XMODEM/CRC Send and Receive

```
; XMODEM/CRC Sender/Receiver for the 65C02
;
; By Daryl Rictor Aug 2002
;
; A simple file transfer program to allow transfers between the SBC and a
; console device utilizing the x-modem/CRC transfer protocol.  Requires
; ~1200 bytes of either RAM or ROM, 132 bytes of RAM for the receive buffer,
; and 12 bytes of zero page RAM for variable storage.
;
;***************************************************************************
; This implementation of XMODEM/CRC does NOT conform strictly to the
; XMODEM protocol standard in that it (1) does not accurately time character
; reception or (2) fall back to the Checksum mode.

; (1) For timing, it uses a crude timing loop to provide approximate
; delays.  These have been calibrated against a 1MHz CPU clock.  I have
; found that CPU clock speed of up to 5MHz also work but may not in
; every case.  Windows HyperTerminal worked quite well at both speeds!
;
; (2) Most modern terminal programs support XMODEM/CRC which can detect a
; wider range of transmission errors so the fallback to the simple checksum
; calculation was not implemented to save space.
;***************************************************************************
;
; Files transferred via XMODEM-CRC will have the load address contained in
; the first two bytes in little-endian format:
;  FIRST BLOCK
;     offset(0) = lo(load start address),
;     offset(1) = hi(load start address)
;     offset(2) = data byte (0)
;     offset(n) = data byte (n-2)
;
; Subsequent blocks
;     offset(n) = data byte (n)
;
; One note, XMODEM send 128 byte blocks.  If the block of memory that
; you wish to save is smaller than the 128 byte block boundary, then
; the last block will be padded with zeros.  Upon reloading, the
; data will be written back to the original location.  In addition, the
; padded zeros WILL also be written into RAM, which could overwrite other
; data.
;
;-------------------------- The Code ----------------------------
;
; zero page variables (adjust these to suit your needs)
;
;
```

```
lastblk     =    $35        ; flag for last block
blkno       =    $36        ; block number
errcnt      =    $37        ; error counter 10 is the limit
bflag       =    $37        ; block flag

crc     =    $38        ; CRC lo byte  (two byte variable)
crch        =    $39        ; CRC hi byte

ptr     =    $3a        ; data pointer (two byte variable)
ptrh        =    $3b        ;   "    "

eofp        =    $3c        ; end of file address pointer (2 bytes)
eofph       =    $3d        ;  "    "    "    "

retry       =    $3e        ; retry counter
retry2      =    $3f        ; 2nd counter

;
;
; non-zero page variables and buffers
;
;
Rbuff       =    $0300          ; temp 132 byte receive buffer
                   ;(place anywhere, page aligned)
;
;
;  tables and constants
;
;
; The crclo & crchi labels are used to point to a lookup table to calculate
; the CRC for the 128 byte data blocks.  There are two implementations of
these
; tables.  One is to use the tables included (defined towards the end of
this
; file) and the other is to build them at run-time.  If building at run-
time,
; then these two labels will need to be un-commented and declared in RAM.
;
;crclo      =    $7D00          ; Two 256-byte tables for quick lookup
;crchi      =     $7E00          ; (should be page-aligned for speed)
;
;
;
; XMODEM Control Character Constants
SOH     =    $01        ; start block
EOT     =    $04        ; end of text marker
ACK     =    $06        ; good block acknowledged
NAK     =    $15        ; bad block acknowledged
CAN     =    $18        ; cancel (not standard, not supported)
CR      =    $0d        ; carriage return
LF      =    $0a        ; line feed
```

```
ESC     =    $1b        ; ESC to exit

;
;^^^^^^^^^^^^^^^^^^^^^^ Start of Program ^^^^^^^^^^^^^^^^^^^^^^
;
; Xmodem/CRC transfer routines
; By Daryl Rictor, August 8, 2002
;
; v1.0  released on Aug 8, 2002.
;
;
        *=     $FA00        ; Start of program (adjust to your needs)
;
; Enter this routine with the beginning address stored in the zero page
address
; pointed to by ptr & ptrh and the ending address stored in the zero page
address
; pointed to by eofp & eofph.
;
;
        jmp    XmodemRcv    ; quick jmp table
XModemSend  jsr    PrintMsg    ; send prompt and info
        lda    #$00        ;
        sta    errcnt        ; error counter set to 0
        sta    lastblk        ; set flag to false
        lda    #$01        ;
        sta    blkno        ; set block # to 1
Wait4CRC    lda    #$ff        ; 3 seconds
        sta    retry2        ;
        jsr    GetByte        ;
        bcc    Wait4CRC    ; wait for something to come in...
        cmp    #"C"        ; is it the "C" to start a CRC xfer?
        beq    SetstAddr    ; yes
        cmp    #ESC        ; is it a cancel? <Esc> Key
        bne    Wait4CRC    ; No, wait for another character
        jmp    PrtAbort    ; Print abort msg and exit
SetstAddr   ldy    #$00        ; init data block offset to 0
        ldx    #$04        ; preload X to Receive buffer
        lda    #$01        ; manually load blk number
        sta    Rbuff        ; into 1st byte
        lda    #$FE        ; load 1's comp of block #
        sta    Rbuff+1        ; into 2nd byte
        lda    ptr        ; load low byte of start address
        sta    Rbuff+2        ; into 3rd byte
        lda    ptrh        ; load hi byte of start address
        sta    Rbuff+3        ; into 4th byte
        jmp    Ldbuff1        ; jump into buffer load routine

LdBuffer    lda    Lastblk        ; Was the last block sent?
        beq    LdBuff0        ; no, send the next one
        jmp     Done        ; yes, we're done
```

```
LdBuff0     ldx    #$02        ; init pointers
        ldy    #$00        ;
        inc    Blkno        ; inc block counter
        lda    Blkno        ;
        sta    Rbuff        ; save in 1st byte of buffer
        eor    #$FF        ;
        sta    Rbuff+1        ; save 1's comp of blkno next

LdBuff1     lda    (ptr),y        ; save 128 bytes of data
        sta    Rbuff,x        ;
LdBuff2     sec            ;
        lda    eofp        ;
        sbc    ptr        ; Are we at the last address?
        bne    LdBuff4        ; no, inc pointer and continue
        lda    eofph        ;
        sbc    ptrh        ;
        bne    LdBuff4        ;
        inc    LastBlk        ; Yes, Set last byte flag
LdBuff3     inx            ;
        cpx    #$82        ; Are we at the end of the 128 byte block?
        beq    SCalcCRC    ; Yes, calc CRC
        lda    #$00        ; Fill rest of 128 bytes with $00
        sta    Rbuff,x        ;
        beq    LdBuff3        ; Branch always

LdBuff4     inc    ptr        ; Inc address pointer
        bne    LdBuff5        ;
        inc    ptrh        ;
LdBuff5     inx            ;
        cpx    #$82        ; last byte in block?
        bne    LdBuff1        ; no, get the next
SCalcCRC    jsr     CalcCRC
        lda    crch        ; save Hi byte of CRC to buffer
        sta    Rbuff,y        ;
        iny            ;
        lda    crc        ; save lo byte of CRC to buffer
        sta    Rbuff,y        ;
Resend      ldx    #$00        ;
        lda    #SOH
        jsr    Put_chr        ; send SOH
SendBlk     lda    Rbuff,x        ; Send 132 bytes in buffer to the console
        jsr    Put_chr        ;
        inx            ;
        cpx    #$84        ; last byte?
        bne    SendBlk        ; no, get next
        lda    #$FF        ; yes, set 3 second delay
        sta    retry2        ; and
        jsr    GetByte        ; Wait for Ack/Nack
        bcc    Seterror    ; No chr received after 3 seconds, resend
        cmp    #ACK        ; Chr received... is it:
        beq    LdBuffer    ; ACK, send next block
```

```
        cmp    #NAK        ;
        beq    Seterror    ; NAK, inc errors and resend
        cmp    #ESC        ;
        beq    PrtAbort    ; Esc pressed to abort
                    ; fall through to error counter
Seterror    inc    errcnt        ; Inc error counter
        lda    errcnt        ;
        cmp    #$0A        ; are there 10 errors? (Xmodem spec for failure)
        bne    Resend        ; no, resend block
PrtAbort    jsr    Flush        ; yes, too many errors, flush buffer,
        jmp    Print_Err    ; print error msg and exit
Done        Jmp    Print_Good    ; All Done..Print msg and exit
;
;
;

XModemRcv   jsr    PrintMsg    ; send prompt and info
        lda    #$01
        sta    blkno        ; set block # to 1
        sta    bflag        ; set flag to get address from block 1
StartCrc    lda    #"C"        ; "C" start with CRC mode
        jsr    Put_Chr        ; send it
        lda    #$FF
        sta    retry2        ; set loop counter for ~3 sec delay
        lda    #$00
                sta    crc
        sta    crch        ; init CRC value
        jsr    GetByte        ; wait for input
                bcs    GotByte        ; byte received, process it
        bcc    StartCrc    ; resend "C"

StartBlk    lda    #$FF        ;
        sta    retry2        ; set loop counter for ~3 sec delay
        jsr    GetByte        ; get first byte of block
        bcc    StartBlk    ; timed out, keep waiting...
GotByte     cmp    #ESC        ; quitting?
                bne GotByte1    ; no
;       lda    #$FE        ; Error code in "A" of desired
                brk         ; YES - do BRK or change to RTS if desired
GotByte1        cmp #SOH        ; start of block?
        beq    BegBlk        ; yes
        cmp    #EOT        ;
        bne    BadCrc        ; Not SOH or EOT, so flush buffer & send NAK
        jmp    RDone        ; EOT - all done!
BegBlk      ldx    #$00
GetBlk      lda    #$ff        ; 3 sec window to receive characters
        sta     retry2        ;
GetBlk1     jsr    GetByte        ; get next character
        bcc    BadCrc        ; chr rcv error, flush and send NAK
GetBlk2     sta    Rbuff,x        ; good char, save it in the rcv buffer
        inx            ; inc buffer pointer
```

```
        cpx    #$84        ; <01> <FE> <128 bytes> <CRCH> <CRCL>
        bne    GetBlk        ; get 132 characters
        ldx    #$00        ;
        lda    Rbuff,x        ; get block # from buffer
        cmp    blkno        ; compare to expected block #
        beq    GoodBlk1    ; matched!
        jsr    Print_Err    ; Unexpected block number - abort
        jsr    Flush        ; mismatched - flush buffer and then do BRK
;       lda    #$FD        ; put error code in "A" if desired
        brk            ; unexpected block # - fatal error - BRK or RTS
GoodBlk1    eor    #$ff        ; 1's comp of block #
        inx            ;
        cmp    Rbuff,x        ; compare with expected 1's comp of block #
        beq    GoodBlk2     ; matched!
        jsr    Print_Err    ; Unexpected block number - abort
        jsr     Flush        ; mismatched - flush buffer and then do BRK
;       lda    #$FC        ; put error code in "A" if desired
        brk            ; bad 1's comp of block#
GoodBlk2    jsr    CalcCRC        ; calc CRC
        lda    Rbuff,y        ; get hi CRC from buffer
        cmp    crch        ; compare to calculated hi CRC
        bne    BadCrc        ; bad crc, send NAK
        iny            ;
        lda    Rbuff,y        ; get lo CRC from buffer
        cmp    crc        ; compare to calculated lo CRC
        beq    GoodCrc        ; good CRC
BadCrc      jsr    Flush        ; flush the input port
        lda    #NAK        ;
        jsr    Put_Chr        ; send NAK to resend block
        jmp    StartBlk    ; start over, get the block again
GoodCrc     ldx    #$02        ;
        lda    blkno        ; get the block number
        cmp    #$01        ; 1st block?
        bne    CopyBlk        ; no, copy all 128 bytes
        lda    bflag        ; is it really block 1, not block 257, 513 etc.
        beq    CopyBlk        ; no, copy all 128 bytes
        lda    Rbuff,x        ; get target address from 1st 2 bytes of blk 1
        sta    ptr        ; save lo address
        inx            ;
        lda    Rbuff,x        ; get hi address
        sta    ptr+1        ; save it
        inx            ; point to first byte of data
        dec    bflag        ; set the flag so we won't get another address
CopyBlk     ldy    #$00        ; set offset to zero
CopyBlk3    lda    Rbuff,x        ; get data byte from buffer
        sta    (ptr),y        ; save to target
        inc    ptr        ; point to next address
        bne    CopyBlk4    ; did it step over page boundary?
        inc    ptr+1        ; adjust high address for page crossing
CopyBlk4    inx            ; point to next data byte
        cpx    #$82        ; is it the last byte
```

```
        bne    CopyBlk3    ; no, get the next one
IncBlk      inc    blkno        ; done.  Inc the block #
        lda    #ACK        ; send ACK
        jsr    Put_Chr        ;
        jmp    StartBlk    ; get next block

RDone       lda    #ACK        ; last block, send ACK and exit.
        jsr    Put_Chr        ;
        jsr    Flush        ; get leftover characters, if any
        jsr    Print_Good    ;
        rts            ;
;
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
;=========================================================================
;  I/O Device Specific Routines
;
;  Two routines are used to communicate with the I/O device.
;
; "Get_Chr" routine will scan the input port for a character.  It will
; return without waiting with the Carry flag CLEAR if no character is
; present or return with the Carry flag SET and the character in the "A"
; register if one was present.
;
; "Put_Chr" routine will write one byte to the output port.  Its alright
; if this routine waits for the port to be ready.  its assumed that the
; character was send upon return from this routine.
;
; Here is an example of the routines used for a standard 6551 ACIA.
; You would call the ACIA_Init prior to running the xmodem transfer
; routine.
;
ACIA_Data   =    $7F70        ; Adjust these addresses to point
ACIA_Status =    $7F71        ; to YOUR 6551!
ACIA_Command    =    $7F72        ;
ACIA_Control    =    $7F73        ;

ACIA_Init       lda    #$1F                ; 19.2K/8/1
                sta    ACIA_Control        ; control reg
                lda    #$0B                ; N parity/echo off/rx int off/
dtr active low
                sta    ACIA_Command        ; command reg
                rts                       ; done
;
; input chr from ACIA (no waiting)
;
Get_Chr     clc             ; no chr present
                lda    ACIA_Status     ; get Serial port status
                and    #$08            ; mask rcvr full bit
                beq    Get_Chr2    ; if not chr, done
                Lda    ACIA_Data       ; else get chr
               sec             ; and set the Carry Flag
```

```
Get_Chr2        rts             ; done
;
; output to OutPut Port
;
Put_Chr         PHA                     ; save registers
Put_Chr1         lda    ACIA_Status     ; serial port status
                 and    #$10            ; is tx buffer empty
                 beq    Put_Chr1        ; no, go back and test it again
                 PLA                     ; yes, get chr to send
                 sta    ACIA_Data       ; put character to Port
                 RTS                     ; done
;===============================================================================
;
; subroutines
;
;
;
GetByte     lda    #$00        ; wait for chr input and cycle timing loop
        sta    retry        ; set low value of timing loop
StartCrcLp  jsr    Get_chr        ; get chr from serial port, don't wait
        bcs    GetByte1    ; got one, so exit
        dec    retry        ; no character received, so dec counter
        bne    StartCrcLp    ;
        dec    retry2        ; dec hi byte of counter
        bne    StartCrcLp    ; look for character again
        clc             ; if loop times out, CLC, else SEC and return
GetByte1    rts            ; with character in "A"
;
Flush       lda    #$70        ; flush receive buffer
        sta    retry2        ; flush until empty for ~1 sec.
Flush1      jsr    GetByte        ; read the port
        bcs    Flush        ; if chr recvd, wait for another
        rts             ; else done
;
PrintMsg    ldx    #$00        ; PRINT starting message
PrtMsg1     lda       Msg,x
        beq    PrtMsg2
        jsr    Put_Chr
        inx
        bne    PrtMsg1
PrtMsg2     rts
Msg     .byte    "Begin XMODEM/CRC transfer.  Press <Esc> to abort..."
        .BYTE      CR, LF
                .byte   0
;
Print_Err   ldx    #$00        ; PRINT Error message
PrtErr1     lda       ErrMsg,x
        beq    PrtErr2
        jsr    Put_Chr
        inx
        bne    PrtErr1
```

```
PrtErr2     rts
ErrMsg      .byte     "Transfer Error!"
        .BYTE      CR, LF
                .byte   0
;
Print_Good  ldx    #$00        ; PRINT Good Transfer message
Prtgood1    lda       GoodMsg,x
        beq    Prtgood2
        jsr    Put_Chr
        inx
        bne    Prtgood1
Prtgood2    rts
GoodMsg     .byte    EOT,CR,LF,EOT,CR,LF,EOT,CR,LF,CR,LF
        .byte     "Transfer Successful!"
        .BYTE      CR, LF
                .byte   0



;
;
;=============================================================================
;
;
;  CRC subroutines
;
;
CalcCRC     lda    #$00        ; yes, calculate the CRC for the 128 bytes
        sta    crc        ;
        sta    crch        ;
        ldy    #$02        ;
CalcCRC1    lda    Rbuff,y        ;
        eor     crc+1         ; Quick CRC computation with lookup tables
            tax             ; updates the two bytes at crc & crc+1
            lda     crc        ; with the byte send in the "A" register
            eor     CRCHI,X
            sta     crc+1
             lda     CRCLO,X
            sta     crc
        iny            ;
        cpy    #$82        ; done yet?
        bne    CalcCRC1    ; no, get next
        rts            ; y=82 on exit
;
; Alternate solution is to build the two lookup tables at run-time.  This
might
; be desirable if the program is running from ram to reduce binary upload
time.
; The following code generates the data for the lookup tables.  You would
need to
; un-comment the variable declarations for crclo & crchi in the Tables and
Constants
```

```
; section above and call this routine to build the tables before calling the
; "xmodem" routine.
;
;MAKECRCTABLE
;       ldx     #$00
;       LDA    #$00
;zeroloop   sta     crclo,x
;       sta     crchi,x
;       inx
;       bne    zeroloop
;       ldx    #$00
;fetch      txa
;       eor    crchi,x
;       sta    crchi,x
;       ldy    #$08
;fetch1     asl    crclo,x
;       rol    crchi,x
;       bcc    fetch2
;       lda    crchi,x
;       eor    #$10
;       sta    crchi,x
;       lda    crclo,x
;       eor    #$21
;       sta    crclo,x
;fetch2     dey
;       bne    fetch1
;       inx
;       bne    fetch
;       rts
;
; The following tables are used to calculate the CRC for the 128 bytes
; in the xmodem data blocks.  You can use these tables if you plan to
; store this program in ROM.  If you choose to build them at run-time,
; then just delete them and define the two labels: crclo & crchi.
;
; low byte CRC lookup table (should be page aligned)
        *= $FD00
crclo
 .byte $00,$21,$42,$63,$84,$A5,$C6,$E7,$08,$29,$4A,$6B,$8C,$AD,$CE,$EF
 .byte $31,$10,$73,$52,$B5,$94,$F7,$D6,$39,$18,$7B,$5A,$BD,$9C,$FF,$DE
 .byte $62,$43,$20,$01,$E6,$C7,$A4,$85,$6A,$4B,$28,$09,$EE,$CF,$AC,$8D
 .byte $53,$72,$11,$30,$D7,$F6,$95,$B4,$5B,$7A,$19,$38,$DF,$FE,$9D,$BC
 .byte $C4,$E5,$86,$A7,$40,$61,$02,$23,$CC,$ED,$8E,$AF,$48,$69,$0A,$2B
 .byte $F5,$D4,$B7,$96,$71,$50,$33,$12,$FD,$DC,$BF,$9E,$79,$58,$3B,$1A
 .byte $A6,$87,$E4,$C5,$22,$03,$60,$41,$AE,$8F,$EC,$CD,$2A,$0B,$68,$49
 .byte $97,$B6,$D5,$F4,$13,$32,$51,$70,$9F,$BE,$DD,$FC,$1B,$3A,$59,$78
 .byte $88,$A9,$CA,$EB,$0C,$2D,$4E,$6F,$80,$A1,$C2,$E3,$04,$25,$46,$67
 .byte $B9,$98,$FB,$DA,$3D,$1C,$7F,$5E,$B1,$90,$F3,$D2,$35,$14,$77,$56
 .byte $EA,$CB,$A8,$89,$6E,$4F,$2C,$0D,$E2,$C3,$A0,$81,$66,$47,$24,$05
 .byte $DB,$FA,$99,$B8,$5F,$7E,$1D,$3C,$D3,$F2,$91,$B0,$57,$76,$15,$34
 .byte $4C,$6D,$0E,$2F,$C8,$E9,$8A,$AB,$44,$65,$06,$27,$C0,$E1,$82,$A3
```

```
 .byte $7D,$5C,$3F,$1E,$F9,$D8,$BB,$9A,$75,$54,$37,$16,$F1,$D0,$B3,$92
 .byte $2E,$0F,$6C,$4D,$AA,$8B,$E8,$C9,$26,$07,$64,$45,$A2,$83,$E0,$C1
 .byte $1F,$3E,$5D,$7C,$9B,$BA,$D9,$F8,$17,$36,$55,$74,$93,$B2,$D1,$F0

; hi byte CRC lookup table (should be page aligned)
        *= $FE00
crchi
 .byte $00,$10,$20,$30,$40,$50,$60,$70,$81,$91,$A1,$B1,$C1,$D1,$E1,$F1
 .byte $12,$02,$32,$22,$52,$42,$72,$62,$93,$83,$B3,$A3,$D3,$C3,$F3,$E3
 .byte $24,$34,$04,$14,$64,$74,$44,$54,$A5,$B5,$85,$95,$E5,$F5,$C5,$D5
 .byte $36,$26,$16,$06,$76,$66,$56,$46,$B7,$A7,$97,$87,$F7,$E7,$D7,$C7
 .byte $48,$58,$68,$78,$08,$18,$28,$38,$C9,$D9,$E9,$F9,$89,$99,$A9,$B9
 .byte $5A,$4A,$7A,$6A,$1A,$0A,$3A,$2A,$DB,$CB,$FB,$EB,$9B,$8B,$BB,$AB
 .byte $6C,$7C,$4C,$5C,$2C,$3C,$0C,$1C,$ED,$FD,$CD,$DD,$AD,$BD,$8D,$9D
 .byte $7E,$6E,$5E,$4E,$3E,$2E,$1E,$0E,$FF,$EF,$DF,$CF,$BF,$AF,$9F,$8F
 .byte $91,$81,$B1,$A1,$D1,$C1,$F1,$E1,$10,$00,$30,$20,$50,$40,$70,$60
 .byte $83,$93,$A3,$B3,$C3,$D3,$E3,$F3,$02,$12,$22,$32,$42,$52,$62,$72
 .byte $B5,$A5,$95,$85,$F5,$E5,$D5,$C5,$34,$24,$14,$04,$74,$64,$54,$44
 .byte $A7,$B7,$87,$97,$E7,$F7,$C7,$D7,$26,$36,$06,$16,$66,$76,$46,$56
 .byte $D9,$C9,$F9,$E9,$99,$89,$B9,$A9,$58,$48,$78,$68,$18,$08,$38,$28
 .byte $CB,$DB,$EB,$FB,$8B,$9B,$AB,$BB,$4A,$5A,$6A,$7A,$0A,$1A,$2A,$3A
 .byte $FD,$ED,$DD,$CD,$BD,$AD,$9D,$8D,$7C,$6C,$5C,$4C,$3C,$2C,$1C,$0C
 .byte $EF,$FF,$CF,$DF,$AF,$BF,$8F,$9F,$6E,$7E,$4E,$5E,$2E,$3E,$0E,$1E
;
;
; End of File
;
```