct connection from one side of the dual board to the other, bypassing the cat-5 cable and the bias network. Termination resistors of values from 37.5 Ω to 50 Ω were soldered directly to the serial input pins on the Hotlink receiver. Bias voltages were varied in the range of three volts to four volts (the Hotlink data sheet recommends 3.5 volts). The category-5 cable and bias network were disconnected from the circuit by opening a dip switch, and a direct connection was made between a Hotlink transmitter and receiver. In this case, 50 Ω termination was placed directly on the receiver pins with an 82 Ω pull-up resistor to +5 volts and a 130 Ω pull-down resistor to ground. In all cases, the serial link failed to transmit and receive 32, 33, or 34.

A solution to this unresolved problem is to substitute control symbols for the values which cause problems. A K28.1--K28.0 can be sent instead of 32, K28.1--K28.1

can be sent instead of 33, and K28.1--K28.2 instead of 34. When received, these symbols would signal the receiver state machine to place a 32, 33, or 34 into the receive FIFO at the time the symbol is recognized.

This work around can not be implemented using the Altera EPM7128ELC84-15 used in this project, due to the insufficient logic and routing resources currently remaining in the PLD. However, a pin-compatible EPM7160ELC84 PLD could be purchased which should have enough resources to perform the work around.

Summary

The printed circuit board for the TMS320C4x communications board was designed and tested using a CAD tool. The PCB was made, and the board was built. The board was subjected to several tests to check functionality. A data dependent encoding error was noted, and a recommended work around was suggested.

Chapter 5

Conclusion and Recommendations

Conclusion

Most of the objectives for this thesis have been met. A working TMS320C4x serial communications board has been developed and tested, although a data-dependent encoding error must be worked around on three of the 256 encodable data bytes.

The board allows communications across a 15 foot category-5 cable at 4.9 MBytes per second, which is adequate for many applications but is less than the 11.5 MBytes per second data rate which a 'C4x DSP can achieve with practical ribbon cable lengths. Although the board was tested only at distances up to 15 feet, the board should allow 'C4x processors to communicate at 4.9 MBytes per second for distances up to 38 meters --much farther than is possible with the one meter limitation of ribbon cables.

The bottleneck in the system was in the design of the transmitter state machine, TxLnk, and in the speed at which the 'C4x state machine operates. TxLnk transitions through a state in which it detects data in the transmit FIFO, another in which it latches the data at the output of the FIFO, and a third in which the state machine transmits the data. Thus, three clock cycles pass before a byte of data is sent through the data link.

The board can transmit data only as fast as it is loaded. PROTOSM takes a clock cycle through a synchronizing flip flop to detect that the 'C4x wishes to transfer data, and four states to write one byte of data into the FIFO and acknowledge the data transfer. Five clock cycles are needed to write one byte into the system, and 16 clock cycles pass before a full word is written into the FIFO. These are the factors which currently limit the data rate.

Recommendations

Although the board will not currently allow the 'C4x processors to communicate at full data rate, the board is still a potentially valuable tool. Two steps must be taken to speed up the operation of the board.

First, the speed at which the 'C4x communications state machine runs must be improved. Currently in the Altera EPM7128ELC84-15 part, the fastest that the state machine can be clocked is 45MHz. The recommended speed grade of the PLD is 10nsec. The use of a 10nsec part should allow the use of a 60MHz or better clock for the state machine. Data should be transferred at a much faster rate between the board and the processor, eliminating this data bottleneck.

Second, to improve the throughput of the serial communications board, an attempt should be made to reduce the overhead involved in data transfers. The number of states PROTOSM and TxLnk use to transfer data could be examined, and reduced, if possible.

The state assignments might also be redesigned to use one-bit adjacent states to remove synchronization flip flops.

Another recommendation is that the board should be tested with a fiber optic transceiver. Funds were not available during this project to purchase the necessary transceivers and cable. The board might have to be used in a noisy environment, and performance characteristics for the fiber optic link should be recorded.

Finally, an effort should be made to reduce the size of this board. This board was designed to be used in a VME chassis. Someone in the future might wish to add the function of this board to a card which does not have the space a VME or a VXI chassis can allow.

Bibliography

Bibliography

Skahill, Kevin 1995. VHDL for Programmable Logic. San Jose, CA: Cypress Semiconductor

Wakerly, John F. 1994. Digital Design Principals and Practices. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Widmer, A. X. and Franaszek, P. A. *A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code.* IBM Journal of Research and Development, 27 No. 5:440-451, September 1983.

Altera Data Book, March 1995, Altera Corporation, San Jose CA

Fibre Channel-Physical and Signaling Interface (FC-PH). ANSI X3.230-1994. 1994, New York, NY: American National Standards Institute

List of References

References

[1] TMS320C4x Users Guide, 1994, Dallas Texas, Texas Instruments Incorporated.

[2] Sorensen, Brian E., Design of a Digital Crossbar/Crosspoint Switch for TMS320C4x Digital Signal Processors. December 1993, Masters Thesis, University of Tennessee.

[3] Sztipanovits, J., Wilkes, D. M., Biegl, C., and Lynd, L. E. *The Multigraph and Structural Adaptivity.* IEEE Transactions on Signal Processing Vol. 41 No. 8, August 1993.

[4] Cypress Hotlink Users Guide, June 1995, San Jose, CA: Cypress Semiconductor.

[5] High Performance Data Book, 1993, San Jose CA: Cypress Semiconductor.

[6] Keiser, Gerd 1991. Optical Fiber Communications. New York, NY: McGraw-Hill, Inc.

Appendix

VHDL Code Dictionary

```
library ieee;
use ieee.std_logic_1164.all;

package CodeDefs is

        subtype Code is std_logic_vector (3 downto 0);
        subtype Sp_Char is std_logic_vector (7 downto 0);

        ------------------------------------------------------------------------
        -- Code Dictionary
        ------------------------------------------------------------------------

        -- CMD_FLAGS codes

        constant BUS_IDLE               :       Code := "0000";
        constant SEND_HALT              :       Code := "0001";
        constant SEND_ENABLE            :       Code := "0010";
        constant HALT_TRANSMITTER       :       Code := "0011";
        constant ENABLE                 :       Code := "0100";
        constant TESTING                :       Code := "1111";


        -- Special Codes

        constant K280                   :       Sp_Char := "00000000";
        constant K281                   :       Sp_Char := "00000001";
        constant K282                   :       Sp_Char := "00000010";
        constant K283                   :       Sp_Char := "00000011";
        constant HI_Z                   :       Sp_Char := "ZZZZZZZZ";

        ------------------------------------------------------------------------
        -- K280,K280    Link Status
        -- K280,K281    Halt Transmitter
        -- K280,K282    Resume Transmitter
        ------------------------------------------------------------------------


end CodeDefs;
```

Hotlink Top Level

```
--------------------------------------------------------------------
--
-- This is the top-level file for the C4x Communication boards
-- controlling logic.
--------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;

library work;
use work.codedefs.all;

entity Hotlink is port
        (
        -- Pin definitions for Transmitter Controller.

        nRST            :       in std_logic;
        CLK26           :       in std_logic;
        RCLK            :       in std_logic;
        CLK40           :       in std_logic;
        nTEF            :       in std_logic;
        nTPAFE          :       in std_logic;
        nTHF            :       in std_logic;
        nRESET          :       out std_logic;
        nTENR           :       out std_logic;
        nTEST           :       out std_logic;
        nTENA           :       out std_logic;
        nTENN           :       out std_logic;
        MODE            :       out std_logic;
        FOT             :       out std_logic;
        SVIOL           :       out std_logic;
        TXSCnD          :       out std_logic;
        nTOE            :       out std_logic;
        TQ              :       out std_logic_vector (7 downto 0);
        nREF            :       in std_logic;
        nRPAFE          :       in std_logic;
        nRHF            :       in std_logic;
        RD              :       in std_logic_vector (7 downto 0);
        nRDY            :       in std_logic;
        RVIOL           :       in std_logic;
        RSCnD           :       in std_logic;
        nRENW           :       out std_logic;
        LSTAT           :       out std_logic;
        REF             :       out std_logic;
        ATOKEN          :        in std_logic;
        nROE            :       out std_logic;
        ACREQIN         :       in std_logic;
        nACREQOUT       :       out std_logic;
        ACACKIN         :       in std_logic;
        nACACKOUT       :       out std_logic;
        ACRDYIN         :       in std_logic;
```

```
        nACRDYOUT       :       out std_logic;
        ACSTRBIN        :       in std_logic;
        nACSTRBOUT      :       out std_logic;
        nTENW           :       out std_logic;
        SO              :       in std_logic;
        CARDET          :       out std_logic;
        nRENR           :       out std_logic;
        nRP             :       in std_logic
        );

end Hotlink;

architecture Netlist of Hotlink is

------------------------------------------------------------
-- TxCtl signals.
------------------------------------------------------------

component PROTOSM  -- C4X State machine

        port
        (
         CLK40              : in std_logic;
         nRST               : in std_logic;
         ATOKEN             : in std_logic;
         nTPAFE             : in std_logic;
         nREF               : in std_logic;
         nRHF               : in std_logic;
         nTEF               : in std_logic;
         nTHF               : in std_logic;
         nROE               : out std_logic;
         ACREQIN            : in std_logic;
         nACREQOUT          : out std_logic;
         ACACKIN            : in std_logic;
         nACACKOUT          : out std_logic;
         ACRDYIN            : in std_logic;
         nACRDYOUT          : out std_logic;
         ACSTRBIN           : in std_logic;
         nACSTRBOUT         : out std_logic;
         nTENW              : out std_logic;
         nRENR              : out std_logic
   );

end component;

component TxLnk

        port
        (
        CLK26           :       in std_logic;
        nRP             :       in std_logic;
        CMD_FLAGS       :       in std_logic_vector (3 downto 0);
```

```
        ACK             :       out     std_logic;
        TEST_PASSED     :       in std_logic;
        nTEF            :       in std_logic;
        nTHF            :       in std_logic;
        nRST            :       in std_logic;
        nRESET          :       out std_logic;
        nTENA           :       out std_logic;
        nTENR           :       out std_logic;
        TXSCnD          :       out std_logic;
        nTOE            :       out std_logic;
        TQ              :       out std_logic_vector (7 downto 0)
        );

end component;

component RLink

        port
        (
        RCLK            :               in std_logic;
        ACK             :               in std_logic;
        nRPAFE          :               in std_logic;
        nREF            :               in std_logic;
        nRHF            :               in std_logic;
        nRST            :               in std_logic;
        RD              :               in std_logic_vector (7 downto 0);
        nRDY            :               in std_logic;
        RVIOL           :               in std_logic;
        RSCnD           :               in std_logic;
        CMD_LINES       :               out std_logic_vector (3 downto 0);
        nRENW           :               out std_logic;
        TEST_PASSED     :               out std_logic;
        nTEST           :               out std_logic;
        LSTAT           :               out std_logic;
        REF             :               out std_logic
        );

end component;

component dff           -- Located in Altera VHDL library
        port
        (
        D                       : in STD_LOGIC;
        CLK                     : in STD_LOGIC;
        CLRN                    : in STD_LOGIC;
        PRN                     : in STD_LOGIC;
        Q                       : out STD_LOGIC
        );
end component;
```

```
signal ACKa, TEST_PASSEDa     : std_logic;
signal CMD_FLAGSa             : std_logic_vector (3 downto 0);
signal VCC                    : std_logic;
signal nTEFq, nTHFq, nTPAFEq  : std_logic;
signal nREFq, nRHFq, nRPAFEq  : std_logic;
signal ACACKINq, ACREQINq     : std_logic;
signal ACSTRBINq, ACRDYINq    : std_logic;

begin -- netlist

--These pins are not currently to be used for control.

nTENN         <= '1';
MODE          <= '0';
FOT           <= '0';
SVIOL         <= '0';
CARDET        <= TEST_PASSEDa;
VCC           <= '1';

-------------------------------------------------------------
-- Component instantation statements.
-------------------------------------------------------------
SYNCH1:       dff port map (nTEF, CLK26, nRST, VCC, nTEFq);
SYNCH2:       dff port map (nTHF, CLK26, nRST, VCC, nTHFq);
SYNCH4:       dff port map (nREF, CLK40, nRST, VCC, nREFq);
SYNCH5:       dff port map (nRHF, CLK40, nRST, VCC, nRHFq);

SYNCH6:       dff port map (ACREQIN, CLK40, nRST, VCC, ACREQINq);
SYNCH7:       dff port map (ACRDYIN, CLK40, nRST, VCC, ACRDYINq);
SYNCH8:       dff port map (ACSTRBIN, CLK40, nRST, VCC, ACSTRBINq);
SYNCH9:       dff port map (ACACKIN, CLK40, nRST, VCC, ACACKINq);

xmtctl:  TxLnk port map (CLK26, nRP, CMD_FLAGSa, ACKa, TEST_PASSEDa,
                nTEFq, nTHFq, nRST, nRESET,
                nTENA, nTENR, TXSCnD, nTOE, TQ(7 downto 0));

rcvctl:  RLink port map (RCLK, ACKa, nRPAFE, nREF, nRHF, nRST,
                                        RD( 7 downto 0), nRDY, RVIOL, RSCnD,
                                        CMD_FLAGSa, nRENW, TEST_PASSEDa, nTEST,
                                        LSTAT, REF);

c4xctl: protosm port map ( CLK40, nRST, ATOKEN, nTPAFE, nREFq, nRHFq,
                nTEF, nTHF, nROE, ACREQINq, nACREQOUT, ACACKINq,
                nACACKOUT, ACRDYINq, nACRDYOUT, ACSTRBINq, nACSTRBOUT,
                nTENW, nRENR);

end Netlist;
```

Receiver State Machine

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

library work;
use work.codedefs.all;


entity RLink is port
        (
        RCLK            :       in std_logic;
        ACK             :       in std_logic;
        nRPAFE          :       in std_logic;
        nREF            :       in std_logic;
        nRHF            :       in std_logic;
        nRST            :       in std_logic;
        RD              :       in std_logic_vector (7 downto 0);
        nRDY            :       in std_logic;
        RVIOL           :       in std_logic;
        RSCnD           :       in std_logic;
        CMD_LINES       :       out std_logic_vector (3 downto 0);
        nRENW           :       out std_logic;
        TEST_PASSED     :       out std_logic;
        nTEST           :       out std_logic;
        LSTAT           :       out std_logic;
        REF             :       out std_logic
        );

end RLink;

architecture Behavioral of RLink is

subtype StateType is std_logic_vector (6 downto 0);

signal HALT_FLAG        :       std_logic:= '0';
signal OK_FLAG          :       std_logic:= '0';
signal TST_PASSED       :       std_logic:= '0';
signal L_FLAG           :       std_logic;
signal STATE            :       StateType;

constant RESET          :       StateType := "0011111"; --1F
constant TEST           :       StateType := "0001111"; --0F
constant IDLE           :       StateType := "0010000"; --10
constant NEED_HALT      :       StateType := "0010001"; --11
constant NEED_ENA       :       StateType := "0010010"; --12
constant LINK_OK        :       StateType := "0110000"; --30
constant HALT_XMT       :       StateType := "0010011"; --13
constant RESUME_XMT     :       StateType := "0010100"; --14
constant GOT_K280       :       StateType := "1110000"; --70

begin
```

```
test_count:PROCESS(RCLK, nRST)
        variable tst_cnt            :       integer range 0 to 7;
        begin
                if (nRST = '0') then
                        tst_cnt := 0;
                        TST_PASSED <= '0';
                elsif (RCLK'event and RCLK = '1') then
                        if (nRDY = '1') then
                                tst_cnt := tst_cnt + 1;
                                if (tst_cnt = 7) then
                                        TST_PASSED <= '1';
                                end if;
                        end if;
                end if;
end process test_count;

RcvCntl: process (RCLK, nRST)

begin

if (nRST = '0') then
        L_FLAG <= '1';
        STATE <= RESET;

else
        if (RCLK'event and RCLK = '1') then

                case STATE is

                        when RESET =>
                                STATE <= TEST;


                        when TEST =>
                                if (TST_PASSED = '0') then
                                        STATE <= TEST;
                                else
                                        STATE <= IDLE;
                                end if;

                        when IDLE =>

                                if ((nREF = '0') and (nRHF = '0') and
                                        (nRPAFE = '0') and (HALT_FLAG = '0')) then
                                                STATE <= NEED_HALT;

                                elsif ((nREF = '1') and (nRHF = '1') and
                                        (nRPAFE = '1') and (HALT_FLAG = '1')) then
                                                STATE <= NEED_ENA;

                                elsif ((nRDY = '0') and (RVIOL = '0')
                                         and (RSCnD = '1')) then
```

```
                    if (RD = K280) then
                        STATE <= GOT_K280;
                    end if;

                else
                    STATE <= IDLE;

                end if;


when NEED_HALT =>

        HALT_FLAG <= '1';

        if (ACK = '0') then
            STATE <= IDLE;
        else
            STATE <= NEED_HALT;
        end if;

when NEED_ENA =>

        HALT_FLAG <= '0';

        if (ACK = '0') then
            STATE <= IDLE;
        else
            STATE <= NEED_ENA;
        end if;

when GOT_K280 =>

        if (RD = K280) then
            STATE <= LINK_OK;
        elsif (RD = K281) then
            STATE <= HALT_XMT;
        elsif (RD = K282) then
            STATE <= RESUME_XMT;
        else
            STATE <= IDLE;
        end if;

when LINK_OK =>
        L_FLAG <= '0';
        if (OK_FLAG = '0') then
            OK_FLAG <= '1';
        else
            OK_FLAG <= '0';
        end if;

        STATE <= IDLE;
```

```
                when HALT_XMT =>

                        if (ACK = '1') then
                                STATE <= IDLE;
                        else
                                STATE <= HALT_XMT;
                        end if;

                when RESUME_XMT =>

                        if (ACK = '0') then
                                STATE <= IDLE;
                        else
                                STATE <= RESUME_XMT;
                        end if;

                when OTHERS =>
                        STATE <= RESET;

                end case;

        end if;
end if;

end process RcvCntl;

Output: process(STATE)

begin
        CMD_LINES(3 downto 0)       <=      STATE(3 downto 0);
        nTEST                       <=      STATE(4);
        TEST_PASSED                 <=      TST_PASSED;
        LSTAT                       <=      OK_FLAG;
        nRENW                       <=      L_FLAG or RSCnD;
        REF                         <=      RVIOL;
end process Output;

end Behavioral;
```

Transmitter State Machine

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

library work;
use work.codedefs.all;



entity TxLnk is port
        (
        CLK26                   :       in std_logic;
        nRP                     :       in std_logic;
        CMD_FLAGS               :       in std_logic_vector (3 downto 0);
        ACK                     :       out     std_logic;
        TEST_PASSED             :       in std_logic;
        nTEF                    :       in std_logic;
        nTHF                    :       in std_logic;
        nRST                    :       in std_logic;
        nRESET                  :       out std_logic;
        nTENA                   :       out std_logic;
        nTENR                   :       out std_logic;
        TXSCnD                  :       out std_logic;
        nTOE                    :       out std_logic;
        TQ                      :       out std_logic_vector (7 downto 0)
        );

end TxLnk;

architecture behavioral of TxLnk is

subtype StateType is std_logic_vector(9 downto 0);
signal DOUT             :       std_logic_vector(1 downto 0);
signal STATE            :       StateType;
signal HALT_FLAG        :       std_logic;
signal COUNT_FLAG       :       std_logic := '1';
signal ACK_FLAG         :       std_logic := '0';

constant RESET          :       StateType := "0011100010"; -- 0E2
constant TEST           :       StateType := "0010100011"; -- 0A3

constant IDLE           :       StateType := "0111100011"; -- 1E3

constant PREP_DATA      :       StateType := "0001000001"; -- 041
constant SEND_DATA      :       StateType := "0000100001"; -- 021

constant LSTAT0         :       StateType := "0010101011"; -- 0AB
constant LSTAT1         :       StateType := "0110101011"; -- 1AB

constant HALT0          :       StateType := "0011100111"; -- 0E7
constant HALT1          :       StateType := "1011100011"; -- 2E3
```

```
constant SEND_HALT0          :      StateType := "0010101111"; -- 0AF
constant SEND_HALT1          :      StateType := "0010110111"; -- 0B7
constant SEND_HALT2          :      StateType := "0011100011"; -- 0E3

constant SEND_RESUME0        :      StateType := "0110101111"; -- 1AF
constant SEND_RESUME1        :      StateType := "0010111111"; -- 0BF
constant SEND_RESUME2        :      StateType := "1111100011"; -- 3E3


begin

status_count:PROCESS(CLK26, nRST)
        variable lnk_cnt        :       integer range 0 to 1023;
        begin
                if (nRST = '0') then
                        lnk_cnt := 0;
                elsif (clk26'event and clk26 = '1') then
                        if (COUNT_FLAG = '1') then
                                lnk_cnt := lnk_cnt + 1;
                                        if (lnk_cnt = 0) then
                                                COUNT_FLAG <='0';
                                        end if;
                        elsif (ACK_FLAG = '1' and COUNT_FLAG <= '0') then
                                        COUNT_FLAG <= '1';
                        end if;
                end if;
end process status_count;

xmtctl: process (CLK26, nRST)

        begin

        if (nRST = '0') then
                STATE <= RESET;

        else
          if (CLK26'event and (CLK26 = '1')) then
                 case STATE is

                        when RESET =>           -- Reset STATE
                                STATE <= TEST;

                        when TEST =>
                                if ((TEST_PASSED = '1') and
                                   (CMD_FLAGS = BUS_IDLE)) then
                                        STATE <= LSTAT0;  -- Enable receiver on other side of link.
                                else
                                        STATE <= TEST;
                                end if;
```

```
when IDLE =>                                --IDLE STATE
        HALT_FLAG <= '0';
        ACK_FLAG  <= '0';
        if (COUNT_FLAG = '0') then
                STATE <= LSTAT0;    -- Transmit LSTAT
        elsif NOT (nTEF = '0' and nTHF = '1') then
                STATE <= PREP_DATA;
        elsif (CMD_FLAGS = HALT_TRANSMITTER) then
                STATE <= HALT0;
        elsif (CMD_FLAGS = SEND_HALT) then
                STATE <= SEND_HALT0;
        elsif (CMD_FLAGS = SEND_ENABLE) then
                STATE <= SEND_RESUME0;
        else
                STATE <= IDLE;
        end if;

when PREP_DATA =>           -- Allow for data from FIFO to settle.
        STATE <= SEND_DATA;

when SEND_DATA =>           -- Transmit data.
        STATE <= IDLE;

when LSTAT0 =>
        ACK_FLAG <= '1';
        STATE <= LSTAT1;

WHEN LSTAT1 =>
if (HALT_FLAG = '1') then
        STATE <= HALT0;
else
        STATE <= IDLE;
end if;

when HALT0 =>                                   -- Halt State.
        HALT_FLAG <= '1';
        if (CMD_FLAGS = SEND_HALT) then
                STATE <= SEND_HALT0;
        elsif (CMD_FLAGS = SEND_ENABLE) then
                STATE <= SEND_RESUME0;
        elsif (COUNT_FLAG = '0') then
                STATE <= LSTAT0;                -- Transmit LSTAT.
        elsif (CMD_FLAGS /= ENABLE) then
                STATE <= HALT0;
        else
                STATE <= HALT1;
        end if;

when HALT1 =>
        STATE <= IDLE;

when SEND_HALT0 =>
```

```
                    STATE <= SEND_HALT1;

                when SEND_HALT1 =>
                    STATE <= SEND_HALT2;

                when SEND_HALT2 =>
                    if ((CMD_FLAGS = BUS_IDLE) and (HALT_FLAG = '1')) then
                            STATE <= HALT0;
                    elsif ((CMD_FLAGS = BUS_IDLE) and (HALT_FLAG = '0')) then
                            STATE <= IDLE;
                    else
                        STATE <= SEND_HALT2;
                    end if;

                when SEND_RESUME0 =>
                    STATE <= SEND_RESUME1;

                when SEND_RESUME1 =>
                    STATE <= SEND_RESUME2;

                when SEND_RESUME2 =>
                    if ((CMD_FLAGS = BUS_IDLE) and (HALT_FLAG = '1')) then
                            STATE <= HALT0;
                    elsif ((CMD_FLAGS = BUS_IDLE) and (HALT_FLAG = '0')) then
                            STATE <= IDLE;
                    else
                            STATE <= SEND_RESUME2;
                    end if;

                when OTHERS =>
                    STATE <= RESET;

                end case;
        end if;
 end if;

end process xmtctl;

output: process(STATE)
begin
        nRESET              <= STATE(0);
        TXSCnD              <= STATE(1);
        nTOE                <= STATE(1);
        ACK                 <= STATE(2);
        DOUT(1 downto 0)    <= STATE(4 downto 3);
        nTENR               <= STATE(5);
        nTENA               <= STATE(6);



end process;
```

```
with DOUT select
        TQ(7 downto 0) <= HI_Z when "00",
                                        K280 when "01",
                                        K281 when "10",
                                        K282 when "11",
                                        HI_Z when others;
end behavioral;
```

'C4x Communications State Machine

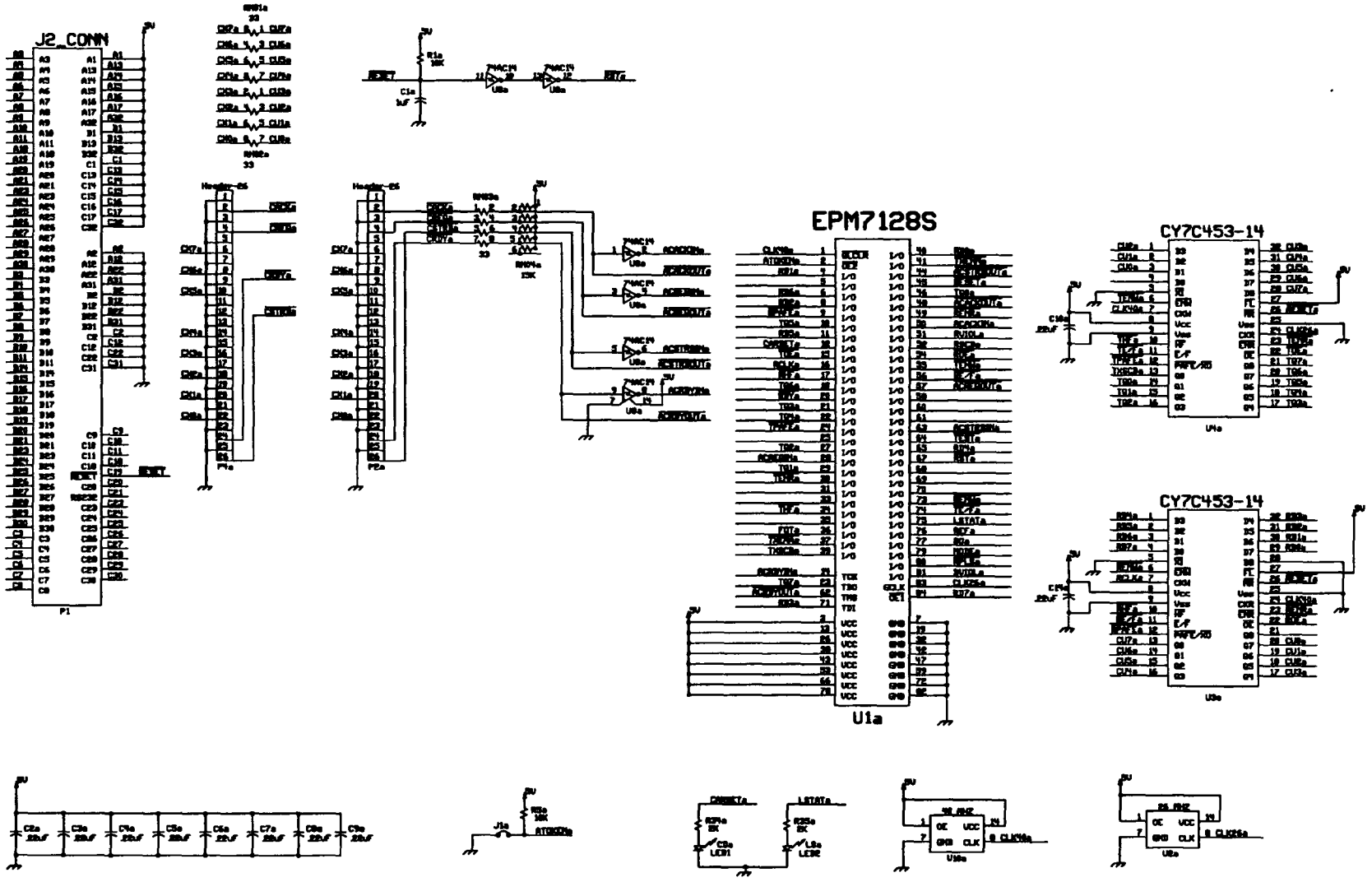
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY protosm is
    PORT
    (
        CLK40           : in std_logic;
        nRST            : in std_logic;
        ATOKEN          : in std_logic;
        nTPAFE          : in std_logic;
        nREF            : in std_logic;
        nRHF            : in std_logic;
        nTEF            : in std_logic;
        nTHF            : in std_logic;
        nROE            : out std_logic;
        ACREQIN         : in std_logic;
        nACREQOUT       : out std_logic;
        ACACKIN         : in std_logic;
        CREQIN          : in std_logic;
        nACACKOUT       : out std_logic;
        ACRDYIN         : in std_logic;
        nACRDYOUT       : out std_logic;
        ACSTRBIN        : in std_logic;
        nACSTRBOUT      : out std_logic;
        nTENW           : out std_logic;
        oe              : out std_logic;
        nRENR           : out std_logic
    );

end protosm;

ARCHITECTURE behavioral OF protosm is

subtype StateType is std_logic_vector(11 DOWNTO 0);

signal CNTEN        : std_logic;
signal STATE        : StateType;
signal chg_flag     : std_logic;
signal C            : integer range 0 to 3;

CONSTANT RESET              :   StateType := "000100000000"; -- 100
CONSTANT GOT_TOKEN          :   StateType := "111100000000"; -- F00
CONSTANT TOKEN_RELEASE      :   StateType := "000101000000"; -- 140
CONSTANT DELAY1             :   StateType := "100101000000"; -- 940
CONSTANT DELAY2             :   StateType := "110101000000"; -- D40
CONSTANT SWITCH_BUS         :   StateType := "010101000000"; -- 640

CONSTANT OUTPUT_ENABLE      :   StateType := "000110000010"; -- 182
CONSTANT DATA_OUT           :   StateType := "000110010001"; -- 191
CONSTANT SENT_DATA_VALID    :   StateType := "000110010000"; -- 190

CONSTANT INPUT_ENABLE       :   StateType := "000000000000"; -- 000
```

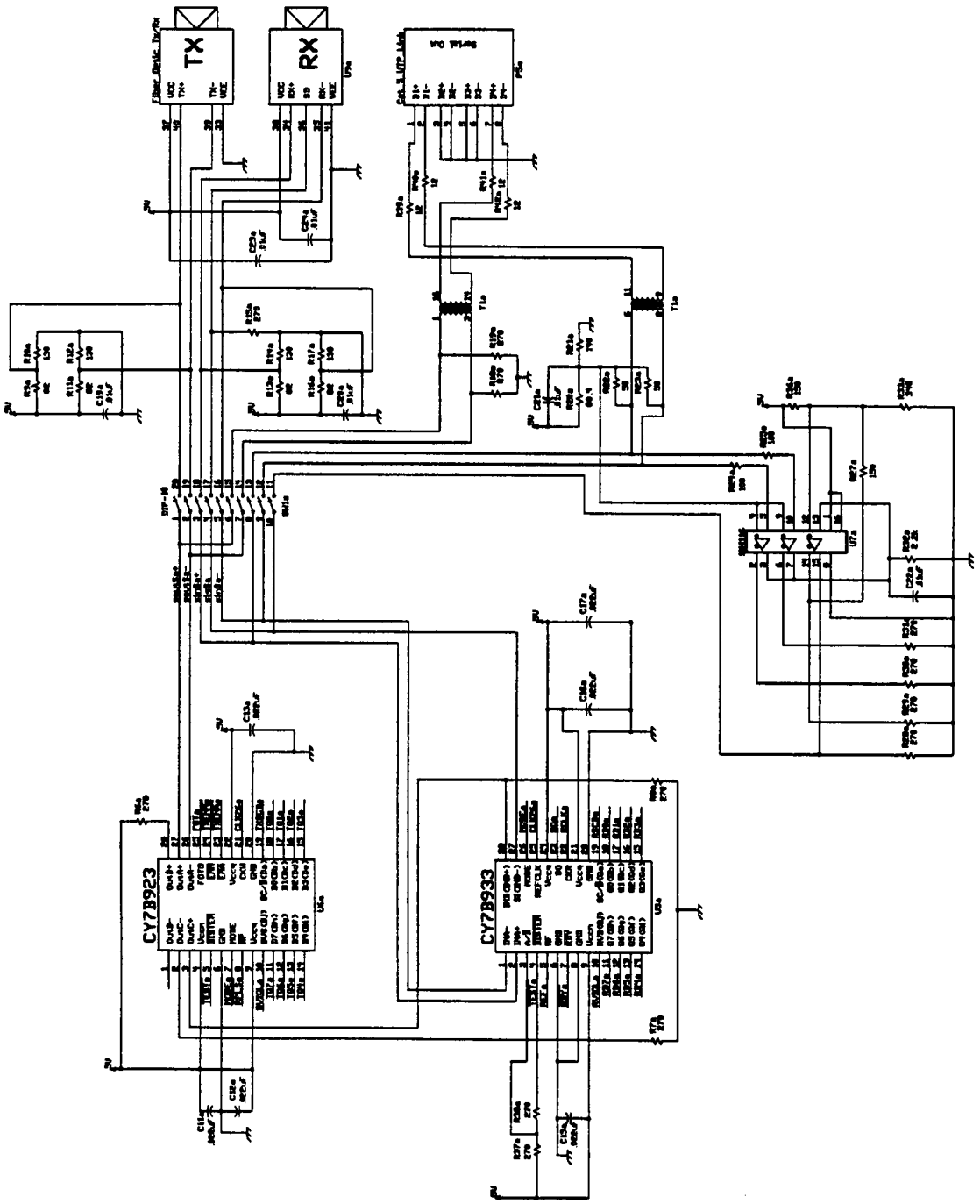
```
CONSTANT READ_DATA              :       StateType := "000000000101"; -- 005
CONSTANT RECV_DATA_VALID        :       StateType := "000000001000"; -- 080

CONSTANT NO_TOKEN               :       StateType := "001000000000"; -- 200
CONSTANT TOKEN_REQUEST          :       StateType := "000000100000"; -- 020
CONSTANT CREQ_HIGH              :       StateType := "111000000000"; -- E00
CONSTANT TURN_BUS               :       StateType := "010100000000"; -- 500

begin
count:process(CLK40, nRST)
        begin
                if (nRST = '0') then
                        C <= 0;
                elsif (clk40'event and (clk40 = '1')) then
                        if (CNTEN = '1') then
                                C <= C+1;
                        end if;
                end if;
end process count;

c4xlnk:process(CLK40, nRST)
 begin
  if nRST='0' then
        chg_flag <= '1';
        STATE <= RESET;
  elsif (CLK40'EVENT AND CLK40 = '1') then
        case STATE is
                        when RESET =>
                                if(ATOKEN = '1') then
                                        STATE <= NO_TOKEN;
                                else
                                        STATE <= GOT_TOKEN;
                                end if;

                        when NO_TOKEN =>
                                if ((ACSTRBIN = '1') and NOT(nTHF = '0' and nTEF = '0' and
                                   nTPAFE = '0')) then
                                        STATE <= INPUT_ENABLE;
                                elsif (NOT (nREF = '0' and nRHF = '1') and C = 0
                                   and chg_flag = '1') then
                                        STATE <= TOKEN_REQUEST;
                                else
                                        STATE <= NO_TOKEN;
                                end if;

                        when TOKEN_REQUEST =>
                                if (ACACKIN = '1') then
                                        STATE <= CREQ_HIGH;
                                else
                                        STATE <= TOKEN_REQUEST;
                                end if;
```

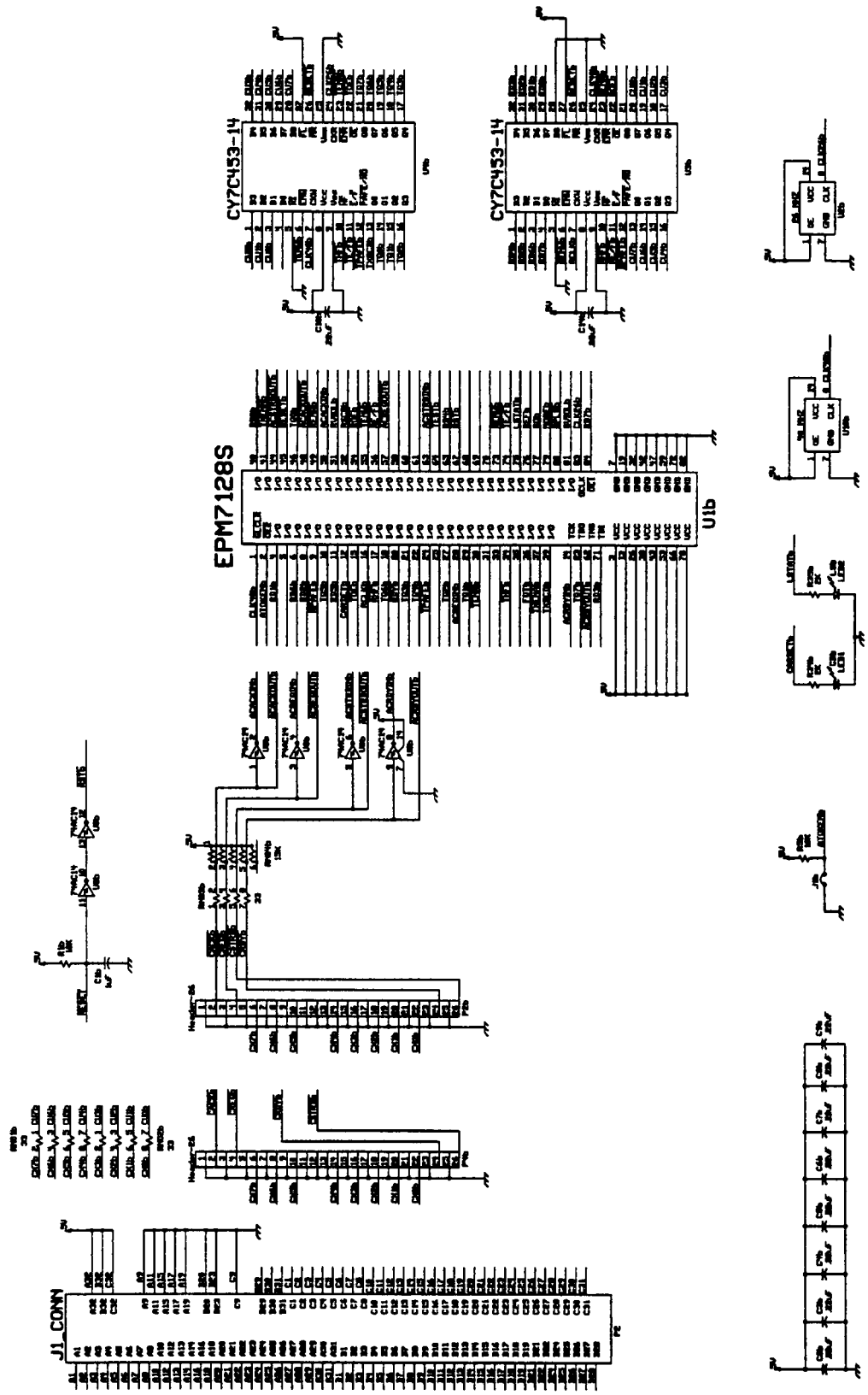
```
when CREQ_HIGH =>
        STATE <= TURN_BUS;

when TURN_BUS =>      -- Token received. Ready to transfer 1 word;
        STATE <= OUTPUT_ENABLE;

when OUTPUT_ENABLE =>
        STATE <= DATA_OUT;

when DATA_OUT =>
        STATE <= SENT_DATA_VALID;

when SENT_DATA_VALID =>
        if (ACRDYIN = '1') then
                STATE <= GOT_TOKEN;
        else
                STATE <= SENT_DATA_VALID;
        end if;

when GOT_TOKEN =>
        if (ACREQIN = '1' and C = 0) then
                STATE <= TOKEN_RELEASE;
        elsif (NOT (nREF = '0' and nRHF = '1') and ACRDYIN = '0') then
                STATE <= OUTPUT_ENABLE;
        else
                STATE <= GOT_TOKEN;
        end if;

when TOKEN_RELEASE =>
        chg_flag <= '0';
        STATE <= DELAY1;

when DELAY1 =>
        STATE <= DELAY2;

when DELAY2 =>
        STATE <= SWITCH_BUS;

when SWITCH_BUS =>
        STATE <= NO_TOKEN;

when INPUT_ENABLE =>
        chg_flag <= '1';
        STATE <= READ_DATA;

when READ_DATA =>
                STATE <= RECV_DATA_VALID;

when RECV_DATA_VALID =>
        if (acstrbin = '0') then
                STATE <= NO_TOKEN;
        else
```

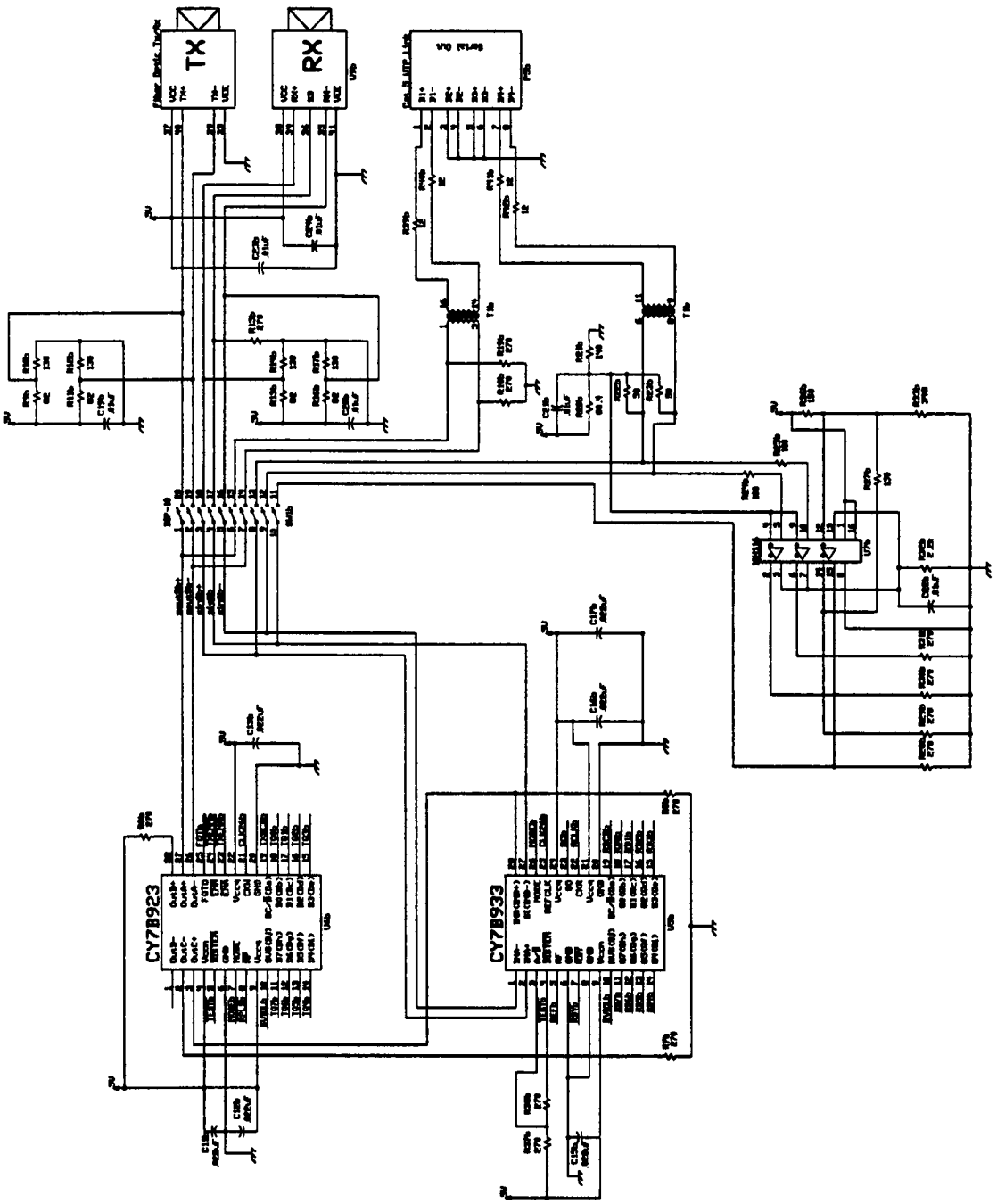
```
                                        STATE <= RECV_DATA_VALID;
                                end if;

                        when others =>
                                STATE <= RESET;

                        end case;
                end if;
end process c4xlnk;

output:process(STATE)

        begin
                nROE            <= NOT STATE(7);
                nTENW           <= NOT STATE(2);
                nRENR           <= NOT STATE(1);
                CNTEN           <= STATE(0);
                oe              <= STATE(8);

                if (STATE(8) = '0') then
                        nACRDYOUT  <= NOT STATE(3);
                        nACREQOUT  <= NOT STATE(5);
                        nACACKOUT  <= 'Z';
                        nACSTRBOUT <= 'Z';
                else
                        nACACKOUT   <= NOT STATE(6) or NOT CREQIN;
                        nACSTRBOUT  <= NOT STATE(4);
                        nACRDYOUT   <= 'Z';
                        nACREQOUT   <= 'Z';

                end if;

        end process output;
end behavioral;
```

Schematics


J2_CONN
P1
EPM7128S
U1a
CY7C453-14
CY7C453-14


TX
RX
CY7B923
CY7B933


CY7C453-14
CY7C453-14
EPM7128S
U1b
J1_CONN


TX
RX
CY7B923
CY7B933

Vita

Robert Earl Leugers was born in Hamilton, Ohio on December 12, 1962. He attended some school in Hamilton, but his family moved to Myrtle Beach, South Carolina in 1972. He graduated from Conway High School, Conway, South Carolina in 1981.

He enlisted in the United States Air Force in 1982 after an aborted attempt to enter college as a music major. Having served for five years, he took his first college courses part time as a business major. After several classes he changed his major to electrical engineering.

He quit the Air Force after serving for nine years to attend The University of Texas at San Antonio to finish his Bachelor of Science degree in Electrical Engineering. Two years before completing the degree, the USAF hired him again, into the PALACE Acquire Cooperative Education Program. The time in this program was spent working for a software engineering section at Kelly Air Force Base, Texas. He was responsible for designing an air data computer simulation to communicate on the Mil-Std-1553B data bus. He graduated with a degree in Electrical Engineering in December of 1994.

The USAF then made him a PALACE Acquire Intern and transferred him to Tinker AFB, Oklahoma City, Oklahoma in May 1995 to work for a Local Area Network design section. August 1995 he was sent to The University of Tennessee Space Institute for a year of graduate studies in Electrical Engineering. He will return to Oklahoma City to design local area networks for the U. S. Air Force and Department of Defense. He has

completed the requirements for a Masters of Science Degree in Electrical Engineering which will be conferred in December 1996.