A LAB ORIENTED PROJECT

ON

DC BALANCED, PARTITIONED-BLOCK, 8B/10B TRANSMISSION CODE IMPLEMENTATION USING PSOC

Prepared under the supervision of

Mr A Amalin Prince

(Electrical and Electronics Engineering Group)

By

Aalap Tripathy 2004P34PS208

For fulfillment of the requirements for

Lab Oriented Project (BITS GC 313)

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE - PILANI,

GOA CAMPUS

ZUARI NAGAR, GOA, INDIA

Abstract

This code is well suited for high speed local area networks and similar data links where the information format consists of packets variable in length from a few to several hundred 8-bit bytes. This code translates each source byte into a constrained 10-bit binary sequence which has excellent performance parameters (run length=5, maximum digital sum variation=6, maximum burst length=5). This is implemented in PSoC by portioning the coder into 5B/6B and 3B/4B subordinate coders.

The project also has enabled UART data from the M8C CPU to be broadcast after the 8B/10B coding as Transmission Control Protocol / Internet Protocol (TCP/IP) Packets using a EAD Controller. Currently an independent EAD Controller (Sparr Electronics) is used. However, the PSoC possesses capabilities to be itself programmed as an Ethernet Adaptor Controller. This capability is yet unexplored.

Key Words

Programmable System on Chip, Line Coding, Power Spectral Density, PSoC Designer, UART, TCP/IP, OSI Layers

Table of Contents

1. Acknowledgements

No part of this work would have been possible without the active support and guidance of Mr A Amalin Prince. His constant support and encouragement let me take up the challenge of a new and hitherto forgotten transmission code for Ethernet.

I sincerely thank the encouragement by Mr M T Abhilash, Lab-in-charge, PSoC Lab, BITS Pilani Goa Campus who has allowed students like us access to lab facilities often at odd hours.

Since, this project has been a culmination of my learning at BITS Pilani Goa Campus over four years, I would like to thank all my instructors especially Mrs Anita Agrawal, Mr Nitin Sharma, Mr A Khadke whose approach to problem solving has taught me many lessons some of which I have tried to incorporate in this work.

Of course, none of this would ever have been possible without the support from Cypress Semiconductors, San Jose. Mr Ashish Garg, Strategic Marketing Engineer, Mr Kaushik Subhramaniam Narayanan have been our companions through this journey. Equally significant have been the support of Mr Kamal Gunsagar, Vice President, Business Development, Cypress Semiconductor, Mr Patrick Kane, Director, Cypress University Alliance, Mr Jeff Dahlin, Principal Application Engineer, Mr Dave van Ess, Principal Application Engineer and Chief of Technical Staff, Mr Ganesh Raja, the PSoC Master who have all regularly reviewed the work done and have made suggestions.

Though this project is complete in itself, I would hope future readers of this report pick up this application and deploy the versatility of PSoC and the ease of data transmission using this new transmission code.

Aalap Tripathy
2004P3PS208

2. Introduction

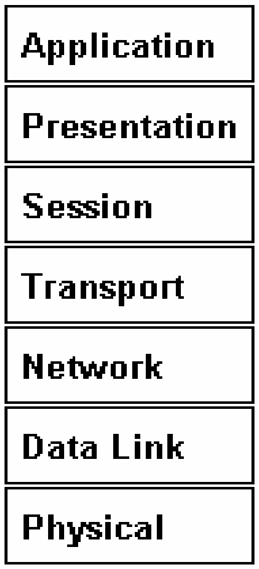
Ethernet Applications for a Microprocessor has tremendous implication for internetworking devices. Though a vast multitude of such devices exist today, there are currently no known applications for Ethernet using Cypress' Programmable System on Chip. Whereas some critics point to the lack of speed of the M8C CPU on the PSoC to successfully deploy IEEE 802.3 Ethernet Standard, this project has been successfully able to deploy a 9 bit transmission code using the UART (Tx and Rx) module in both duplex and half duplex modes.

Further, by using a Ethernet Controller, the UART data was also observed using Ethereal Packet Capture utilities. Therefore, despite speed limitations, we might still be able send Ethernet Packets though the effective transmission rate is limited to Micorcontroller processing speed.

The ideal result for this project was implementation of a DC balanced (no residual DC due to continuous pull ups and pull downs) transmission code. However, it was later explored that such an implementation would be impossible because of PSoC's speed limitations. It was also learnt that Engineers at Cypress are also aiming to deploy onboard native Ethernet Support in the next generation of PSoC Chips.

This project aims to improve data transmission efficiency and bring about DC balance by sending 10 bits for every 8 bits to be transmitted. This is based on the results of Widmer and Franaszek (paper attached in Appendix). Their theoretical results are to be tested using a real world application. When their original paper was published, the Ethernet standard had not been developed and deployed. It is therefore important to explore the relevancy of their results for applications today.

3. OSI Application Layer Model



- OSI – Open System Interconnection
- Follows a Layered Approach
- Allows better interoperability between software and hardware
- Allows design of elaborate but highly reliable protocol stacks
- Protocol stacks can be implemented either in hardware or software, or a mixture of both

This project essentially operates at the Transport Layer of the OSI Stack. We are trying to enhance the functionality of Transmission Control Protocol/ User Datagram Protocol as defined in Ethernet (IEEE 802.3) Standard

The Transport Layer

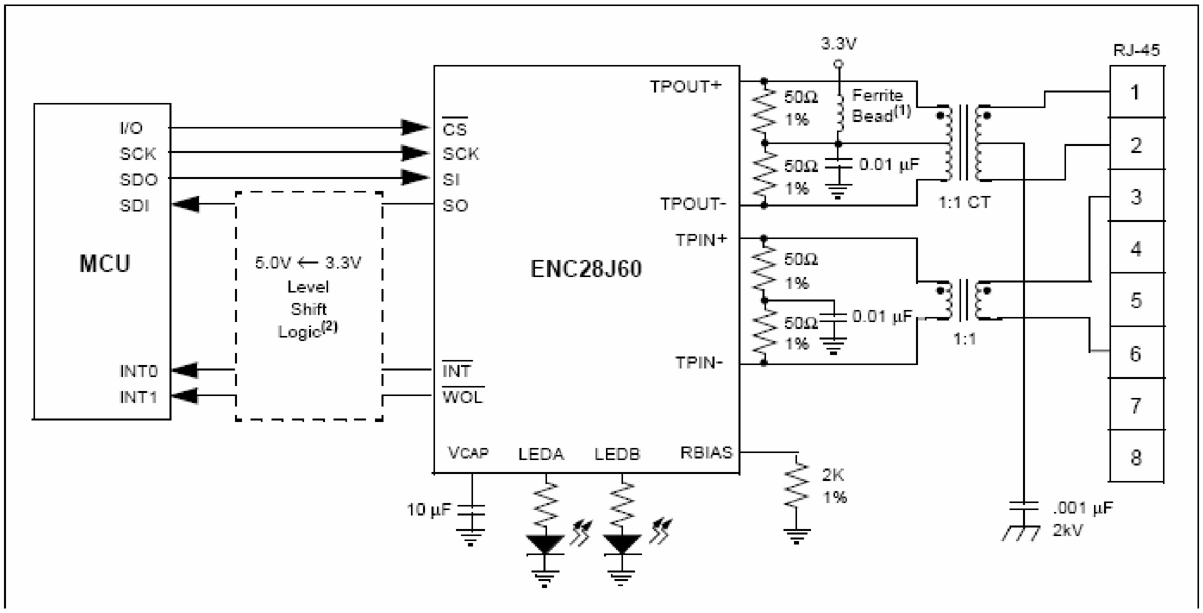
- Provides transparent transfer of data between end users
- Controls reliability of a given link
- Some protocols are state full and connection oriented (cookies)
- Example – TCP , UDP

Further, since we are operating at a hardware level, we might also consider this project to be implementing this modification at the Physical layer. That is we are using the PSoC Microcontroller itself to generate the 10 bit transmission code based on the logic that has already been developed by Widmer and Franaszek in IBM JOURNAL FOR RESEARCH & DEVELOPMENT. Volume 27 No. 5 September 1983

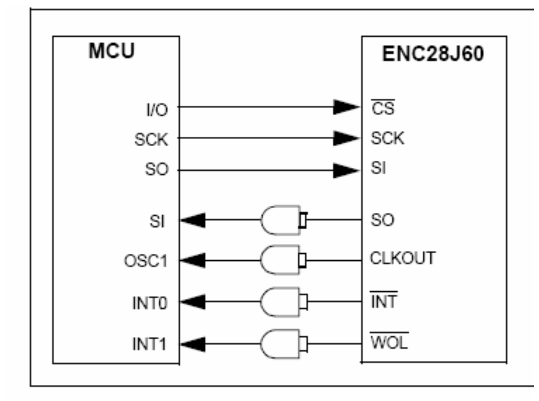
The Physical Layer

- Defines all electrical and physical specifications for connecting devices
- Major Functions -
    - Establishment & Termination of Connections
    - Connection Resolution & Flow Control between Communicating Resources
    - Modulation & Conversion between Digital Data
- Example – Radio, SCSI (Small Computer System Interface)

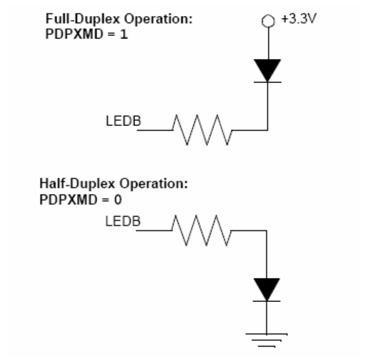
4. Application of ENC28J60



- This is a single Ethernet Interface Chip which needs a differential power transformer and SPI enabled microcontroller to function.
- On the differential transmit pins (TPOUT+/TPOUT-), a 1:1 pulse transformer with a center tap is required.
- The transformers should be rated for isolation of 2 kV or more to protect against static voltages.
- Both portions additionally require two 50Ω, 1% resistors and a 0.01 µF capacitor for proper termination.
- A 2.5V regulator is incorporated internally to generate the necessary voltage.
- However, a 10 µF capacitor for stability purposes from VCAP to GND
- All power supply pins must be externally connected to the same 3.3V power source.
- Similarly, all ground references should be externally connected to the same ground node.
- Each VDD and VSS pin pair should have a 0.1 µF ceramic bypass capacitor placed as close to the pins as possible. (Not Shown)
- Relatively high currents are necessary to operate the twisted pair interface, so all wires should be kept as short as possible and reasonable wire widths should be used on power wires to reduce resistive loss.
- The internal analog circuitry in the ENC28J60 requires that an external 2 kΩ, 1% resistor be attached from RBIAS to ground.



- Level Shifting Using AND Gates (74HCT08 Quad AND)
- MCU (PSoC) Works at 5V and the Pins of ENC28J60 work at 3.3V



- Upon system Reset, the ENC28J60 will detect how the LED is connected and begin driving the LED to the default state

5. About Ethernet Adaptor (EAD) - Overview

EAD is a Device that converts RS-232 protocol into TCP/IP protocol. It enables remote gauging, managing and control of a Serial Device through the network based on Ethernet and TCP/IP by connecting to the existing equipment with RS-232 serial interface. In other words, EAD is a protocol converter that transmits the data sent by Serial equipment as TCP/IP data type and converts back the TCP/IP data received through the network into serial data to transmit back to the equipment. EAD also supports UDP Protocol for broadcast kind of application. The EAD allows you to network-enable a variety of serial devices that were not originally designed to be networked. This capability brings the advantages of remote management and data accessibility to thousands of serial devices over the network.

The EAD is the most cost effective Single port Serial-Ethernet communication device. The EAD supports RS232 serial communication allowing virtually any asynchronous serial device to be accessed over a network. Some models do Support RS 485 and RS422 as add-on feature. As for the Internet connectivity, the EAD supports open network protocols such as TCP/IP allowing Serial Devices to be accessed over broadband network or conventional LAN (Local Area Network) environment.

The EAD provides the management console using Telnet and Serial console port under the password protection support. The EAD was designed to accommodate the unique requirements of the Retail POS, Security, Automation and Medical marketplaces. The EAD uses IP protocol for network communications. For network connections to the serial port TCP, UDP and Telnet protocols are used.

The Ethernet adaptors have what is known as Hardware address or MAC address. It has its own addressing scheme based on a unique six-byte address. This is generally called Media Access and Control (MAC) address. One example of Ethernet Address is given below: 00-50-C2-18-E0-00 or 00:50:C2:18:E0:00.

To identify an individual computer/device on the IP network, the device must have a unique IP address in a Network. The current version of Internet Protocol uses a four-byte number, expressed in dotted decimal notation. Sample IP Address 192.168.0.250

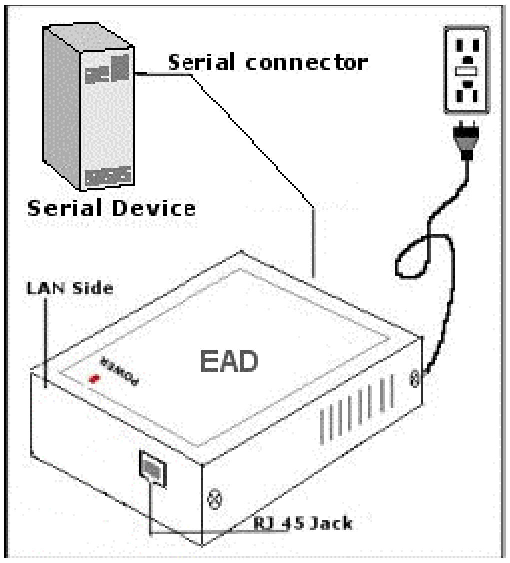
Every TCP connection is established using a destination IP address and a Port number. For example Telnet application commonly uses port number 23 of contacted IP number. The EAD's Serial channel (port) can be associated with a specific TCP or UDP Port number.

Serial Interface

The EAD has a 9- pin RS 232 Male Serial connector in the Metal Box (MB) unit or RJ45 male in Plastic Box Model or with Open Wire / with connectors for PCB Board Level product (depending on the model), which can be connected to any Serial device.

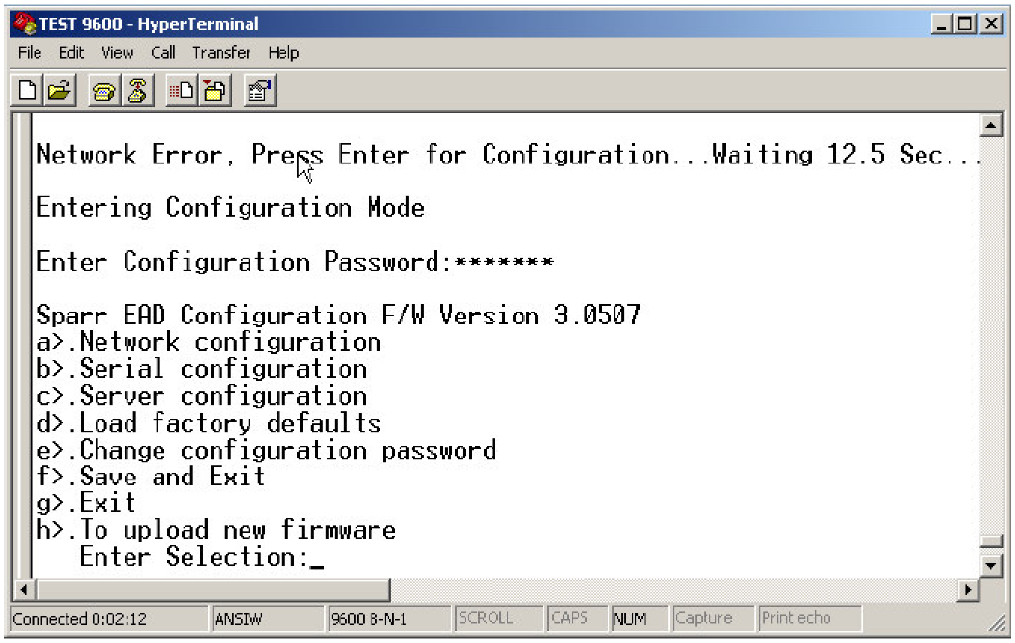
Network Interface

EAD has one RJ45 Female 10 Base-T Ethernet port that supports up to 10 Mbps speed for connection to Local Area Network (LAN) through Hub or Switch. You can also use a Cross Cable to connect to PC's LAN Card directly.

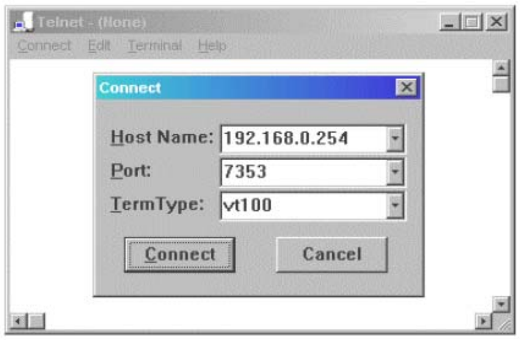


Configuration of the EAD adapter

In order to work as desired the adapter needs to be configured as per the needs and the requirements of the application. The configuration requires the setting of various parameters related to serial communication like the baud rate etc and also the the various parameters related to LAN like the IP address, subnet etc. The adapter may be configured through the HyperTerminal or Telnet

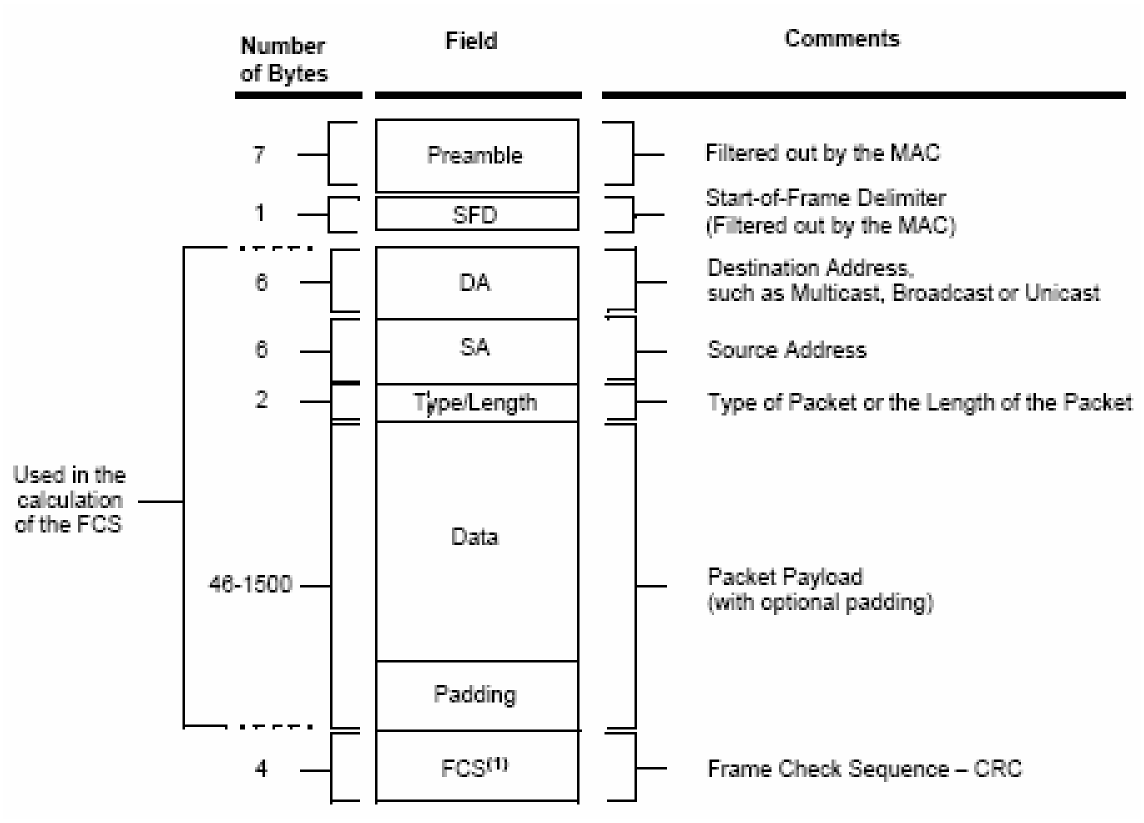


```
Network Error, Press Enter for Configuration...Waiting 12.5 Sec..

Entering Configuration Mode

Enter Configuration Password:*******

Sparr EAD Configuration F/W Version 3.0507
a>.Network configuration
b>.Serial configuration
c>.Server configuration
d>.Load factory defaults
e>.Change configuration password
f>.Save and Exit
g>.Exit
h>.To upload new firmware
   Enter Selection:_
```



Host Name: 192.168.0.254
Port: 7353
TermType: vt100

A standard (IEEE 802.3) Frame


Number of Bytes
Field
Comments
7
Preamble
Filtered out by the MAC
1
SFD
Start-of-Frame Delimiter (Filtered out by the MAC)
6
DA
Destination Address, such as Multicast, Broadcast or Unicast
6
SA
Source Address
2
Type/Length
Type of Packet or the Length of the Packet
Used in the calculation of the FCS
46-1500
Data
Padding
Packet Payload (with optional padding)
4
FCS(1)
Frame Check Sequence – CRC

- Ethernet frames are between 64 and 1518 bytes long.
- The data payload can vary from 46-1500 bytes. **Instead of simply placing data bits as such, this project transmits 10 bits for every 8 bits actually meant to be transmitted.**

6. Primary Code Sequence

```
//---------------------------------------------------
// C main line
//----------------------------------------------------
/*
In order to create this application I combined two UARTs so that I wouldn't
need to have any external wiring. The software for the transmission is
written
for UART1 and for the reception for UART2. In reality this would be for the
same UART.
*/


#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
#include "tx8_1.h"
#include "rx8_1.h"


#define NO_OF_CHARS 8
#define MODULE_ADDRESS 0x5A

unsigned char cPhase;
//co-operative multitasking variable
unsigned char cTx1Buffer[NO_OF_CHARS]=
{MODULE_ADDRESS,0x01,0x02,0x03,0x04,0x05,0x06,0x07};
//transmit buffer
//for thie example, bit 9 is only set for the address byte, which is the
//first byte of a fixed 8 byte sequence
unsigned char cRx1Buffer[NO_OF_CHARS];
//receive buffer
/*two buffers are used because receive and transmit
are in same chip and thus working simultaneously.
Normally there would only be one working in half duplex and so the
same buffer could be use for transmit and receive*/
unsigned char cTxPnt,cRxPnt;
//pointers in buffer
unsigned char cNextParity;
//even parity calculated for the next byte


unsigned char cTxPhase, cRxPhase;
//state variables for processes
unsigned char cBit9;
//decoded bit 9
unsigned char cPrimed;
//indicator that bit 9 has been seen and address matched
unsigned char cFlag;
//to indicate end of received bloc

void Transmit (void);
void Receive (void);
unsigned char ucharTxParity (unsigned char cPbyte);
```

```
void main()
{
    // Insert your main routine code here.
      TX8_1_EnableInt();

      RX8_1_Start(RX8_PARITY_EVEN);
      RX8_1_EnableInt();




      M8C_EnableGInt;
    while (1)
    {
          switch (cPhase)
            {
                  case 0:
                        Transmit();
                  break;
            case 1:
                  Receive();
                  break;
            default:
                  cPhase=0;
                  break;
      }
    }
}


void Transmit (void)
{//transmit data
      int i,j;
      switch (cTxPhase)
      {
            case 0:
                  //initiate transmission
                  //remember to disable receiver
                  if(ucharTxParity(cTx1Buffer[0]))
                  {//if parity is one, and this is the byte with
                  //bit 9=1 then we allow even parity
                        TX8_1_Start(TX8_PARITY_EVEN);
                  }
                  else {//if parity is zero, and this is the byte with
                  //bit 9=1 then we allow odd parity to toggle the result
                        TX8_1_Start(TX8_PARITY_ODD);

                  }

                  //in order to save time and memory prepare for the next
byte during interrupt
                  cTxPnt=1;
                  //point at next location for the next character
                  //as the current one is going now, before an interrupt
                  if(ucharTxParity(cTx1Buffer[cTxPnt]))
                  {//if parity is one, and this is the byte with
                  //bit 9=0 then we allow odd parity
                        cNextParity=(TX8_PARITY_ODD);
                  }
                  else {//if parity is zero, and this is the byte with
                  //bit 9=0 then we allow even parity to toggle the result
                        cNextParity=(TX8_PARITY_EVEN);

                  }
                  //this will be used in the interrupt
                  //done before actual transmission to guarantee
                  //transmitted character is not complete before the end of
the calculation
                  TX8_1_SendData(cTx1Buffer[0]);
                  //start the usart

                  cTxPhase++;
                  break;
            case 1:
            //waiting for end of transmission
                  /*if End of transmission
                           {
                              TX8_1_Start(TX8_PARITY_EVEN);
                              //enable receiver
                           }*/
                  break;
            default:
            //in real life wait at the end of the transmission,switch to
waiting
            //for reception
                  break;
      }
      cPhase++;
}

void Receive (void)
{//receive and process data
      switch (cRxPhase)
      {
            case 0:
                  cFlag=cRxPnt=cPrimed=0;
                  cRxPhase++;
                  break;
            case 1:
                  //wait for data reception completion
                  if (cFlag)
                  {
                        cRxPhase++;
                  }
                  break;
            default:
                  //interpret incoming data
                  break;
      }
      cPhase++;
}
```

```
unsigned char ucharTxParity (unsigned char cPbyte)
{
      unsigned char cI,cJ,cCount;
      cJ=cPbyte;
      cCount=0;
      for (cI=0;cI<8;cI++)
      {//counting the number of ones in the byte
            if (cJ & 0x1)
            {
                  cCount++;
            }
            cJ=cJ>>1;
      }
      return (cCount & 0x01);
      //lsb indicates odd or even i.e. the actual parity bit
}

#pragma interrupt_handler Serial_Rx
//corresponding ljmp in RX8_1.asm
void Serial_Rx (void)
{
      unsigned char cI;
      //as message comes in
      unsigned char cRxStatus;
      //the first read clears this, so it maust be maintained.


      //cehck for errors
      cRxStatus=bRX8_1_ReadRxStatus();
      if (cRxStatus & (RX8_RX_OVERRUN_ERROR |RX8_RX_FRAMING_ERROR))
      {
            //there is an error here-
            //first read the byte to clear the register
            cI=bRX8_1_ReadRxData();
            //reset the pointer
            cRxPnt=0;
            //restart pointer
            cPrimed=0;
            //clear flag indicating correct address
      }
      else {
      //now we use the parity error to determine if bit 9 is set or not
            cI=bRX8_1_ReadRxData();
            if (ucharTxParity(cI))
            {//here there is an odd parity expected
                  if (cRxStatus & (RX8_RX_PARITY_ERROR))
                  {//odd parity exepcted, not seen and triggers an error
                        cBit9=0;
                  }
                  else {//odd parity exepcted, seen and no error seen
                        cBit9=1;
                  }
            }
            else {
            //here there is an even parity expected
                  if (cRxStatus & (RX8_RX_PARITY_ERROR))
                  {//even parity exepcted, not seen and triggers an error
```

```
                        cBit9=1;
                  }
                  else {//even parity exepcted, seen and no error seen
                        cBit9=0;
                  }
            }

            if (cBit9)
            {//if bit 9 set- control word
                  //check if matches address
                  if (cI==MODULE_ADDRESS)
                  {
                        cRx1Buffer[0]=cI;
                        cRxPnt=1;
                        cPrimed=1;
                        //indicate that the address has been seen
                  }
                  else {
                        cPrimed=0;
                        //clear flag to indicate incorrect address
                        cRxPnt=0;
                  }
            }
            else {
                  //check if ready to accept data
                  if(cPrimed)
                  {
                        cRx1Buffer[cRxPnt]=cI;
                        cRxPnt++;
                        if (cRxPnt>=NO_OF_CHARS)
                        {
                              cFlag=1;
                              //indicate to background that block has been
received
                              cPrimed=0;
                        }
                  }
            }
      }

}

#pragma interrupt_handler Serial_Tx
//corresponding ljmp in TX8_1.asm

void Serial_Tx (void)
{
      int i,j;
      TX8_1_bReadTxStatus();
      //clear flag to allow next interrupt
      if (cTxPnt<NO_OF_CHARS)
      {
            TX8_1_Start(cNextParity);
            if(ucharTxParity(cTx1Buffer[cTxPnt+1]))
            {//if parity is one, and this is the byte with
            //bit 9=0 then we allow odd parity
                  cNextParity=(TX8_PARITY_ODD);
            }
            else {//if parity is zero, and this is the byte with
            //bit 9=0 then we allow even parity to toggle the result
                  cNextParity=(TX8_PARITY_EVEN);
            }

            //this will be used in the interrupt
            TX8_1_SendData(cTx1Buffer[cTxPnt]);
            cTxPnt++;
      }
      else {
            cTxPnt=0;
      }
}
```

7. References

1. Kagan, Aubrey. “Implement a Nine-Data-Bit UART on a PC”, EDN Design Ideas, June 6, 1998.
2. Cypress Application Note - AN 2269 - Implement 9-Bit Protocol on the PSoC™ UART
3. Y. Takasaki, M. Tanaka, N. Maeda, K. Yamashita, and K. Nagano, “Optical Pulse Formats for Fiber Optic Digital Communications,” IEEE Trans. Commun. COM-24, 404-413 (1976).
4. J. M. Griffiths, “Binary Code Suitable for Line Transmission,” Electron. Lett. ,79-81 (1969).
5. R. G. Kiwimagi, “Encoding/Decoding for Magnetic Record Storage Apparatus,” IBM Tech. Disclosure Bull. 18, 3147-3149 (1976).
6. A. X. Widmer and P. A. Franaszek, “Transmission Code for High-speed Fibre-Optic Data Networks,” Electron. Lett. 19,
7. P. A. Franaszek, “Sequence-State Coding for Digital Transmission,” BellSyst. Tech. J. 47, 143-157 (1968).
8. P. A. Franaszek, “Sequence-State Methods for Run-Length-Limited Coding,” IBM J. Res. Develop. 14,376-383 (1970).
9. A. M. Patel, “Zero-Modulation Encoding in Magnetic Recording,” IBM J. Res. Develop. 19,366-378 (1975).
10. Peter A. Franaszek, “A General Method for Channel Coding,” IBM J. Res. Develop. 24,638-641 (1980).
11. P. A. Franaszek, “Construction of Bounded Delay Codes for Discrete Noiseless Channels,” IBM J. Res. Develop. 26, 506-514 (1982).
12. B. Marcus, “Sofic Systems and Encoding Data on Magnetic Tape,” Preliminary Report, Notices, Amer. Math. SOC2.9 , 43(1982).
13. R. L. Adler, D. Coppersmith, and M. Hassner, “Algorithms for Sliding Block Codes,” IEEE Trans. Info. Theory IT-29, 5-22 (1983).
14. G. Nigel N. Martin, Glen G. Langdon, Jr., and Stephen J. P. Todd, “Arithmetic Codes for Constrained Channels,” IBM J. Res. Develop. 27,94-I06 (1983).
15. Ta-Mu Chien, “Upper Bound on the Efficiency of DC Constrained Codes,’’ Bell Syst. Tech. J. 49, 2261-2287 (1970).
16. J. J. Stiffler, “Theory of Synchronous Communications,” Prentice-Hall, Inc., Englewood Cliffs, NJ, 1971, pp. 368-372.
17. Cypress Semiconductors Application Note - AN2013 - UART Receiver Errata Workaround
18. Cypress Semiconductors Application Note - AN2081 – Originally Cypress Innovator Design Challenge Contest Entry 280
19. http://www.psocdeveloper.com
20. http://www.time-lines.com/
21. http://easypsoc.com/book/

6. Improvement & Future Work:

1. This project has been able to successfully interface the PSoC to IEEE 802.3 Ethernet Standard using an Ethernet Adaptor Module. However the sheer flexibility of the PSoC makes it possible to program the PSoC itself as a Ethernet Adaptor. This aspect needs to be explored.
2. Now that data transmission over Ethernet is possible, related applications can be developed – one example is – Mess Management System : Bar code readers can be interfaced to the PSoC. Everytime a successful identification is done, based on the time, a entry on the user table (which meal he is eating) can be updated. On board RAM on the PSoC will be unsuitable, so additional components may be interfaced. At the end of a week/month, the cumulative data can be broadcast to a central server or system for record keeping and bill generation.