```
PIXpander beta 0.1 Source
;
; PIXpand beta 0.1 by Sami Khawam (c) 1999-2001.
;
;  http://sami.ticalc.org
;
;
; This is the source code of the beta version of the PIXpand memory
; expander for TI calculator. I am releasing this so that people
; can get a better idea on how the expander works from inside and
; possibily be able to debug it.
;
; This code is not well commented, but should enough to see how it
; is working. Many thing could be further optimised like using the
; interrupt to detect signal changes on the lines on the memory card,
; etc...
;


;
; TODO:
;
; - When receiving, the user should always confirm the (over)write.
;   When a new 32k block is going to be opened user should confirm, but
;    this time the LED flashes faster.
;    This occures when a 1) signle file is > 32k  <-- double confirm (auto)
;                        2) in middle of multi files, when going over
;                           the 32k block;
;
; - When sending, send first a tiny copy of the 1st program in the session
;   with the same name, then the real program so that the user can choose
;   to overwrite.
;   NOT with backups!
;   Receiving Session is left open until the device is reset from the battery.
;   or until keypress.
;
; -When error in transmission, make the transmission repeat.
;
;
; - In the TI89, when going to the VAR-Link menu, a commande is sent. Ignore it!!!!
;
;
; - When reading pages, make the calc return a flag in case of errors
;




                device  pic16f84,xt_osc,wdt_on,protect_off,pwrt_on


;******Variables definitions

                org     0Ch

; Variables definitions. Not sure if all of them are used below; need to be checked

Counter         ds      1               ; 0C
Counter_        ds      1
Counter2        ds      1
Counter3        ds      1
BtnCount        ds      1
BitCount        ds      1
Stat            ds      1
OChar           ds      1
IChar           ds      1
w_copy          ds      1
s_copy          ds      1
AddrH           ds      1
AddrL           ds      1
AddrH_          ds      1
AddrL_          ds      1
CharTemp        ds      1
CharTemp2       ds      1
OCharT          ds      1
ICharT          ds      1
FrameCounter    ds      1
CalcType        ds      1
CalcCommand     ds      1
nByteL          ds      1
nByteH          ds      1
xorcode         ds      1
VarType         ds      1
Parts           ds      1
Temp            ds      1
Temp2           ds      1
Temp3           ds      1
ITemp           ds      1
FlashTimer      ds      1
RBCopy          ds      1
nRetry          ds      1
Buffer          ds      30
```

PIXpander beta 0.1 Source

```
;******Constants definitions

Bit_K           =       220             ; Used for various delays
HBit_K          =       16
RTCC_K          =       160
D_20            =       20

; Constant for OPTION.
OptionWaiting   =       00001111b       ; RTCC: internal

; Constant for INTERUPPT
IntSending      =       00100000b       ; RTCC: On, RB0: Off, GIE: On


;Constants for the in/out porst
RA_Direction    =       00010110b
RA_W_In         =       00010110b       ; Used to make the White open
RA_W_Out        =       00010100b       ; drain line using only one pin
RA_Init         =       00001001b

RB_Direction    =       10010000b
RB_Init         =       11100000b       ; CMD, SEL, and CLK high

RO              =       RA.3
RI              =       RA.2
WI              =       RA.1            ; WO and WI are the same thing
WO              =       RA.1



RED_LED         =       RB.0            ; The 4 LEDs
RED_LED4        =       RB.3
RED_LED3        =       RB.2
RED_LED2        =       RB.1
RED_LED1        =       RB.0

DATA            =       RB.4            ; Memory card lines
CMD             =       RB.5
SEL             =       RB.6
ACK             =       RB.7
CLK             =       RA.0

BTN             =       RA.4            ; Push-button

StopWait        =       Stat.0          ; Various flag. TODO: Remove unused ones
Wait            =       Stat.1
LEDSweep        =       Stat.2
StartWait       =       Stat.3
NoAck           =       Stat.4
Backup          =       Stat.5
NoNewFrame      =       Stat.6
FirstConfirm    =       Stat.7

;Calc Commands
OK_CMD          =       56h
READY_CMD       =       09h
ERROR_CMD       =       36h
SKIP_CMD        =       02h
STOP_CMD        =       01h
EOT_CMD         =       92h

;XPand Commands
SETADDR         =       0F0h
GETADDR         =       0F1h
WRITE_PAGE      =       0F2h
READ_PAGE       =       0F3h
WRITE_DATA      =       0F4h
READ_DATA       =       0F5h


;***********Program Begin

                org     0
                jmp     Start


;***********Interrupt
                org     4
IntHandler                                      ; The interrupt controls the flashing on the LEDs
                mov     w_copy,w                ; Make a copy of w.
                mov     s_copy,STATUS           ; Make a copy of status.

                clrb    T0IF                    ; Clear timer interrupt flag.


                djnz    FlashTimer, :End

                jnb     LEDSweep, :End          ; Make it sweep at everry byte sent/received
                clrb    LEDSweep
```

```
PIXpander beta 0.1 Source
;                mov     w, RB
                 mov     w, RBCopy
                 and     w, #11110000b
                 mov     ITemp, w

                 clc
                 mov     w, << RBCopy
                 and     w, #00001111b
                 snz
                 mov     w, #00000001b

                 or      w, ITemp
                 mov     RBCopy, w
                 mov     RB, w

:End
                 mov     STATUS,s_copy   ; Restore status register
                 swap    w_copy          ; Prepare for swapped move.
                 mov     w,<>w_copy      ; Swap/move to w, status unaffected.
                 reti                    ; return, GIE set

;************Program

Start
                 mov     RA, #RA_Init            ; Set high: RO, WO, TX
                 mov     RB, #RB_Init
                 mov     !RA, #RA_Direction      ; Set port.
                 mov     !RB, #RB_Direction

                 ; Repeated setting of the register. Check if really needed
                 mov     RA, #RA_Init            ; Set high: RO, WO, TX
                 mov     RB, #RB_Init
                 mov     RBCopy, w


:Re
                 clr     IChar
                 clr     Stat                    ; Clear all status.
                 clr     WDT
                 mov     !OPTION, #OptionWaiting
                 mov     INTCON, #IntSending      ;Set up interrupt.


;                 setb    RED_LED                 ; Make a small delay
;                 clr     Counter2
;                 clr     Counter3
;::NoActive       call    Bit_Delay
;                 clr     WDT
;                 mov     !OPTION, #OptionWaiting
;                 call    Delay20ms
;                 djnz    Counter2, :NoActive
;                 djnz    Counter3, :NoActive
;                 clrb    RED_LED

                 clr     AddrL
                 clr     AddrH
                 call    SetLED

; ************************************************
;  Main loop: Wait until a button is pressed (Card->Calc)
;             or until the calc is sending something (Calc->Card)
; ************************************************

Main_loop
                 clr     wdt                     ; Clear WDT so that it does not reset.
                 mov     !OPTION, #OptionWaiting

                 jb      BTN, CheckBtn
TICheck
                 jb      RI, :TT
                 jb      WI, Recv                ; Jump if one is
                 jmp     Main_loop               ;  low, but not boths.
:TT              jb      WI, Main_loop           ; If not true we contrinue down
                                                 ;  to Recv




; ************************************************
;
; Receive from Calc -> Memory Card
; ************************************************

Recv

                 clrb    FirstConfirm

                 mov     FSR, #Buffer

; Add support for CBL modules ?
                 call    GetTI                           ; Get Calc type
                 mov     CalcType, OChar
(c) 1999-2001 Sami Khawam, http://sami.ticalc.org
```

```
PIXpander beta 0.1 Source
                mov     INDF, OChar                     ; Save in buffer
                inc     FSR
                cje     OChar, #READ_PAGE, ReadPages

                cja     OChar, #0A9h, :ChkCmd           ; Not valid calc
                cja     OChar, #80h, :valid_calc_type   ; 80-A9
                cja     OChar, #10h, :error             ; Not valid 10-80
:valid_calc_type
                call    GetTI                           ; Get Command
                mov     CalcCommand, OChar
                mov     INDF, OChar                     ; Save in buffer
                inc     FSR
                cjne    OChar, #06h, :error             ; If no data header.

;Take all byte, but only save what begins with a valid calc and 0x06 command

                call    SetLED

                mov     AddrL_, AddrL           ; Save WAddr
                mov     AddrH_, AddrH

:NextVar
                mov     Parts, #1               ; A backup has 3 parts
                clrb    Backup


; Get the header data and save it in RAM. Only after confirmation write
; it on the memory card.

                call    GetTI                   ; Get nByteL
                mov     INDF, OChar             ; Save in buffer
                inc     FSR
                mov     nByteL, OChar           ; Number of bytes in header (Low)

                call    GetTI                   ; No header is larger than 255 bytes
                                                ;  -> Ignore that byte (High)

                mov     Counter2, #3
:sendnext_
                call    GetTI                   ; Get 3 bytes to get the VarType
                mov     INDF, OChar             ; Save in buffer
                inc     FSR
                mov     VarType, OChar
                djnz    Counter2, :sendnext_

;Optimisation TODO: Instead of Vartype, use OCharT directely

                cje     VarType, #1Dh, :Backup  ; Jmp if backup
                cjne    VarType, #0Fh, :noback1 ; Jmp if not backup
:Backup
                mov     Parts, #3               ; Backup has 3 parts
                setb    Backup
:noback1

                dec     nByteL                  ; Decrement to make it get the checksum
                                                ;   ( 3-1 = 2), which is not included in nByteL
                mov     Counter2, nByteL
:sendnext
                call    GetTI                   ; Get rest of bytes and
                mov     INDF, OChar             ; Save in buffer
                inc     FSR
                djnz    Counter2, :sendnext

                mov     CalcCommand, #OK_CMD
                call    PutCmd

; Save all the above data in the RAM, then only after confirmation write it.
; Check if error (free space), and see if the 32k barrier is exceeded.
; If yes make a double confirmation.

                sb      FirstConfirm
                call    Confirm                 ; Confirm overwrite
                setb    FirstConfirm


                call    SetLED                  ; Turn LED's on when writing the
                                                ; file from RAM to card

                setb    NoNewFrame              ; Initialize new frame


                mov     Temp, FSR
                mov     FSR, #Buffer
:save_next
                mov     OChar, INDF             ; Write Calc-Type
                call    WriteByte
                inc     FSR
                cjne    FSR, Temp, :save_next   ; Break if we arrive at last address



                call    ClrLED                  ; Clear LEDs.
```

PIXpander beta 0.1 Source

```
                setb    GIE                     ; Set the LEDs swap on.

                mov     CalcCommand, #READY_CMD         ; If no error
                call    PutCmd

                call    Get4B

:NextPart
                call    TI2Mem          ; Calc-Type
                call    TI2Mem          ; Command. = 0x15

                call    TI2Mem
                mov     nByteL, OCharT

                call    TI2Mem
                mov     nByteH, OCharT


                mov     Counter2, nByteL
                mov     Counter3, nByteH
                inc     Counter3
:sendnext2
                call    TI2Mem

                djnz    Counter2, :sendnext2
                djnz    Counter3, :sendnext2


                call    TI2Mem                  ; Checksum
                call    TI2Mem                  ; Checksum


                mov     CalcCommand, #OK_CMD    ;PutOK if no error
                call    PutCmd

                djnz    Parts, :NextPart        ; Do all parts(->backup)

                jb      Backup, :EndSession


:WaitForEOT
                mov     FSR, #Buffer

                call    GetTI                   ; Get Calc type
                mov     CalcType, OChar
                mov     INDF, OChar             ; Save in buffer
                inc     FSR

                call    GetTI                   ; Get Command
                mov     CalcCommand, OChar      ; 0x92 =  EOT
                mov     INDF, OChar             ; Save in buffer
                inc     FSR

                cjne    OChar, #EOT_CMD, :NextVar     ; If no EOT.

                call    GetTI                   ; Zero
                call    GetTI                   ; Zero

                mov     CalcCommand, #OK_CMD    ;PutOK if to stop
                call    PutCmd

:EndSession

                clr     OChar
                call    WriteByte               ; Write only 0x00 to END


;Write 00s to end current transmission of frame (if not at end of frame).
;If exacly at end, add a frame full of 00s.

:fillframe
                jb      NoNewFrame, :end
                clr     OChar
                call    WriteByte
                jmp     :fillframe

:end
                setb    SEL

                call    ClrLED
                call    Start_delay
                call    SetLED

                mov     AddrL, AddrL_           ; Restore WAddr
                mov     AddrH, AddrH_
                jmp     Main_loop

:ChkCmd



;If errors, put address where it was.
```

```
PIXpander beta 0.1 Source
:error
;                jb      NoNewFrame, :fillframe_
;                clr     OChar
;                call    WriteByte
;                jmp     :error
:fillframe_
                mov     AddrL, AddrL_           ; Restore WAddr
                mov     AddrH, AddrH_
   ;Optimise the following lines
                call    ClrLED
                mov     RB, #RB_Init
                setb    RED_LED1
                setb    RED_LED2
                mov     RBCopy, RB
                jmp     Start:Re                ;Main_loop




; ************************************************
; Several funcitons to control the LEDs and read the
; push-button
; ************************************************

Confirm
        mov     Temp, #210
        call    ClrLED
:wait1  jb      BTN, :checkbtn_
        clr     WDT
        mov     !OPTION, #OptionWaiting
        call    Bit_Delay
        djnz    Temp, :wait1
        call    SetLED
:wait2  jb      BTN, :checkbtn_
        clr     WDT
        mov     !OPTION, #OptionWaiting
        call    Bit_Delay
        djnz    Temp, :wait2
        jmp     Confirm
:checkbtn_
        call    Delay20ms
        call    Delay20ms
        jnb     BTN, Confirm          ; If glitch
        call    SetLED
        ret


ClrLED
        clrb    GIE                     ; Set the LEDs swap off.
        jb      GIE, $-1
        mov     w, RBCopy
        and     w, #11110000b
        mov     RBCopy, w
        mov     RB, w

        ret

SetLED
UpdateDisplay
        mov     w, AddrH
        and     w, #00000011b           ; Get first 2 bits
        mov     Temp3, w
        inc     Temp3
        clr     Temp2
        setc
:rot
        rl      Temp2
        djnz    Temp3,:rot
        mov     w, RBCopy
        and     w, #11110000b
        or      w, Temp2
        mov     RB, w                   ; Update display
        mov     RBCopy, w
        ret

; ************************************************
; Check if the button was pusshed (not a glitch) and
; send the variable to the calc in this case.
; ************************************************

CheckBtn
        mov     BtnCount, #225
        call    Delay20ms
        call    Delay20ms
        jnb     BTN, Main_loop          ; If glitch
;        mov     RB, #RB_Init
:CountTime
        call    Delay20ms
        inc     BtnCount
        jz      :SendF                  ; If more than 255-245 = 10
        jb      BTN, :CountTime
```

```
PIXpander beta 0.1 Source
:End
        mov     w, AddrH
        and     w, #11111100b           ;
        mov     AddrL, w                ; Save higher bits
        inc     AddrH                   ; Increment by 32k
        and     AddrH, #00000011b
        or      AddrH, AddrL            ; Stay in the same 1MBit always.
        clr     AddrL                   ; Clear posisiton in current 32k
        call     UpdateDisplay
        jmp     Main_loop

:SendF
        call    UpdateDisplay
:wait   clr     wdt
        jb      BTN, :wait
        mov     !OPTION, #OptionWaiting

                                        ; Continue down to Send



; ************************************************
; Send variables from the card to the calc
; ************************************************
Send
;                clrb    SEL

                mov     AddrL_, AddrL           ; Save WAddr
                mov     AddrH_, AddrH

                setb    NoNewFrame              ; Initialize new frame
                call    ReadByte                ; Read Byte
:Test_calc_type
                mov     CalcType, IChar
;                cja     IChar, #0A9h, :error             ; Not valid calc
;                cja     IChar, #80h, :valid_calc_type   ; 80-A9
;                cja     IChar, #10h, :error             ; Not valid 10-80
:valid_calc_type
                call    ReadByte
                mov     CalcCommand, IChar
;                cjne    IChar, #06h, :error             ; If no data header.

;Take all byte, but only send what begins with a valid calc and 0x06 command

                setb    GIE                     ; Set the LEDs swap on.

;                call    SetLED

:NextVar
                mov     Parts, #1
                clrb    Backup

                mov     IChar, CalcType         ; Calc-Type
                call    PutTI
                mov     IChar, CalcCommand      ; Command. 0x06
                call    PutTI

                call    Mem2TI
                mov     nByteL, ICharT
                clr     IChar                   ; Second size byte = 0
                call    PutTI

      ;Optimise: use nByteL as counter directely
                mov     Counter2, #3
:sendnext_

                call    Mem2TI
                mov     VarType, ICharT
                djnz    Counter2, :sendnext_

      ;Optimise: Instead of Vartype, use ICharT directely
                cje     VarType, #1Dh, :Backup          ; Jpm if backup
                cjne    VarType, #0Fh, :noback1         ; Jpm if not backup
:Backup
                mov     Parts, #3
                setb    Backup
:noback1

;                sub     nByteL, #3
                dec     nByteL                          ; Just -1 to include checksum bytes
                mov     Counter2, nByteL
:sendnext
                call    Mem2TI
                djnz    Counter2, :sendnext

;                call    Mem2TI          ; Checksum        we used -1 not -3 before
;                call    Mem2TI          ; Checksum


                call    Get4B

                jmp     :CalcByte
(c) 1999-2001 Sami Khawam, http://sami.ticalc.org
```

PIXpander beta 0.1 Source

```
:wait_calc
                clr     Counter2
                clr     Counter3
;                call    ClrLED
:wait1          jnb     RI, :CalcByte
                jnb     WI, :CalcByte
                clr     WDT
                mov     !OPTION, #OptionWaiting
                djnz    Counter2, :wait1
;                djnz    Counter3, :wait1
;                call    SetLED
:wait2          jnb     RI, :CalcByte
                jnb     WI, :CalcByte
                clr     WDT
                mov     !OPTION, #OptionWaiting
                djnz    Counter2, :wait2
;                djnz    Counter3, :wait2
                jmp     :wait_calc

                clr     WDT
                mov     !OPTION, #OptionWaiting

:CalcByte
                call    GetTI                           ; Calc-Type
                call    GetTI                           ; Command
                mov     CalcCommand, OChar
                call    GetTI
                call    GetTI

                cjne    CalcCommand, #ERROR_CMD, :noerror
                call    GetTI                           ; If error
                call    GetTI                           ; Command
                mov     CalcCommand, OChar
                call    GetTI
                cje     CalcCommand, #STOP_CMD, :error
                cjne    CalcCommand, #SKIP_CMD, :error
                mov     CalcCommand, #OK_CMD            ; If skip
                call    PutCmd
                jmp     :error

:noerror        mov     CalcCommand, #OK_CMD            ; If no error
                call    PutCmd


:NextPart
                call    Mem2TI          ; Calc-Type
                call    Mem2TI          ; Command. = 0x15

                call    Mem2TI
                mov     nByteL, ICharT

                call    Mem2TI
                mov     nByteH, ICharT

                mov     Counter2, nByteL
                mov     Counter3, nByteH
                inc     Counter3
:sendnext2
                call    Mem2TI
                djnz    Counter2, :sendnext2
                djnz    Counter3, :sendnext2

                call    Mem2TI           ; Checksum
                call    Mem2TI           ; Checksum

                call    Get4B

                djnz    Parts, :NextPart          ; Do all parts(->backup)

                jb     Backup, :EndSession

:SendEOT
                call    ReadByte
                test    IChar
                jnz     :Test_calc_type           ; zero means finished

                mov     CalcCommand, #EOT_CMD     ; Put EOT to stop
                call    PutCmd

                call    Get4B                     ; Get OK
:EndSession

; Read 00s to end current transmission of frame (if not at end of frame).


:end
                call    ReadByte
                jnb     NoNewFrame, :end

                setb    SEL
```

```
PIXpander beta 0.1 Source
                mov     AddrL, AddrL_           ; Restore Addr
                mov     AddrH, AddrH_

                call    ClrLED
                call    Start_delay
                call    SetLED

                jmp     Main_loop

:error
                call    ReadByte
                jnb     NoNewFrame, :error

                ;If errors, put address where it was and write 00s.
                mov     AddrL, AddrL_           ; Restore Addr
                mov     AddrH, AddrH_
                call    ClrLED
                mov     RB, #RB_Init
                setb    RED_LED2
                setb    RED_LED3
                mov     RBCopy, RB
                jmp     Start:Re                ;Main_loop



; ************************************************
; Used to dump the content of the card to the calc or PC
;
; ************************************************

ReadPages

                call    ClrLED
                setb    GIE                     ; Set the LEDs swap on.

                mov     AddrL_, AddrL           ; Save WAddr
                mov     AddrH_, AddrH

                setb    NoNewFrame              ; Initialize new frame

                call    GetTI
                mov     Counter3, Ochar

;                clrb    SEL

                        mov     IChar, #'X'     ; For Debug, mark a new frame
                        call    PutTI           ; For debug

:nextpage
;                       clrb    SEL

;                       setb    NoAck
;                       mov     OChar, #81h
;                       call    SendRcv
;;                      clrb    NoAck
;                       setb    NoAck
;                       mov     OChar, #'R'             ; Read-Command  = 'R'
;                       call    SendRcv


                mov     Counter2, #127
;                mov     Counter2, #200
:nextbyte
                call    Mem2TI
;                       clr     OChar
;                       call    SendRcv
                djnz    Counter2, :nextbyte
;                   setb    SEL
;;                  call    Start_delay             ;       A small delay.
;                call    Delay20ms               ; 20ms delay
;                call    Delay20ms               ; 20ms delay
                djnz    Counter3, :nextpage

                setb    SEL

                jmp     Send:end


; TODO: Seperate EndFrame and NextFrame to make ending the frame correctly.
; ************************************************
;
WriteByte
                clr     wdt
                mov     !OPTION, #OptionWaiting

                jb      NoNewFrame, :NewFrame

                xor     xorcode, OChar          ; checksum

                call    SendRcv
```

```
                djnz    FrameCounter, :end

                call    Start_delay             ;       A small delay.

:FrameEnd
                mov     OChar, xorcode
                call    SendRcv

                clr     OChar
                call    SendRcv
                cjne    IChar, #5Ch, :Error

                clr     OChar
                call    SendRcv
                cjne    IChar, #5Dh, :Error

                clr     OChar
                call    SendRcv
                cjne    IChar, #47h, :Error


                call    Start_delay             ;       A small delay.

                setb    SEL
                call    Delay20ms               ; 20ms delay
                clrb    SEL
                call    Start_delay             ;       A small delay.

:NextFrame
                inc     AddrL
                snz                             ; Skip if no overflow
                inc     AddrH
                setb    NoNewFrame

:end            ret

:NewFrame
                mov     OCharT, OChar           ; Save OChar

                mov     OChar, #81h
                call    SendRcv

                mov     OChar, #57h             ; Write-Command
                call    SendRcv

                clr     OChar
                call    SendRcv
                cjne    IChar, #5Ah, :Error

                clr     OChar
                call    SendRcv
                cjne    IChar, #5Dh, :Error

                mov     OChar, AddrH   ;addr.H
                call    SendRcv

                mov     OChar, AddrL   ;addr.L
                call    SendRcv

                mov     w, AddrL
                xor     w, AddrH              ;!!!!!!!!!!!!!!
                mov     xorcode, w

                mov     FrameCounter, #128

                clrb    NoNewFrame
                mov     OChar, OCharT           ; Restore OChar
                jmp     WriteByte

:Error
                clrb    WO
                clrb    RO
                setb    SEL
                call    Start_Delay
                call    ClrLED
                mov     RB, #RB_Init
                mov     RA, #RA_Init
                setb    RED_LED2
                setb    RED_LED4
;                jmp     Main_loop
                jmp     Start:Re




; TODO: Seperate EndFrame and NextFrame.
; ***********************************************
;
ReadByte
                clr     wdt
                mov     !OPTION, #OptionWaiting
```

PIXpander beta 0.1 Source

```
                mov     nRetry, #8

                jb      NoNewFrame, :NewFrame

                clr     OChar
                call    SendRcv

                xor     xorcode, IChar          ; checksum

                djnz    FrameCounter, :end

                mov     ICharT, IChar           ; Save IChar

;                call    Start_delay             ;        A small delay.

:FrameEnd
                clr     OChar
                call    SendRcv                  ; Get and check xor code.
                cjne    IChar, xorcode, :Error
                                ;; Add a NoAck hre if ACK is going to be used
                setb    NoAck

                clr     OChar
                call    SendRcv
                cjne    IChar, #'G', :Error      ; Code for : 'G'


;                call    Start_delay             ;        A small delay.
                setb    SEL
                call    Delay20ms               ; 20ms delay
                call    Delay20ms               ; 20ms delay
                call    Delay20ms               ; 20ms delay
                call    Delay20ms               ; 20ms delay

:NextFrame
                inc     AddrL
                snz                             ; Skip if no overflow
                inc     AddrH
                setb    NoNewFrame

                mov     IChar, ICharT           ; Restore IChar

:end            ret

:retry_nf

                setb    SEL
                call    Delay20ms               ; 20ms delay
                call    Delay20ms               ; 20ms delay
                call    Delay20ms               ; 20ms delay
                call    Delay20ms               ; 20ms delay
                call    Delay20ms               ; 20ms delay

                djnz    nRetry, :NewFrame
                jmp     :Error

:NewFrame

                clrb    SEL
                call    Start_delay             ;        A small delay.
;                call    Delay20ms               ; 20ms delay

;                   clrb    NoAck
                    setb    NoAck
                mov     OChar, #81h
                call    SendRcv
;                   clrb    NoAck
                    setb    NoAck

                mov     OChar, #'R'             ; Read-Command  = 'R'
                call    SendRcv

                clr     OChar
                call    SendRcv
;                cjne    IChar, #'Z', :Error        ; 'Z'
                cjne    IChar, #'Z', :retry_nf

                clr     OChar
                call    SendRcv
                cjne    IChar, #']', :Error     ; ']'

                mov     OChar, AddrH
                call    SendRcv

                mov     OChar, AddrL
                call    SendRcv

                clr     OChar
                call    SendRcv
                cjne    IChar, #5Ch, :Error     ; '\'
```

```
PIXpander beta 0.1 Source
                clr     OChar
                call    SendRcv
                test    IChar                   ; On some cards there is a: 0x5C 0x00 0x5D response
                jnz     :test_IChar
                clr     OChar
                call    SendRcv                 ; Read the 0x5D, if there was a 0 previously
:test_IChar       test  IChar
                  jz    :Error
                ;  cjne    IChar, #5Dh, :Error

                clr     OChar
                call    SendRcv
                cjne    IChar, AddrH, :Error

                clr     OChar
                call    SendRcv
                cjne    IChar, AddrL, :Error

                mov     w, AddrL
                xor     w, AddrH
                mov     xorcode, w

                mov     FrameCounter, #128

                clrb    NoNewFrame
                call    Start_delay             ;        A small delay.
                jmp     ReadByte

:Error
                clrb    WO
                clrb    RO
                setb    SEL
                call    Start_Delay
                call    ClrLED
                mov     RA, #RA_Init
                mov     RB, #RB_Init
                setb    RED_LED1
                setb    RED_LED3

                        mov     IChar, #'e'     ; For Debug
                        call    PutTI           ; For debug

;                jmp     Main_loop
                jmp     Start:Re



;
; ************************************************
;

SendRcv                         ; Send OChar, get IChar

                clrb    GIE                     ; Set the LEDs swap off.
                jb      GIE, $-1

                clr     IChar
                mov     BitCount, #8

:xmit           clrb    CLK             ; Falling-Edge

                clc
                rr      OChar
                movb    CMD, c

;                jmp        $+1
;                jmp        $+1
;                jmp        $+1
                jmp     $+1
                jmp     $+1
                jmp     $+1
                jmp     $+1
                jmp     $+1
                jmp     $+1

                setb    CLK

                clc
                movb    c,DATA          ; Read DATA on the rising edge
                rr      IChar

;                jmp        $+1
;                jmp        $+1
;                jmp        $+1
                jmp     $+1
                jmp     $+1
                jmp     $+1
                jmp     $+1
                jmp     $+1
                jmp     $+1

                djnz    BitCount,:xmit
(c) 1999-2001 Sami Khawam, http://sami.ticalc.org
```

```
                setb    CMD
                setb    CLK


;                   jb      NoAck, :snd

;                   jb      ACK, $


:snd
                    setb    GIE




;******Delay for 1/2 Bit
Start_delay     mov     Counter, #HBit_K
:Loop           nop
                djnz    Counter, :Loop
                ret


;******Delay for 1 Bit
Bit_delay       mov     Counter, #Bit_K
:Loop           nop
                djnz    Counter, :Loop
                ret

;******Delay for 20ms
Delay20ms       mov     Counter_, #D_20
:Loop2
                clr     Counter
:Loop1          clr     wdt
                djnz    Counter, :Loop1
                djnz    Counter_, :Loop2
                mov     !OPTION, #OptionWaiting
                ret

PutCmd
                mov     IChar, CalcType
                call    PutTI
                mov     IChar, CalcCommand
                call    PutTI
                clr     IChar
                call    PutTI
                clr     IChar
                call    PutTI
                ret


;*****
;* Get 4 bytes from calc
;*****
Get4B           mov     Counter, #4
:loop           call    GetTI
                djnz    Counter, :Loop
                ret

; ************************************************
; * Get a byte from the calc and save it on the memory card
; ************************************************
TI2Mem
                call    GetTI
                mov     OCharT, OChar
                call    WriteByte
                ret

; ************************************************
; * Get a byte from the memory card and send it to the calc
; ************************************************
Mem2TI
                call    ReadByte
                mov     ICharT, IChar
                call    PutTI
                ret

;*********
;* Get a byte from the calc and store it in Char.
;*********
GetTI           clr     OChar
                mov     BitCount, #8

:GetLoop1       jb      WI, :GetLoop2           ;  Wait until R or W goes low,
                jnb     RI, :GetLoop1           ;  but not both at the same time.

:GetZero        clrb    RO                      ;   Get 0
                clc                             ;
                rr      OChar                   ;
                jnb     WI, $                   ;
                setb    RO                      ;
```

```
                jnb     RI, $                   ;
                djnz    BitCount, :GetLoop1     ;
                setb    LEDSweep
                ret

:GetLoop2       jb      RI, :GetLoop1           ;  not both at the same time.

:GetOne         clrb    WO                      ;   Get 1
                mov     !RA, #RA_W_Out          ; Set Direction.
                clrb    WO
                stc                             ;
                rr      OChar                   ;
                jnb     RI, $                   ;
                mov     !RA, #RA_W_In           ; Set Direction.
;                setb    WO                      ;
                jnb     WI, $                   ;
                djnz    BitCount, :GetLoop1     ;
                setb    LEDSweep
                ret

;********* End of GetTI



;*********
;* Send the byte stored in Char to the calc
;*********
PutTI           mov     BitCount, #8

:Wait
                mov     !RA, #RA_W_In           ; Set Direction.
;                setb    WO                      ;
                setb    RO                      ;
                jnb     WI, :Wait               ; Wait until R and W
                jnb     RI, :Wait               ; are high.

:PutLoop1       clc
                rr      IChar
                jc      :PutOne

:PutZero        clrb    WO                      ;   Put Zero
                mov     !RA, #RA_W_Out          ; Set Direction.
                clrb    WO
                jb      WI, $                   ;
                jb      RI, $                   ;
                mov     !RA, #RA_W_In           ; Set Direction.
;                setb    WO                      ;
                jnb     WI, $                   ;
                jnb     RI, $                   ;
                djnz    BitCount, :PutLoop1     ;
                setb    LEDSweep

                ret

:PutOne         clrb    RO                      ;
                jb      RI, $                   ;
                jb      WI, $                   ;   Put One
                setb    RO                      ;
                jnb     RI, $                   ;
                jnb     WI, $                   ;
                djnz    BitCount, :PutLoop1     ;
                setb    LEDSweep

                ret

;********* End of PutTI
```