Experiment No. 8
SERIAL COMMUNICATION USING THE ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER (ACIA)
ECE 441

Peter Chinetti

December 1, 2013

Date Performed:	November 21, 2013
Partners:	Zelin Wu
Instructor:	Professor Saniie

1 Introduction

1.1 Purpose

The purpose of this experiment is to introduce the student to the following topics:

- the Asynchronous Communications Interface Adapter IC (ACIA, MC6850)
- the fundamentals of serial asynchronous data communications
- the RS-232-C Serial Communications Standard

1.2 Background

1.3 The Asynchronous Communications Interface Adapter (ACIA)

The ACIA (MC6850) provides the data formatting and control to interface serial asynchronous data communications systems to parallel bus systems. When the CPU writes data to the ACIA in a parallel format, the ACIA performs a parallel-to-serial conversion before transmitting the data serially. Similarly, when the ACIA receives data in a serial format, it performs a serial-to-parallel conversion, which enables the CPU to read the data in a parallel format.

1.4 Asynchronous Serial Communication

In serial communication systems, a byte of data is transmitted one bit at a time along the same physical wire. In asynchronous serial communications systems, start and stop bits are added before and after the data to inform the receiving device as to where the data begins and ends. When transmitting serial data asynchronously, the data packet must adhere to the following specific format.
The first bit transmitted is the Start Bit, and this bit indicates the beginning of a character word. This bit is always a logic 0.
Next, 7 or 8 Data Bits are transmitted, one bit at a time, starting with the Least Significant Bit or LSB (i.e. D0), and increasing towards the Most Significant Bit or MSB (i.e. D7).
The Parity Bit is sent next. It can be either logic 0 or 1 depending upon the data and the type of parity selected in the ACIAs Control Register.
Finally the Stop Bits are transmitted. These bits indicate the end of a character word. There can be either 1 or 2 Stop Bits, and they are always logic 1.

1.5 RS-232-C Serial Communications Standard

RS-232-C is the name given to the hardware standard for the serial transmission of data from one computer to another computer or peripheral device. The voltage levels for a logic 0 are +3 to +15 Volts, and for a logic 1 are -3 to -15 Volts. For ACIA #1 and ACIA #2, the RS-232 signals are made available on two 25-pin connectors (plug type DB-25 Connectors) at the back of the SANPER-1 Educational Lab Unit.
In the SANPER-1 Unit, the TTL serial data being transmitted by the ACIA is first inverted, and then converted to RS-232-C type voltages by an integrated circuit known as a TTL to RS-232-C Converter (Motorola Part No.: MC1488). This device converts the TTL signals (0 or +5 Volts) to RS-232 signals (-3 to -15 Volts or +3 to +15 Volts). The RS-232 data is then sent to the receiving computer or peripheral device.
When RS-232 data is received, it is inverted and then converted to TTL level voltages by an integrated circuit known as a RS-232-C to TTL Converter (Motorola Part No. Experiment #8 - 3MC1489). This device converts the RS-232 signals (-3 to -15 Volts or +3 to +15 Volts) to TTL signals (0 or +5 Volts). The TTL data is then input to the ACIA on the RX DATA pin.

2 Lab Procedure and Equipment List

2.1 Equipment

- SANPER System
- Computer with TUTOR software

2.2 Procedure

Test ACIA with a test program.

3 Results, Analysis and Discussion

3.1 Test Program

```
  ORG $2100

FERROR  DC.B   ' MADE A FRAMING ERROR'
EFERROR DC.B   $00
OVRN   DC.B   'YALL MADE A OVERRUN ERROR'
EOVRN DC.B   $00
DTD DC.B      'DTD ERROR'
EDTD   DC.B   $00

***********************************
SUBRT1
*INITIALIZE
  MOVE.B  #$43,$10041 ;reset
  MOVE.B  #$15,$10041 ;set output
  RTS
***********************************

***********************************
SUBRT7
*ENABLE INTERRUPTS
  MOVE.B  #$43,$10041 ;reset
  MOVE.B  #$95,$10041 ;set with interrupts
  RTS
***********************************

***********************************
SUBRT8
*INTERRUPT SERVICE ROUTINE
  move.b $10041,d0
  btst #0,d0
  bne READ_REQ
  btst #2,d0
  bne DTD_IRQ
*CHECK IF ERROR FREE
READ_REQ
  BTST.B  #4,d0
  beq next_check
  BSR ERROR
  rts
next_check  BTST.B  #5,$10041
  beq passed_checks
  BSR ERROR2
  rts
passed_checks
  move.b $10043,(A0)+
  bsr check_table
  rts
```

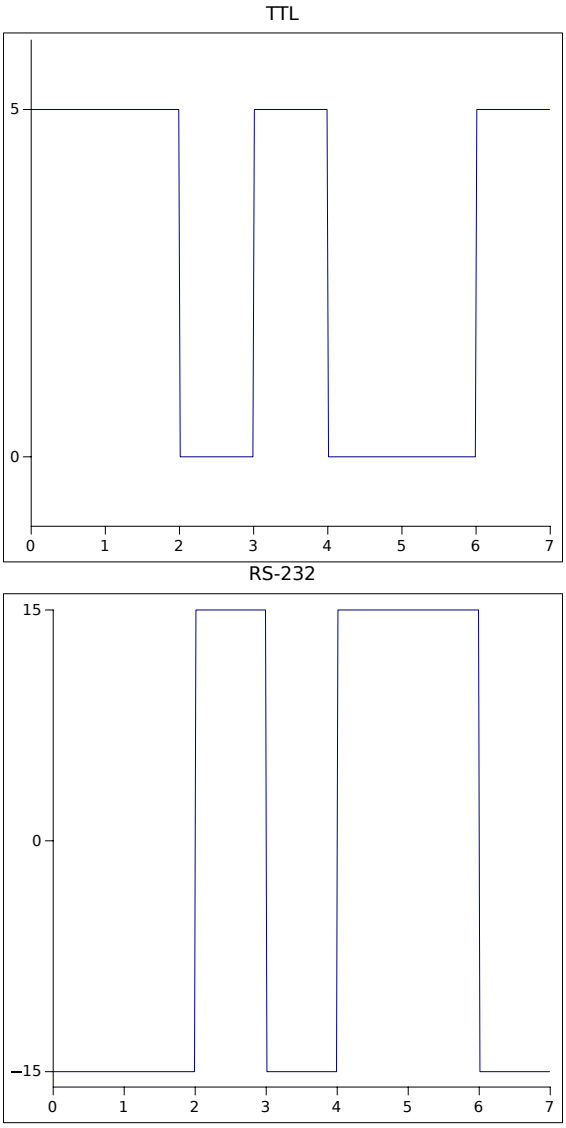
```
check_table
49    cmpa #$919,a0
      blt check_table_done
51    move.l #8,d0 ; init counter
check_table_loop
53    bsr subrt4
      dbf d0,check_table_loop
55    lea #$910,a0
check_table_done
57    rts
DTD_IRQ
59    LEA DTD,A5
      LEA EDTD,A6
61    MOVE.B  #227,D7
      TRAP  #14
63    RTS
************************************
65 SUBRT2
*FILL
67
INPUT   MOVE.B  #241,D7 ;set up read trap
69    TRAP  #14
      RTS
71


73 TRANS
      BTST.B  #1,$10041 ;see if clear to send
75    BEQ TRANS ;spin if not clear
      MOVE.B  (A5)+,$10043 ; write out chars
77    RTS
************************************
79 SUBRT3
      BTST.B  #0,$10041 ; check if data recieved
81    BEQ SUBRT3 ;spin
*CHECK IF ERROR FREE
83    BTST.B  #4,$10041 ;check for framing error
      BNE ERROR
85    BTST.B  #5,$10041 ;check for overrun error
      BNE ERROR2
87 *NO ERROR
      MOVE.B  $10043,D0
89    RTS
********************************
91 SUBRT4
      BSR SUBRT3
93    BSR PRINT
      RTS
95

**********************************
97 ERROR LEA FERROR,A5
      LEA ERROR,A6
99    MOVE.B  #227,D7
      TRAP  #14
101   RTS
ERROR2  LEA OVRN,A5
103   LEA EOVRN,A6
      MOVE.B  #227,D7
```

```
105    TRAP   #14
       RTS
107
109 PRINT MOVE.B   #248,D7     ; PRINT A CHAR IN D0
       TRAP   #14
111    RTS
113
    ***************************
115 SUBRT5
       MOVE.B   #9,D2 ;set up counter
117    MOVEA.L #$900,A5 ;set up input buffer addrs
       MOVEA.L #$900,A6
119
       BSR    SUBRT1 ;init ACIA
121    BSR INPUT ;read char
    LOOP:
123
       BSR TRANS ;write char
125    BSR SUBRT3 ;read char
       BSR PRINT
127    DBF D2,LOOP ;decrement loop
       RTS
129
    **********************
131 SUBRT6
133    bsr subrt1 ;init
       move.l #8,d2 ;loop 9 times
135 6_loop   bsr subrt4 ;read, then output to terminal
       dbf d2,6_loop ;this does it one at a time
137    rts
139
141
143 START
    DONE
145    MOVE.B   #228,D7
       TRAP   #14
147
149
       END START    ; last line of source
```

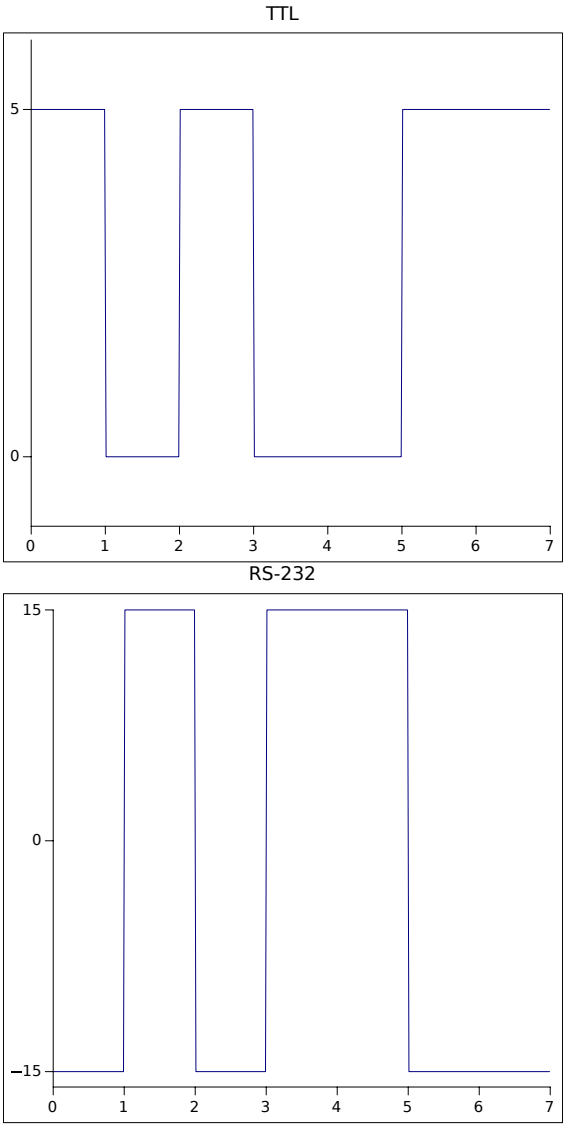
LAB8.X68

3.2 TTL/RS232

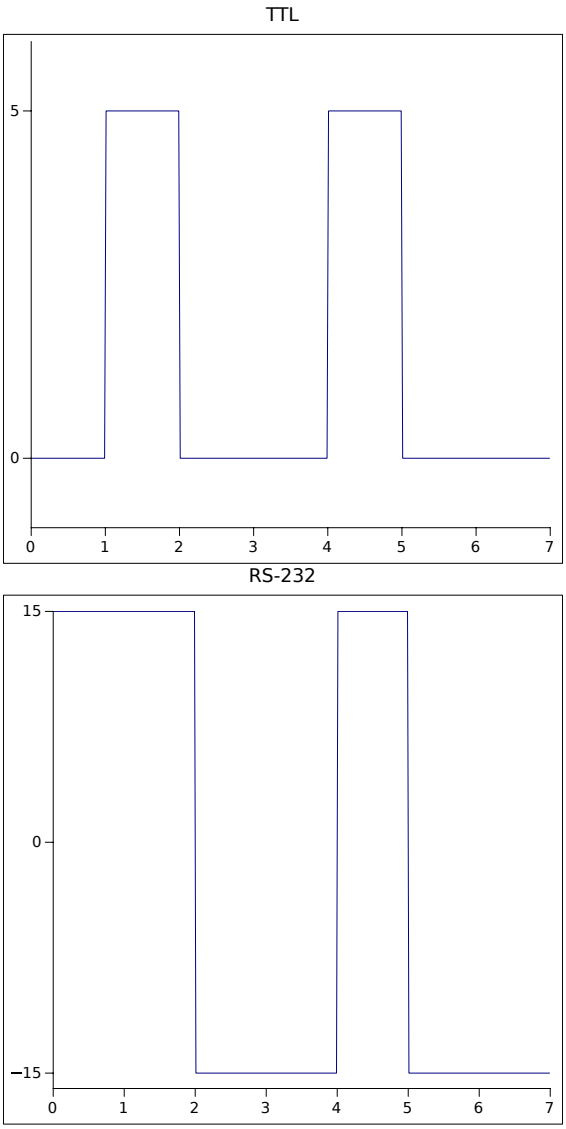
3.2.1 i



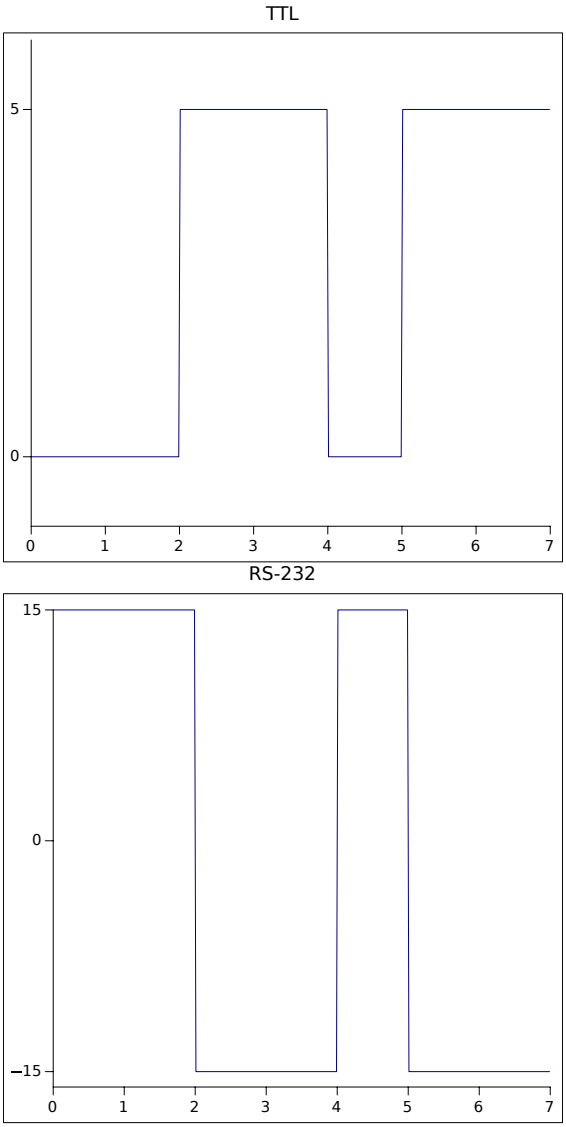
3.2.2 S



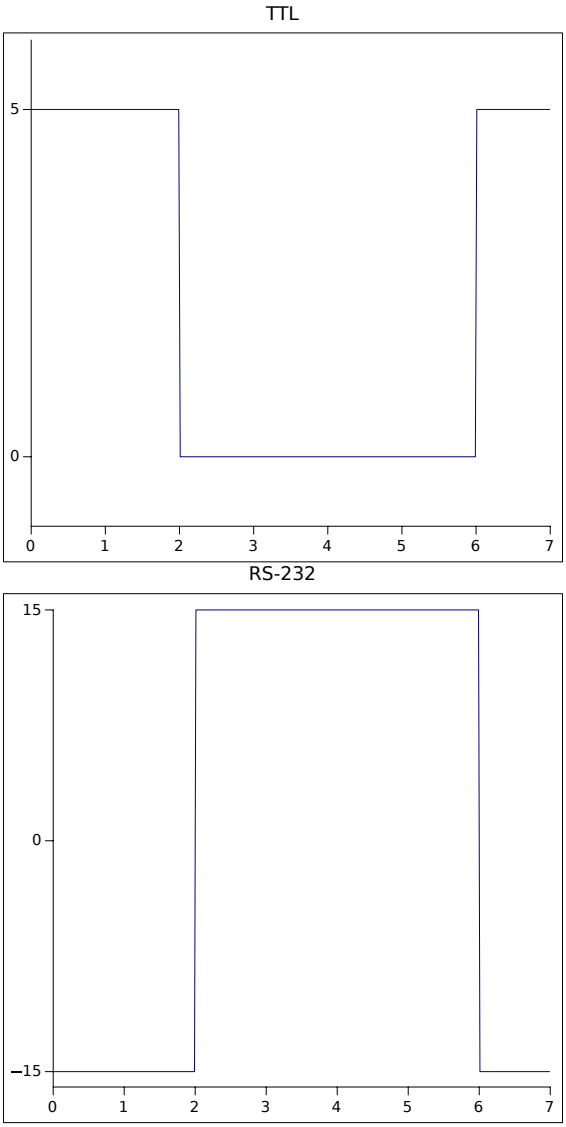
3.2.3 $



3.2.4 ESC



3.2.5 a



3.3 Missing Block

A simple reason why there could be a missing block is that the transmitter could have sent a block before the receiver was listening. This could be fixed by properly connecting RTS and CTS, rather than tying to ground.

3.4 Polling vs. Interrupts

Polling is easier to implement, and will never interrupt critical sections of code. Interrupts are more efficient and allow more total work to be done in normal use cases, however, they are harder to implement.

3.5 SR Interrupt bits

SR bits 2 (DCD) and 0 (RDRF), can be configured to raise interrupts. The DCD interrupt is raised when it is enabled in the CR, and the DTD line goes high. The RDRF interrupt is raised when it is enabled in the CR and the Receive Data Register is filled.

3.6 SR=0xA3

Interrupt true, no parity error, overrun error, no framing error, Clear To Send, DCD error, transmit register empty, receive register empty.

3.7 CR=0xC2

Interrupts enabled, RTS high transmit interrupts inhibited, 7-E-2, 64 division sampling

3.8 CR=0x81 (Even Parity)

3.8.1 !

Parity = 1

3.8.2 7

Parity = 0

3.8.3 N

Parity = 0

3.8.4 P

Parity = 1

4 Conclusions

This experiment was accomplished. An ACIA was tested.