```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <conio.h>
#include<time.h>
#include <stdint.h>
//#define COM_PORT "COM5"      // Change to your COM port
#define COM_PORT "COM17"      // Change to your COM port
#define BAUD_RATE CBR_115200  // Baud rate
#define TOTAL_BYTES 1800      // Total bytes to transmit
#define TOTAL_Frame 1800      // Total bytes to transmit
#define ODP_BYTES 128
#define BYTE_DELAY_MS 0.7      // Increased delay between bytes
#define MAX_RETRIES 3        // Number of retries for failed writes
#define RECEIVE_BUFFER_SIZE TOTAL_BYTES+ODP_BYTES
#define frame_length    86400  // SIMULATED DATA FL BUT OLD IS   86400
#define uint8_t unsigned char
#define uint16_t unsigned short
#define int64_t long long
#define uint64_t unsigned long long
#define Num_chn 16
#define POS_STATE_VECTOR        3                    /* Pos(3), Vel(3),RcvClk(1),RcvClkDrift(1) */
#define VEL_STATE_VECTOR        3                    /* Pos(3), Vel(3),RcvClk(1),RcvClkDrift(1) */
#define MAX_ORBIT_CLKPAR        7                    /* Pos(3), Vel(3),RcvClk(1),RcvClkDrift(1) */
#define Type22_data_length1    37
#define MAX_GPS_SAT1    16
#define MAX_NAV_SAT1    14
#define MAX_NAV_SAT2    4
double lstate[6]={0};
double lGPSstate[Num_chn][6]={0};

FILE *fp1,*fp2,*fp3,*ODPfile4;
unsigned int Frame_ctr =0;
long long temp_val1=0;
   double ltemp_val1 =0;
   unsigned char temp_arr[8];
typedef struct{

 double lEstimate;

}stEpoch;
stEpoch lstEpoch;
void getenggfrombytes(char indx)
{
   char indx1;
   //temp_val1 = receiveBuffer[indx1]| (receiveBuffer[indx1];
}

#pragma pack(push,1)
typedef struct
{
uint16_t usUart_Hdr;
uint8_t ucMsmt_Ctr;
uint8_t ucNav_State;
uint8_t ucAnt_Sts;
      uint8_t ucPVT_Ava_Sts;
uint8_t ucGPS_Sat;
      uint8_t ucNAV_Sat;
uint16_t usGPS_WeekNo; /* GPS week number */
      double      lGPS_TimeOfWeek; /* Seconds of week */
   #include <stdio.h>
   #include <stdlib.h>
   #include <windows.h>
   #include <conio.h>
   #include<time.h>
   #include <stdint.h>
   //#define COM_PORT "COM5"       // Change to your COM port
   #define COM_PORT "COM17"       // Change to your COM port
   #define BAUD_RATE CBR_115200  // Baud rate
   #define TOTAL_BYTES 1800      // Total bytes to transmit
   #define TOTAL_Frame 1800      // Total bytes to transmit
   #define ODP_BYTES 128
   #define BYTE_DELAY_MS 0.7       // Increased delay between bytes
   #define MAX_RETRIES 3        // Number of retries for failed writes
   #define RECEIVE_BUFFER_SIZE TOTAL_BYTES+ODP_BYTES
   #define frame_length    86400  // SIMULATED DATA FL BUT OLD IS   86400
   #define uint8_t unsigned char
   #define uint16_t unsigned short
   #define int64_t long long
   #define uint64_t unsigned long long
   #define Num_chn 16
   #define POS_STATE_VECTOR       3                     /* Pos(3), Vel(3),RcvClk(1),RcvClkDrift(1) */
   #define VEL_STATE_VECTOR       3                     /* Pos(3), Vel(3),RcvClk(1),RcvClkDrift(1) */
   #define MAX_ORBIT_CLKPAR       7                     /* Pos(3), Vel(3),RcvClk(1),RcvClkDrift(1) */
   #define Type22_data_length1     37
   #define MAX_GPS_SAT1    16
   #define MAX_NAV_SAT1    14
   #define MAX_NAV_SAT2    4
   double lstate[6]={0};
   double lGPSstate[Num_chn][6]={0};

  FILE *fp1,*fp2,*fp3,*ODPfile4;
  unsigned int Frame_ctr =0;
  long long temp_val1=0;
     double ltemp_val1 =0;
     unsigned char temp_arr[8];
  typedef struct{

   double lEstimate;

  }stEpoch;
  stEpoch lstEpoch;
  void getenggfrombytes(char indx)
  {
     char indx1;
     //temp_val1 = receiveBuffer[indx1]| (receiveBuffer[indx1];
  }

  #pragma pack(push,1)
  typedef struct
  {
  uint16_t usUart_Hdr;
  uint8_t ucMsmt_Ctr;
  uint8_t ucNav_State;
  uint8_t ucAnt_Sts;
        uint8_t ucPVT_Ava_Sts;
  uint8_t ucGPS_Sat;
        uint8_t ucNAV_Sat;
  uint16_t usGPS_WeekNo; /* GPS week number */
        double      lGPS_TimeOfWeek; /* Seconds of week */
        //uint64_t lTOW_nanosec; /* Seconds of week */
  double lLeoSvPos[POS_STATE_VECTOR];
  float      fLeoSvVel[VEL_STATE_VECTOR];
  float fGdop;
  float fPdop;
        float fDelta_Time;
        uint16_t usChksum;
        uint16_t usNAV_WeekNo; /* GPS week number */
        double      lNAV_TimeOfWeek; /* Seconds of week */
        uint8_t ucType22_data[Type22_data_length1];
        uint8_t ucGPS_SV_StsLB[MAX_GPS_SAT1]; /* Indexes of usable satellites for looping */
  uint8_t ucGPS_SVID[MAX_GPS_SAT1]; /* sat Ids trackedin channedls*/
  uint8_t ucGPS_Cndr[MAX_GPS_SAT1]; /* CNDR trackedin channedls*/
  double lGPS_MeasCode[MAX_GPS_SAT1]; /* lSmoothP1 code */
  double lGPS_MeasDoppler[MAX_GPS_SAT1]; /* Doppler measurement */
  double lGPS_MeasCarrier[MAX_GPS_SAT1]; /* lSmoothP1 carrier */
        double lGPS_Rec_Clk_bias;
  double lGPS_Rec_Clk_drift;
        uint8_t ucEphemeris_SVID_Data[71];//iDelta_Time; need to define structure
  uint16_t usMSg_Rec_Ctr;
        uint8_t ucAST_Debug_Info[22];
  uint8_t ucNAV_SVID[MAX_NAV_SAT1]; /* sat Ids trackedin channedls*/
  uint8_t ucNAV_Cndr[MAX_NAV_SAT1]; /* CNDR trackedin channedls*/
  double lNAV_MeasCode[MAX_NAV_SAT2]; /* lSmoothP1 code */
  double lNAV_MeasDoppler[MAX_NAV_SAT2]; /* Doppler measurement */
  double lGpsSv[MAX_GPS_SAT1][MAX_ORBIT_CLKPAR]; /* sat index wise GPS state vectors */
        int8_t iGPS_Elevation[MAX_GPS_SAT1];
        int16_t iGPS_Azimuth[MAX_GPS_SAT1];
        uint16_t usUart_ASTTx_Ctr;
        uint8_t ucAST_RST_Ctr;
        uint8_t ucAST_RST_ID;
  uint8_t ucGPS_SV_StsMB[MAX_GPS_SAT1]; /* Indexes of usable satellites for looping */
        uint16_t usNAV_SV_Sts[MAX_NAV_SAT1];
        double lNAV_Rec_Clk_bias;
  double lNAV_Rec_Clk_drift;
  uint8_t ucAST_Debug_Spare[40];
  uint16_t usTotal_Chksum;

  }StEpochData_UART1;

  #pragma pack(pop)

  StEpochData_UART1 SEpochData_UART_RX1;
  double getdoublefrom8bytes(const uint8_t *temp_arr1)
  {

        return ((int64_t)((uint64_t)temp_arr1[0] << 0) |
              (int64_t)((uint64_t)temp_arr1[1] << 8) |
              (int64_t)((uint64_t)temp_arr1[2] << 16) |
              (int64_t)((uint64_t)temp_arr1[3] << 24) |
              (int64_t)((uint64_t)temp_arr1[4] << 32) |
              (int64_t)((uint64_t)temp_arr1[5] << 40) |
              (int64_t)((uint64_t)temp_arr1[6] << 48) |
              (int64_t)((uint64_t)temp_arr1[7] << 56));
  }
  int main()
  {
     HANDLE hSerial;
     DCB dcbSerialParams = {0};
     COMMTIMEOUTS timeouts = {0};
     DWORD bytesWritten,bytesRead;
     BOOL writeStatus,readStatus;
     unsigned int ui_frame_ctr =0;
     unsigned int i;
     unsigned char byteToSend,receiveBuffer[RECEIVE_BUFFER_SIZE] = {0};
```

```
int retryCount;
fp1 = fopen("input.dat","wt+");
fp2 = fopen("newoutput.txt","rt+");
fp3 = fopen("outputodp.dat","wt+");
ODPfile4 = fopen("odpEngg.dat","wt+");
printf("UART Single Byte Transmission Program\n");
printf("-----------------------------------\n");
printf("Sending %d bytes one at a time to %s at %d baud\n",
     TOTAL_BYTES, COM_PORT, BAUD_RATE);

// Open the serial port
hSerial = CreateFile(
   "\\\\.\\COM6",
   GENERIC_READ | GENERIC_WRITE,
   0,
   NULL,
   OPEN_EXISTING,
   FILE_ATTRIBUTE_NORMAL,
   NULL);

if (hSerial == INVALID_HANDLE_VALUE)
{
   printf("Error opening serial port! Error code: %d\n", GetLastError());
   return 1;
}

// Set device parameters
dcbSerialParams.DCBlength = sizeof(dcbSerialParams);

if (!GetCommState(hSerial, &dcbSerialParams))
{
   printf("Error getting device state. Error code: %d\n", GetLastError());
   CloseHandle(hSerial);
   return 1;
}

dcbSerialParams.BaudRate = BAUD_RATE;
dcbSerialParams.ByteSize = 8;
dcbSerialParams.StopBits = ONESTOPBIT;
dcbSerialParams.Parity = NOPARITY;

// Disable flow control
dcbSerialParams.fOutxCtsFlow = FALSE;
dcbSerialParams.fRtsControl = RTS_CONTROL_DISABLE;
dcbSerialParams.fOutX = FALSE;
dcbSerialParams.fInX = FALSE;

if (!SetCommState(hSerial, &dcbSerialParams))
{
   printf("Error setting device parameters. Error code: %d\n", GetLastError());
   CloseHandle(hSerial);
   return 1;
}

// Set more generous communication timeouts
timeouts.ReadIntervalTimeout = MAXDWORD; // No interval timeout
timeouts.ReadTotalTimeoutConstant = 300; // 1 second
timeouts.ReadTotalTimeoutMultiplier = 0;
timeouts.WriteTotalTimeoutConstant = 0300; // 1 second
timeouts.WriteTotalTimeoutMultiplier = 0;

if (!SetCommTimeouts(hSerial, &timeouts))
{
   printf("Error setting timeouts. Error code: %d\n", GetLastError());
```

```
    CloseHandle(hSerial);
    return 1;
}

if (!PurgeComm(hSerial, PURGE_RXCLEAR | PURGE_TXCLEAR))
{
    printf("Error flushing port: %d\n", GetLastError());
}
else
{
    printf("Successfully flushed serial buffers\n");
}

printf("Beginning transmission...\n");
while(ui_frame_ctr < frame_length)
{
        ui_frame_ctr++;
        printf("\nInitiated TX");
    /*  fprintf(fp1,"TX "); //uncomment
    // Send bytes one at a time
    for (i = 0; i < TOTAL_BYTES; i++)
    {
        // Create byte value (0-255 repeating pattern)
       // byteToSend = (char)(i % 256) + 0xaa + (0 & 0xff);
        fscanf(fp2, "%02x  ",&byteToSend);

        retryCount = 0;
        do {
           // Send single byte
           writeStatus = WriteFile(
              hSerial,
              &byteToSend,
              1,              // Send exactly 1 byte
              &bytesWritten,
              NULL);

           if (!writeStatus || bytesWritten != 1)
           {
              printf("Warning: Error writing byte %d. Error code: %d (Retry %d/%d)\n",
                  i, GetLastError(), retryCount + 1, MAX_RETRIES);
              Sleep(10 * (retryCount + 1)); // Increasing delay between retries
           }
           retryCount++;
           if(i< 4)
              printf("cntr: %0d : 0x%02X \t", i,(unsigned char)byteToSend);
           // while(!writeStatus){}
         } while ((!writeStatus || bytesWritten != 1) && (retryCount < MAX_RETRIES));

         if (!writeStatus || bytesWritten != 1)
         {
            printf("Fatal error: Failed to write byte %d after %d retries\n", i, MAX_RETRIES);
            CloseHandle(hSerial);
            return 1;
         }
          //fprintf(fp1,"cntr: %0d : 0x%02X \t", i,(unsigned char)byteToSend);
          fprintf(fp1," 0x%02X \t", (unsigned char)byteToSend);  //uncomment
         // Small delay between bytes
         Sleep(BYTE_DELAY_MS);
     }
     for (i = 0; i < TOTAL_Frame-TOTAL_BYTES; i++)
     {
        // Create byte value (0-255 repeating pattern)
       // byteToSend = (char)(i % 256) + 0xaa + (0 & 0xff);
        fscanf(fp2, "%02x  ",&byteToSend);
     }
```

```
printf("\nSuccessfully sent %d bytes one at a time\n", TOTAL_BYTES);

//printf("\nLast byte are %02x %02x %02x %02x\n", TOTAL_BYTES);
Sleep(400);

for (i = 0; i < 1*1000; i++)
{
   for (int j = 0; j < 1*1000; j++)
   {

   }

}
printf("%d",writeStatus);
fprintf(fp1,"\nRX:"); //uncomment
//for (i = 0; i < TOTAL_BYTES; i++)
//{
memset(receiveBuffer,0,sizeof(receiveBuffer));*/
writeStatus = 1;
if(writeStatus == 1)
{
   readStatus = ReadFile(
   hSerial,
   receiveBuffer,
   RECEIVE_BUFFER_SIZE,
   &bytesRead,
   NULL);


   //}
   while(!readStatus){};
   if(readStatus == 1)
   {
      Frame_ctr++;
   }
   printf("\nSuccessfully RECEIVED %d bytes one at a time\n %d", bytesRead,readStatus);

   for (DWORD j = 0; j < RECEIVE_BUFFER_SIZE; j++)
   {
     // fprintf(fp1,"cntr: %0d : 0x%02X \t", i,(unsigned char)byteToSend);
    if(j<10)
       printf(" %0d : 0x%02X \t",j,(unsigned char)receiveBuffer[j]);

       fprintf(fp1," 0x%02X \t",(unsigned char)receiveBuffer[j]);
       if(j>1797)
       {
          // printf("cntr: %0d : 0x%02X \t", j,(unsigned char)receiveBuffer[j]);
           fprintf(fp3," 0x%02X \t",(unsigned char)receiveBuffer[j]);
       }

   }
   printf("\n");
   fprintf(fp1,"\n");
   fprintf(fp3,"\n");
   for (int j = 0; j < 1*1000; j++)
   {

   }
   Sleep(350);

   memcpy(&SEpochData_UART_RX1,&receiveBuffer,sizeof(SEpochData_UART_RX1));

   printf("ODP Flag: %0d :  \t ", (unsigned char)receiveBuffer[1798]);
   temp_val1 = ((unsigned short)receiveBuffer[1800]<<8) | receiveBuffer[1799];
```

```
         printf("ODP WN: %02X %02X \t ",(unsigned char)receiveBuffer[1799],(unsigned
char)receiveBuffer[1800]);

         fprintf(ODPfile4,"\n %4d \t ",(unsigned short)temp_val1);
         printf("Num sat: %0d :  \t ", (unsigned char)receiveBuffer[1809]);
           unsigned char ucindx=0,ucindx1 =0;
           for( ucindx=0; ucindx<8;ucindx++){
              temp_arr[ucindx] = receiveBuffer[1801+ucindx];

           }
           temp_val1 = 0;
           memcpy(&temp_val1, &temp_arr,sizeof(temp_val1));
           printf(" TOW: %lf",temp_val1/1e+6);
            fprintf(ODPfile4," %lf",temp_val1/1e+6);
            fprintf(ODPfile4," %0d :  \t ", (unsigned char)receiveBuffer[1798]);
          //  fprintf(ODPfile4,"%0d :  \t ", (unsigned char)receiveBuffer[1797]);
           // fprintf(ODPfile4,"%0d :  \t ", (unsigned char)receiveBuffer[1796]);

           for(  ucindx1 =0; ucindx1<7;ucindx1++)
           {
              for(  ucindx =0; ucindx<8;ucindx++)
                 {
                    temp_arr[ucindx] = receiveBuffer[1810+(ucindx1*8)+ucindx];
                 }
           lstate[ucindx1] =    (double) getdoublefrom8bytes(temp_arr)/1e+6;
           }


            //printf(" EST:%llx  %lld  %f",ltemp_val1,ltemp_val1,(double)ltemp_val1/1e+6);
           // printf("%02x %02x %02x %02x %02x %02x %02x
%02x",temp_arr[0],temp_arr[1],temp_arr[2],temp_arr[3],temp_arr[4],temp_arr[5],temp_arr[6],temp_arr[7]);

            printf(" \n EST POS-X: %8.6lf    %8.6lf    %8.6lf",lstate[0],lstate[1],lstate[2]);
            printf(" \n EST Vel-X: %8.6lf    %8.6lf    %8.6lf",lstate[3],lstate[4],lstate[5]);
            fprintf(ODPfile4, " %8.6lf    %8.6lf    %8.6lf",lstate[0],lstate[1],lstate[2]);
            fprintf(ODPfile4, " %8.6lf    %8.6lf    %8.6lf",lstate[3],lstate[4],lstate[5]);
            unsigned char nSat_indx =0;
           while(nSat_indx<8)
           {
              for( ucindx1 =0; ucindx1<7;ucindx1++)
              {
                 for( ucindx =0; ucindx<8;ucindx++)
                    {
                       temp_arr[ucindx] = receiveBuffer[1858+(ucindx1*8)+ucindx];
                    }
              lGPSstate[nSat_indx][ucindx1] =    (double) getdoublefrom8bytes(temp_arr)/1e+6;
              }
              nSat_indx++;
           }
            double lRef_State_Err[6]={0};
            for( ucindx1 =0; ucindx1<3;ucindx1++)
            {
               lRef_State_Err[ucindx1]= (double)SEpochData_UART_RX1.lLeoSvPos[ucindx1]-lstate[ucindx1];
               lRef_State_Err[ucindx1+3]= (double)SEpochData_UART_RX1.fLeoSvVel[ucindx1]-lstate[ucindx1+3];


            }
            printf(" \n LDZ: POS-X: %8.6lf    %8.6lf
%8.6lf",lRef_State_Err[0],lRef_State_Err[1],lRef_State_Err[2]);
             printf(" \n LDZ: Vel-X: %8.6lf    %8.6lf    %8.6lf",lRef_State_Err[3],lRef_State_Err[4],lRef_State_Err[5]);
             fprintf(ODPfile4," \t LDZ: POS-X: %8.6lf    %8.6lf
%8.6lf",lRef_State_Err[0],lRef_State_Err[1],lRef_State_Err[2]);
             fprintf(ODPfile4," \t LDZ: Vel-X: %8.6lf    %8.6lf
%8.6lf",lRef_State_Err[3],lRef_State_Err[4],lRef_State_Err[5]);
```

```
        }
        //else
            //Sleep(200);
    }
    if (!CloseHandle(hSerial))
        {
            printf("Error closing port: %d\n", GetLastError());
            return 1;
        }

    printf("Serial port successfully closed\n");
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);

    // Close the serial port (only once)
    return 0;
}
```