UART APPLICATION NOTES

GENERAL APPLICATION NOTE FOR STARTECH UART FAMILY

Printed December 17, 1996

The AN-450 provides additional information to guide users to design or utilize the STARTECH product line. This document can also be used for all the STARTECH UART product lines.

GENERAL INFORMATION

STARTECH offers UART's with or without FIFO capabilities, and are marked as 45X for non FIFO families and 55X for FIFO families. All parts with sharing part numbers are foot print compatible in some extent, like ST16C450 and ST16C550, ST16C2450 and ST16C2550, etc.

This section will describe general terms for commonly used flags and registers.

OVERRUN ERROR:
The flag is set to "1" to warn the user that a serial data has been received and previous serial data has not been read from receive holding register. The new serial data will over write the previous data in the receive holding register. Note that previous serial data has been lost and user does not have an access to that data.

PARITY ERROR:
This flag is set "1" to indicate that received serial data contains mismatched parity or data bit error in the received data.

PARITY:
Four common types of parities are used in the STARTECH Uart families; Odd Parity, Even Parity, Forced Mark Parity and Forced Space Parity.

ODD PARITY:
Odd Parity is calculated by adding all the "1's" in a data stream and adding a parity bit to the total bits, to make the total count an odd number.

Example -1: A data byte with the following pattern 11010010 will require to add a parity bit of "1" to bring the total count for "1's" to an odd number. Based on this data pattern, serial data with odd parity will be transmitted as 110100101.

Example -2: A data byte with the following pattern 10011000 will require to add a parity bit of "0" to maintain the total count of "1's" to an odd number. Based on this data pattern, serial data with odd parity will be transmitted as 100110000.

EVEN PARITY:
Even Parity is calculated by adding all the "1's" in a data stream and adding a parity bit to the total bits, to make the count an even number.

Example -3: A data byte with the following pattern 10000101 will require to add a parity bit of "1" to bring the total count for "1's" to an even number. Based on this data pattern, serial data with even parity will be transmitted as 100001011.

Example -4: A data byte with the following pattern 00001111 will require to add a parity bit of "0" to maintain the total count for "1's" to an even number. Based on this data pattern, serial data with even parity, will be transmitted as 000011110.

FORCED SPACE PARITY:
Parity bit on the serial byte is set to "0" regardless of total number of "1's" (even or odd counts).

FORCED MARK PARITY:
Parity bit on the serial byte is set to "1" regardless of total number of "1's" (even or odd counts).

FRAMING ERROR:
The flag is set to "1" to indicate that received data does not have correct start or stop bits. This can cause when the Uarts are set for 8-bits word and receiving a serial data of 7-bits word or any mismatched data patterns.

BREAK SIGNAL INDICATION:
This flag is set to "1" to warn the user that transmitter is sending continuous "0" data without stop bit (RX input is low for more that one word).

TRANSMIT/RECEIVE FIFO:
STARTECH offers 16 byte transmit FIFO and 16 byte receive FIFO for all its products with 55X part numbers. These FIFO's are static 19 X 16 bit RAM with control logic to form a ring counter. Initializing the FIFO will set the write and read pointers to the same location.

TRANSMIT EMPTY:
This flag is set "1" to indicate that, there is no character in the transmit holding and transmit shift register

TRANSMIT HOLDING EMPTY:
This flag is set "1" to indicate that, there is one or more empty locations in the transmit holding register. User has to check this bit before loading characters in the transmit holding register. In non FIFO mode, user can load one character at a time when this flag is set and 16 characters when FIFO mode is utilized.

RECEIVER DATA READY:
This bit is set "1" to indicate that, receiver has one or more character in the receive holding register. User has to check this bit prior to read receive holding register. In non FIFO mode, only one character at time can be read. In FIFO mode up to 16 characters can be read if time bit is set.

RECEIVE TIME-OUT:
This mode is enabled when STARTECH UART is operating in FIFO mode. Receive time out will not occur if the receive FIFO is empty. The time out counter will be reset at the center of each stop bit received or each time receive holding register is read. The actual time out value is T (**T**ime out length in bits)= 4 X P (**P**rogrammed word length) + 12. To convert time out value to a character value, user has to divide this number to its complete word length + parity (if used) + number of stop bits and start bit.

Example -7: If user programs the word length = 7, and no parity and one stop bit, Time out will be:
T = 4 X 7(programmed word length) +12 = 40 bits
Character time = 40 / 9 [(programmed word length = 7) + (stop bit = 1) + (start bit = 1)] = 4.4 characters.

Example -8: If user programs the word length = 7, with parity and one stop bit, the time out will be:
T = 4 X 7(programmed word length) + 12 = 40 bits
Character time = 40 / 10 [(programmed word length = 7) + (parity = 1) + (stop bit = 1) + (start bit = 1) = 4 characters.

BAUD RATE GENERATOR:
STARTECH provides a 16 bit digital divider to obtain all necessary baud rates. The 16 bit divider is broken down in to two 8-bit dividers which will be addressed as MSB divider (upper 8-bits) and LSB divider (lower 8-bits). To calculate the transmit/receive data rate it is necessary to know the provided clock rate (frequency) to STARTECH parts. STARTECH utilizes 16 clocks for each transmit bit and 16 clocks to sample the received data. Note that inorder to access these dividers, user has to enable the divisor latch access bit through the Line Control Register.

Bit rate is calculated by:
Dividing decimal number = (Clock rate) / (16 X bit rate).
To program the digital divider, dividing decimal number should be converted to hex (base 16) number and split into two 8-bits sections.

Example -5: To obtain 4800 Hz baud rate, assuming 1.8432 MHz input clock, the dividing decimal value is (input clock=1843200) / (16 X 4800) = 24

24 decimal = 0018 Hex, this value is translated to MSB = 00 Hex and LSB = 18 Hex.

BAUD RATE VERSUS BIT RATE:
The baud rate defines the width of each bit regardless of word, parity and stop bit length. Bit rate, is the rate of the transmission which each character is transmitted or received. The 2400 baud rate transmission is translated to 2400 Hz per bit for each character in a word. With 2400 baud you can transmit between 7 to 12 characters per slot.

PROGRAMMING STEPS:
The AN-450 provides the easy steps to program STARTECH Uart family. Note that all numbers are in Hex format not decimal.

Write 80 Hex to **LCR** (**L**ine **C**ontrol **R**egister) to enable baud rate generator divider latch
to set 2400 Hz baud rate:
write 00 Hex to MSB of baud rate generator (address location 1).
Write 30 Hex to LSB of baud rate generator (address location 0).

Select you word, parity and stop bit format from STARTECH Uart data sheet.
to set 8 bits, no parity and one top bit and disable the divisor access latch
write 03 Hex to **LCR** (**L**ine **C**ontrol **R**egister):

if you need to use Uarts with FIFO, select your receive trigger level from data sheet.
to enable FIFO with 14 character trigger level
write CF Hex to **FCR** (**F**IFO **C**ontrol **R**egister)

enable interrupt sources
write 01 Hex to **IER** (**I**nterrupt **E**nable **R**egister) to select receive interrupt.

to set RTS and DTR outputs to low and enable the interrupt output
write 0B Hex to **MCR** (**M**odem **C**ontrol **R**egister).

The STARTECH Uart is ready for transmit and receive operation.

Read **MSR** (**M**odem **S**tatus **R**egister) to check the status of **CD, RI, DSR, CTS** input pins.

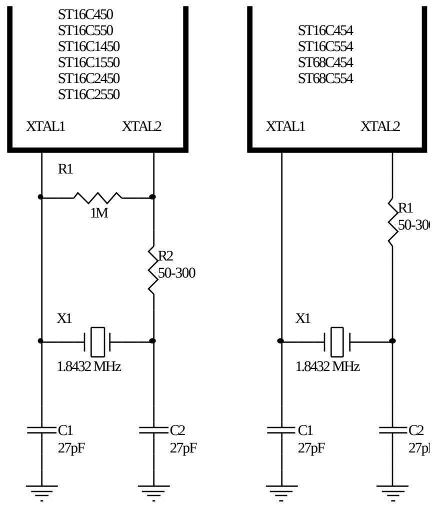
Read **LSR** (**L**ine **S**tatus **R**egister).

For polling applications (non interrupt mode) user has to monitor bit zero of this register to verify valid data in the receive holding register.

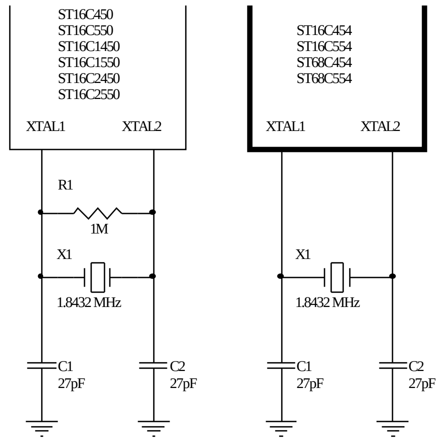
Check the Transmit Holding Empty bit before loading data in the transmit holding register,

continue the transmission.

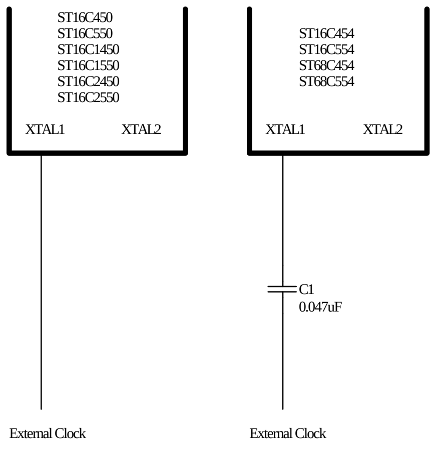
Serial Crystal Connections



Parallel Crystal Connections



External Clock Connections



C PROGRAM SAMPLE

```
; File: sample.c            Package:UART init
; This is a sample code to show how to initialize the UART series of chips
; from Startech Semiconductors.
; This also includes some basic external loop back thru' two different
; ports using the FIFO capability.
; This also includes external loop back thru a different computer

#include      <stdio.h>
#include      <string.h>
#include      <fcntl.h>

#define       TRUE                     1
#define       FALSE                    0

/* These are the various offsets for the registers inside the chip */
#define       RHR                      0x00  /* Receive Holding Register     */
#define       THR                      0x00  /* Receive Holding Register     */
#define       IER                      0x01  /* Interrupt Enable Register    */
#define       FCR                      0x02  /* FIFO control Register        */
#define       ISR                      0x02  /* Interrupt Status Register    */
#define       LCR                      0x03  /* Line control register        */
#define       MCR                      0x04  /* Modem Control Register       */
#define       LSR                      0x05  /* Line Status Register         */
#define       MSR                      0x06  /* Modem Status Register        */
#define       SCR                      0x07  /* Scratch pad Register         */

/* This two offsets are used for defining the baud rate       */
#define       DIVLSB                   0x00  /* Divisor LSB latch address    */
#define       DIVMSB                   0x01  /* Divisor MSB Latch address    */

/*\
 * Program table for baud rate
 * This represents the LSB and MSB divisor latch data
\*/
char baud_table[8][2] = {
            { 0x80, 0x01 },            /* 300  */
            { 0x60, 0x00 },            /* 1200 */
            { 0x30, 0x00 },            /* 2400 */
            { 0x0c, 0x00 },            /* 9600 */
            { 0x06, 0x00 },            /* 19K  */
            { 0x03, 0x00 },            /* 38k  */
            { 0x02, 0x00 },            /* 56k  */
            { 0x01, 0x00 }             /* 115k */
};
```

```
/* Baud Rates */
#define     _COM_300_           0
#define     _COM_1200_          1
#define     _COM_2400_          2
#define     _COM_9600_          3
#define     _COM_19K_           4
#define     _COM_38K_           5
#define     _COM_56K_           6
#define     _COM_115K_          7

/* Parity */
#define     _COM_NOPARITY_      0
#define     _COM_ODDPARITY_     1
#define     _COM_EVENPARITY_    2

/* Stopbits  */
#define     _COM_STOP1_         0
#define     _COM_STOP2_         1
#define     _COM_STOP1_5_       1

/* word length  */
#define     _COM_CHR5_          0
#define     _COM_CHR6_          1
#define     _COM_CHR7_          2
#define     _COM_CHR8_          3

/* word length  */
#define     _COM_FIFO1_         0
#define     _COM_FIFO4_         1
#define     _COM_FIFO8_         2
#define     _COM_FIFO14_        3

/*\
 * This function checks the existence of a port.
 * It is very simple. Take the port address then write to the scratch pad
 * an the read it back. If the data read back the same as one that was
 * written then return TRUE else return FALSE.
\*/
int
check_port(com_port)
int com_port;
{

  int i;

       printf("Checking for port %4xH\n",com_port);
  /* Write 1010 1010 (0xaa) to scratch pad*/
```

```
      printf(“Writing AAH in %4xH\n”,com_port);
   outportb(com_port + SCR, 0xaa);

   /* read it back. If it the same then return TRUE */
   i = inportb(com_port + SCR);

      printf(“Read back %2xH from %4xH\n”,i,com_port);

   if( i == 0xaa)
        return TRUE;
   else
        return FALSE;

}

/*\
 * This is the work horse function which actually setups the UART.
 * It needs to know every thing.
\*/
int
init_uart(port,baud,parity,data,stop,fifo,trigger)
int  port,baud,parity,data,stop,fifo,trigger;
{

   char lcr_byte;

   /* Set divisor latch */
   outportb(port+LCR, 0x80) ;

   printf(“Divisor Latch is %2xH %2xH (High Low)\n”,
                                   baud_table[baud][1],baud_table[baud][0]);
   outportb(port+DIVLSB, baud_table[baud][0]) ;
   outportb(port+DIVMSB, baud_table[baud][1]) ;


   /* Reset to normal Programming */
   /* Program the lcr_byte for the above parameters */
   lcr_byte = 0x00;
   lcr_byte = data; /* Set the  bit0 & bit1 for word length */
   lcr_byte ;= stop << 3; /* Set the bit2 for stop bit */
   if(parity != _COM_NOPARITY_) {
       lcr_byte ;= 1 << 4;  /* Set the bit3 for parity */
       if(parity == _COM_EVENPARITY_)
          lcr_byte ;= 1 << 5; /* Set the bit4 for EVEN parity */
   }

   printf(“LCR byte is %2xH\n”,lcr_byte);
   /* Program LCR  */
   outportb(port+LCR, lcr_byte) ;

   if(fifo) {
     char fifo_byte;

     printf("Programming FIFOs without DMA mode\n");

     /* Have to first set the fifo enable */
     fifo_byte = 0x01;
     outportb(port+FCR,fifo_byte);

     /* Now program the FIFO */
     fifo_byte = 0x07; /* set bit0 - FIFO enable, Reset RCVR and XMIT FIFO */
     fifo_byte ;= trigger << 7; /* set bit6 and bit7 with the trigger level */

     /* Program FCR */
     outportb(port+FCR,fifo_byte);
     if(~(inportb(port + ISR) & 0xc0)) {
          printf("This port %4xH does not have FIFOs\n");
          printf("Hence did not program Enable FIFOs\n");
     }
   }

   /* Program IER */
   printf("Programming IER for interrupt on bit0 RCV holding Register\n");
   outportb(port+IER, 0x01);

   return TRUE;
}

/*\
 * This is the test mode.
 * It gets the address of the ports checks to see if they are there.
 * Note: If a driver already exists I am not sure how to temporarily remove it.
 * Well we will worry about it later.
 * Warn the use to remove any drivers that are on the ports.
 * Especially the mouse driver.
 * pass the address to the test552 routine.
\*/
int test_mode()
{
          int i,j,k; /* generic variables */
          char port1[10], port2[10];
          int pt1,pt2;  /* this are the integer port numbers */

          void test552();

          printf("WARNING: This program will not work if the ports to be tested\n");
          printf("        have drivers installed in them. e.g Mouse driver\n");
          printf("        Please remove the drivers before doing this test.\n");

          while(TRUE) {
             printf("First Port Address (In HEX) > ");
             scanf("%s",port1);
             pt1 = strtol(port1,NULL,16);
             fflush(stdin);
             /*\
              * Check if this port exists. else loop
             \*/
              if(check_port(pt1))
                    break;
              printf("Error: Port %4xH does not exist. Try again\n",pt1);
          }


          while(TRUE) {
             printf("Second Port Address (In HEX) > ");
             scanf("%s",port2);
             pt2 = strtol(port2,NULL,16);
             fflush(stdin);
             /*\
              * Check if this port exists. else loop
             \*/
              if(check_port(pt2))
                    break;
              printf("Error: Port %4xH does not exist. Try again\n",pt2);
          }

          /* Test 554 with the two port addresses */
          test552(pt1,pt2);

          return TRUE;

}



/*\
 * It first generates a random number for the data size to be generated.
 * Then generates a random data whose length is equal to the data size.
 * It puts it out on both the ports and polls for the interrupt to occur.
 * It reads both the ports until all characters are received OR a timeout
 * has occured. It then prints out the error Messages if any.
 * This loop is done for ever.
\*/
```

```
void test552(p1,p2)
unsigned int p1, p2;
{
   int i,j,c,w,n;
   unsigned char outbuf[20], inbuf1[20], inbuf2[20];
   unsigned char pbuf[200];
   unsigned long timeout, pass;


   printf(“ST16C552 External Loop Test Beginning\n”) ;
   printf(“Testing ports %4x and %4x\n\n”, p1, p2) ;
   printf(“Programing ports for 56K,8 bit,no parity,1 stop bit,FIFO trigger level 01\n”);
   printf(“This program uses POLLED mode for testing\n”);
   printf(“Press Cntrl-C to stop the testing and quit\n”);
   printf(“Note: The ports will remain at the above settings after the TEST\n”);

   /* Programming ports for 8 bits, no parity, 56K baud,
                                       FIFO enabled at level 01 */
   /* Program first port */
   printf(“Programming port %x4\n”,p1);
   init_uart(p1,_COM_56K_,_COM_NOPARITY_,
               _COM_CHR8_,_COM_STOP1_,TRUE,_COM_FIFO1_);

   /* Program Second Port */
   printf(“Programming port %x4\n”,p2);
   init_uart(p2,_COM_56K_,_COM_NOPARITY_,
               _COM_CHR8_,_COM_STOP1_,TRUE,_COM_FIFO1_);

   printf(“Starting test\n”);
   for (pass = 1 ; ; pass++) {
          /* generate random size for data */
          n = rand() ;
          n += n >> 8 ;
          n &= 0x0f ;

          /* Make sure we never get a 0 as the random size data */
          if(n != 0x0f)
            n++ ;

          /* generate random data */
          for (w = 0 ; w < n ; w++) {
                 c = rand() ;
                 c += c >> 8 ;
                 c &= 0xff ;
                 c ;= 0x01 ; /* no NULLs allowed */
                 outbuf[w] = c ;
          }
```

```
outbuf[w] = NULL;

printf("******** Pass %10ld Sending %d *********\015", pass, n) ;

/* Transmitt the data */
for (i = 0 ; i < n ; i++ ) {
        outportb(p1, outbuf[i]) ;
        outportb(p2, outbuf[i]) ;
}

/* loop waiting for intr pending */
for ( i = 0;;i++ ) {
        if ((~inportb(p1+ISR) & 0x01) && (~inportb(p2+ISR) & 0x01))
                break;
}

/* receive data until all has been received OR timeout */
timeout = 0x0008F ;
for (i = j = 0; ((i < 20) && (j < 20));) {
        if (inportb(p1+LSR) & 0x01) inbuf1[i++] = inportb(p1) ;
        c = rand() ;
        c += c >> 8 ;
        c &= 0x001f ;
        c++ ;
        for ( ; c != 0; c—) ;
        if (inportb(p2+LSR) & 0x01) inbuf2[j++] = inportb(p2) ;
        if (timeout— == 0) break ;
}

/* If timed out then print message else comparse data */
if(timeout == 0 )
        printf("Timed out on Ports\n");
else {
   inbuf1[i] = inbuf2[j] = NULL;
   /* compare results */
   if (strcmp(outbuf, inbuf1) ;; ( i != n)) {
           printf("\nError:%04x Sent:    ", p2) ;
           for ( w = 0; w < n; w++ )
                   printf(" %02x", outbuf[w]) ;
           printf("\n%04x Received:", p1) ;
           for ( w = 0; w < i; w++ )
                   printf(" %02x", inbuf1[w]) ;
           printf("\n") ;
   }
   if (strcmp(outbuf, inbuf2) ;; ( j != n )) {
           printf("\nError:%04x Sent:    ", p1);
           for ( w = 0; w < n; w++ )
                   printf(" %02x", outbuf[w]) ;
```

```
                  printf("\n%04x Received:", p2) ;
                  for ( w = 0; w < j; w++ )
                        printf(" %02x", inbuf2[w]) ;
                  printf("\n") ;
            }
         }
   }
}
```