PBUS+, A flexible network for factory control

Richard Walker

2/2/09

Abstract

Modern process control systems require reliable and flexible mechanisms for connecting peripherals to a central control computer. Component changes and upgrades should be easily accommodated. A simple high-speed serial network based on RS485[1] technology is proposed. A Master/Slave protocol gives each subsystem in the network a unique address. The Master addresses data registers in the Slave nodes to either read process information or to update control parameters. The physical layer of the proposed network is differential CAT5 twisted-pair cable to provide high immunity to electrical and magnetic interference. The protocol includes checksum protection, acknowledgments and retry to achieve highly reliable communication. A reference slave node implementation is presented using a low cost Peripheral Interface Controller (PIC) chip for less than $5/node.

[1] http://en.wikipedia.org/wiki/EIA-485

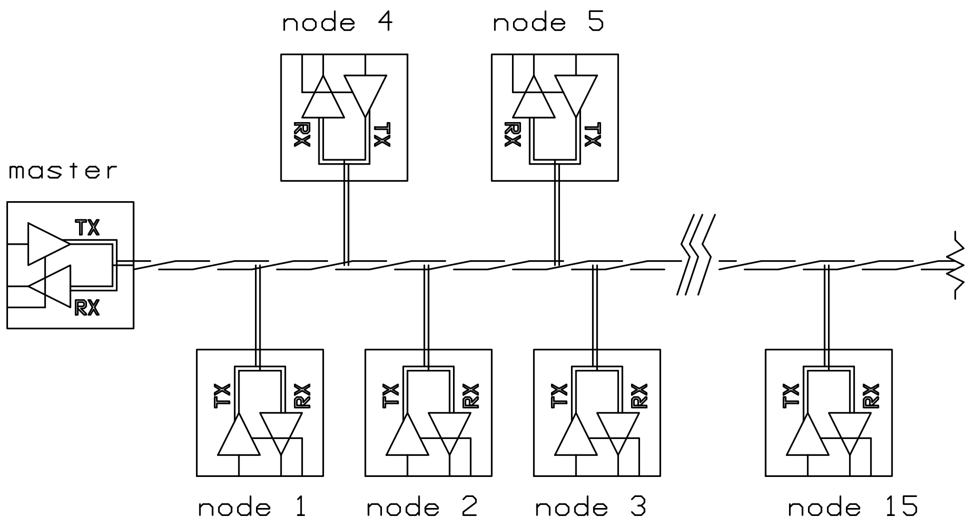


Figure 1: A PBUS+ network showing a master node and multiple slaves

0.1 Network Architecture

Card-cages are often used to interconnect hardware subsystems. Card cage systems, such as Multibus, have several drawbacks:

1. A high cost is incurred per function due to the large board size and the interconnect logic for each board.
2. A large amount of cabling must typically be run between the process sensors/actuators and the central card cage.
3. Functions are limited by a fixed board size.

To overcome these problems, a high speed serial network is proposed. Control subsystems are "daisy-chained" as passive drops on a CAT-5 twisted-pair cable. Each subsystem may be independently designed to be any size needed. A Microchip Peripheral Interface Controller (PIC)[2] runs firmware to implement a packet-based RS485 data communication protocol. Since PIC chips have between 12 and 36 general purpose analog and digital I/O lines, a typical slave node will consist of nothing more than the network interface chip itself plus some interface logic to allow the PIC to directly control relays, read voltages, etc. Each slave node communicates with master control processor via a defined packet structure. This simple interface allows any block to be replaced, as technology improves, with any new system capable of providing an equivalent control function and of responding to the standard interface.

0.1.1 Physical Layer

Figure 0.1 shows the topology of a PBUS+ network. Up to 15 slave units may be daisy-chained on a CAT-5 twisted pair cable. The signalling is fully differential and provides high rejection of electromagnetic interference. Four twisted pairs are available in CAT-5 cable, so two extra pairs are used for network power transmission and a shield ground. Since the network itself provides power, it is possible run the communication bus interface off the network supply and opto-isolate it from the controlled system's power supply. This ensures that no ground loops are created between subsystems through the interconnect network.

0.1.2 Packet Structure

PBUS+ is an RS485-like multi-drop bus for interconnecting PIC and other microcontroller-driven devices. It is derived from the published PBUS[3] protocol and source code by Peter Jakob. The line coding used is standard RS232 with 9 bits, no parity and 1 stop bit. The data rate is application dependent, but may be as high as 112 kbaud.

[2] http://www.microchip.com
[3] http://jap.hu/electronic/pbus.html

PBUS+ differs from PBUS in that it uses a 9th bit for header delimiting and takes advantage of the Microchip PICs ability to interrupt on 9th bit set. This reduces the processor overhead in slave nodes. Instead of every node reading all packets, each node sets its Universal Asynchronous Receiver/Transmitter (UART) to interrupt the processor only when a character is received with the 9th bit set high. All other characters are simply ignored. The master node sets the 9th bit on the first byte of each packet which contains the slave address and byte count for the packet. When a character is received with the 9th bit is set, the slave processor is interrupted and the character is checked to see if the address matches the node ID number. If so, then 8-bit reception is enabled and the entire packed is processed. If not, then the character is discarded and the slave processor returns to background processing.

PBUS+ assigns a unique device ID in the range 1-f for every controller on the bus. The maximum number of slave devices on one bus is 15. Current protocol is one master, multiple slaves. Packets consist of 8-bit bytes. All packets contain at least 3 bytes. Maximum length is 18 bytes.

byte offset	bits	name	description
0	8	ADDFLAG	9-th bit is set high ONLY on address bytes
0	7-4	DEVID	destination device PBUS ID
0	3-0	LEN	data length (0 means 3 byte packet)
1	7-0	CMD	MASTER command or SLAVE response code
2	7-0	DATA[]	optional data (LEN=0 means no data)
2+LEN	7-0	CKSUM	packet checksum

0.1.3 Standard Commands

All PBUS+ nodes implement the following commands. In addition, each node may define other commands as needed. In the table below "M" stands for Master and "S" stands for Slave.

Node	Code	Parameters	Description
M	5e	CVER	(check node version)
S	60	ROK <version #><type>	
M	5f	CPING <optional data>	test if node is present
S	6f	RECHO <echoed data>	
M	58	CNOOP	(no operation)
S	60	ROK	
M	5b	CLAST	(repeat last response)
S	??	(duplicate of last packet)	
M	5c	CRES	reset statistics counters
S	60	ROK	
M	5d	CSTAT	(get statistics)
S	60	ROK<E1><E2><E3>	

The CSTAT command returns three bytes to report various statistics gathered by the node since the last CRES reset command. <E1> is the number of checksum errors on packets addressed to this node. <E2> is the total number of packets headers seen on the network. <E3> is the number of packets addressedto this node received with a good checksum.

0.2 Reference Implementation

To test the protocol, a copy of the PBUS library was obtained and modified to support the 9th bit address flag. The source code for PBUS+ is in Microchip assembly language and is structured as a library which is linked with a user application. All the commands listed above are implemented in the core library code and a stub is provided for the user to add any further commands as desired.

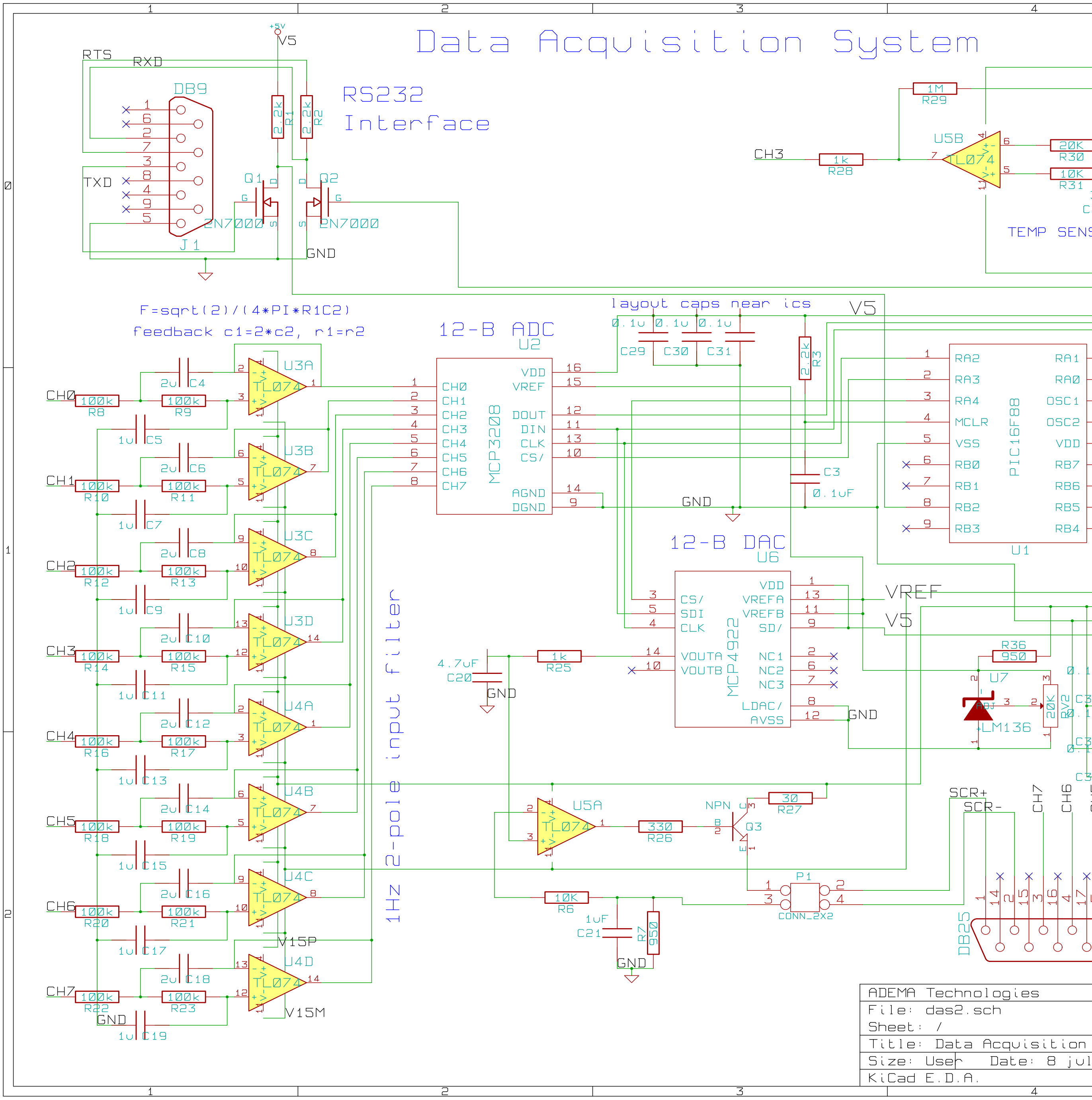
The PBUS+ application was tested using a Data Acquisition System (DAS) as the test vehicle. The DAS was designed to attach to the SCR control board of an Czochralski Crystal Growth furnace and to monitor and log several key process variables. The reference application only tests the packet communication mechanism. The RS485 bus interface circuitry was not included in this initial test. However the differential drive circuits are well known and very low risk.

The monitored process variables are described in the following table.

process variable	description
PHV A-B	Secondary Voltage across phases A,B
PHB B-C	Secondary Voltage across phases B,C
PHV C-A	Secondary Voltage across phases C,A
CTI A	Primary Current, phase A
CTI B	Primary Current, phase B
CTI C	Primary Current, phase C
TEMP	IR temperature
HV	DC Heater Voltage

In addition to monitoring 8 process variables, the DAS circuitry was designed to be capable of driving the current loop control input for the three phase SCR controller. Although not tested at the time of this report, the SCR control capability allows fully testing the PBUS+/RS485 strategy in a closed-loop KVA or Temperature control system.

0.2.1 Schematic



The DAS consists of 7 key modules: 1) Microchip CPU, 2) Lowpass filters for input signal processing, 3) ADC converter chip, 4) DAC converter chip, 5) Communication interface circuitry, 6) Temperature sensing circuit, 7) SCR drive circuit. Each will be discussed, in turn.

The CPU is a Microchip P16F88 which is a flash programmable controller with 3 counter-timers, 15 pins of analog/digitial I/O, 7 ADC channels, a UART, a PWM channel, and an SPI serial communication module. Not all these functions are needed in this application, but it is convenient to have more resources than necessary when doing development. This chip retails for $5.00 in single quantity, and provides a fully programmable core for implementing a PBUS+ node.

The lowpass filters are implemented with two TL072 opamps arranged in a 2nd order 1Hz lowpass filter. Preliminary testing showed that the process signals are highly contaminated with switching noise and that reliable measurements were not possible without excessive averaging unless an LPF was used.

The ADC and DAC converters are both 12-bit accurate. They are controlled through a four-wire SPI protocol using a CHIP_SELECT, DATA_IN, DATA_OUT and a CLOCK line. The DATA_IN, DATA_OUT and CLOCK are shared by both chips, and the desired chip is enabled for communication using the CHIP_SELECT line. The control of the 8-channel ADC and two channel DAC only requires 4 I/O lines on the processor.

For this prototype a simple RS232 interface is used as the physical layer. The packet protocol is PBUS+. The disadvantage of an RS232 interface is that only one peripheral can be controlled by the master computer. RS232 is not capable of multidrop. This capability may be explored in a future experiment. The RS232 interface requires two pins to interface with the processor. An RS485 driver would require one additional pin to control the state of the TX driver on the interface chip.

The temperature sensing and SCR drive circuits are duplicates of the circuits currently used in the Czochralski furnace controller. They are implemented with two sections of a TLO74 opamp. The output and input of the circuits directly connect to ADC and DAC inputs and require no further resources from the CPU.

0.2.2 Firmware Listing

Library Firmware Code (pbus88.asm)

```
;****************************************************************
;
;     Filename:       pbus88.asm
;     Description:    rs485-like multi-drop bus
;                     with half-duplex serial protocol
;                     one master, max 15 slaves
;
;     Author:         el@jap.hu:   http://jap.hu/electronic/
;     Modified:       walker@omnisterra.com: added 9-bit headers
;                     to reduce interrupt load on slaves.  Increased
;                     data rate to 19.2 kbaud.
;
;****************************************************************
;     Notes:
;
; To use the interrupt handler, SAVE these registers:
; W, STATUS, FSR (if used in main program!)
; TMR1 is reserved for RX/TX scheduling
;
;****************************************************************

        EXTERN pbus_handler
        GLOBAL pbus_init, pbus_int_handler
        GLOBAL rxe1cnt, rxe2cnt, rxe3cnt
        GLOBAL rxbuf, txbuf

         list p=16f88

        #include <p16f88.inc>
         errorlevel      1,-(305)

.pbusda UDATA
```

```
;***** VARIABLE DEFINITIONS

        GLOBAL bytecnt

bytecnt res 1   ; serial routine flags
chksum  res 1   ; buffer checksum
rxe1cnt res 2   ; rx checksum error counter
rxe2cnt res 2   ; rx all packets counter
rxe3cnt res 2   ; rx my packets counter

; don't change buffer order!
rxbuf   res .19 ; receive buffer
txbuf   res .19 ; transmit buffer
rmax    equ .19 ; size of rxbuf
tmax    equ .19 ; size of txbuf

; * CONSTANTS
#define ROK      0x60
#define RECHO    0x6f
#define DIRBIT   PORTB, 3                 ; rs485 dir 1=RX

#define PBUS_VER 0x88
#define MAIN_VER 0x20

devid           equ 0x50                  ; devid 0 is master

;f_osc          equ .4000000
f_osc           equ .20000000
ser_baudrate    equ .19200
ser_baud        equ (f_osc/(ser_baudrate*.16)) - 1

;bittime        equ (.1000000/ser_baudrate)
bittime         equ .52                   ; 1/(19.2 kbaud) = 52usec

rxdelay         equ .10 * bittime         ; delay to switch from  TX to RX
                                          ; rxdelay: STOP + START + 8data

txdelay         equ .10 * bittime         ; delay to switch from  RX to TX
                                          ; txdelay: starts at end of RX STOP

maxdelay        equ .4900                 ; max delay in packet between bytes
                                          ; if this is not big enough you may
                                          ; have problems with the time delay of
                                          ; the tcsetattr() call that switches
                                          ; the polarity of 9th bit in master
t1prescaler     equ .8
t1rxdelay       equ .65536 - ((f_osc / .4000000)*(rxdelay  / t1prescaler))
t1txdelay       equ .65536 - ((f_osc / .4000000)*(txdelay  / t1prescaler))
t1maxdelay      equ .65536 - ((f_osc / .4000000)*(maxdelay / t1prescaler))

;---------------------------------------------------------------------------
.pbuslib        CODE

pbus_cmd_vectors
                addlw   pbus_cmd
                movwf   PCL                       ; command jump table
```

```
pbus_cmd        goto    cnoop           ; 58: noop
                goto    rxb_ena         ; 59: illegal
                goto    rxb_ena         ; 5a: illegal
                goto    clast           ; 5b: clast
                goto    cres            ; 5c: cres
                goto    cstat           ; 5d: cstat
                goto    cver            ; 5e: cver
                goto    cping           ; 5f: cping

pbus_init       clrf    bytecnt         ; clear all BER counters
                clrf    rxe1cnt
                clrf    rxe1cnt+1
                clrf    rxe2cnt
                clrf    rxe2cnt+1
                clrf    rxe3cnt
                clrf    rxe3cnt+1

                banksel FSR
                movlw   rxbuf           ; init rxbuf and enable rx
                movwf   FSR
                clrf    chksum

                ; init serial, enable RX itr
                BANKSEL SPBRG           ; init serial, enable RX itr
                movlw   ser_baud
                movwf   SPBRG           ; baud rate register

                BANKSEL RCSTA
                movlw   b'11010000'     ; SPEN:7 RX9:6 CREN:4 ADDEN:3 (9BIT)
                movwf   RCSTA           ; pins for serial
                bsf     RCSTA,ADDEN     ; turn on address recognition

                banksel TXSTA
                movlw   b'01100100'     ; TX9:6 TXEN:5, SYNC:4 BRGH:2
                movwf   TXSTA

                banksel RCREG
                movf    RCREG,W         ; flush receive buffer
                movf    RCREG,W
                movf    RCREG,W

                banksel T1CON
                movlw   b'00110000'     ; 1:8 prescale
                movwf   T1CON           ; init TMR1

                banksel PIE1
                bsf     PIE1, RCIE      ; enable interrupt on rx
                bcf     PIE1, TXIE      ; disable interrupt on tx
                bsf     PIE1, TMR1IE    ; enable TMR1 interrupt

                banksel TMR1L
                movlw   low t1maxdelay
                movwf   TMR1L
                movlw   high t1maxdelay
                movwf   TMR1H
                banksel T1CON
                bsf     T1CON, TMR1ON ; enable TMR1
```

```
                banksel PIR1
                bcf     PIR1,TMR1IF

                movlw   (1<<GIE)+(1<<PEIE)
                movwf    INTCON
                return

;**************************************************************
; Interrupt handler
;**************************************************************
pbus_int_handler                        ; interrupt handler

                movlw   0xff            ; toggle port A for debug
                xorwf   PORTA,f

                BANKSEL PIR1
                btfsc   PIR1, TMR1IF
                  goto  tmrint          ; TMR1 overflow

                btfsc   PIR1, RCIF
                  call  rxint           ; RX buffer full

                btfss   PIR1, TXIF
                  return                ; TX buffer still full
                                        ; fall through if clear

                btfsc   RCSTA, CREN
                   return               ; in RX mode!
                ;goto   txint

;**************************************************************
; TX timeout (falls through from above)
;**************************************************************
txint           movf    bytecnt, W
                bz      txbdone

                comf    chksum, W
                addlw   1               ; load W with -chksum
                decfsz  bytecnt, F      ; if not last byte
                  movf  INDF, W         ; transmit a byte from packet
                                        ; else send chksum

                movwf   TXREG           ; TXIF cleared writing TXREG
                addwf   chksum, F
                incf    FSR, F
                return

txbdone         ; TX done (no more bytes in bytecnt)
                ; automatically schedule a TX-> RX switch - rxdelay

                BANKSEL PIE1
                bcf     PIE1, TXIE ; disable TX itr:TXIF endless loop

                BANKSEL txbuf
                bcf     T1CON, TMR1ON
                bcf     PIR1, TMR1IF
```

```
                movlw   low t1rxdelay
                movwf   TMR1L
                movlw   high t1rxdelay
                movwf   TMR1H
                bsf     T1CON, TMR1ON ; enable TMR1
                return

;***********************************************************
; TMR1 timeout
;***********************************************************

tmrint          ; timer int
                ; decide if a receive timeout/switch (goto rxb_ena)
                ; or a transmit switch

                bcf PIR1, TMR1IF        ; clear TMR1 itr flag

                movlw   txbuf           ; if FSR points at txbuf
                subwf   FSR, W          ; goto sched_tx
                bz      sched_tx

rxb_ena         bcf     DIRBIT          ; set RS485 port direction
                                        ; SN75176 driver is RX active low
                bsf     RCSTA, CREN     ; enable receiver
                bsf     RCSTA, ADDEN    ; bit 9 interrupt mask
                bcf     T1CON, TMR1ON   ; disable TMR1

                movlw   rxbuf
                movwf   FSR             ; init rxbuf and enable rx
                clrf    bytecnt
                return

;***********************************************************
sched_tx        bcf     RCSTA, CREN     ; disable RX
                bcf     T1CON, TMR1ON   ; disable TMR1
                BANKSEL PIE1
                bsf     PIE1, TXIE      ; schedule TX itr
                BANKSEL txbuf
                return

;***********************************************************
; RX byte received
;***********************************************************

rxint           ; a byte was received, set timeout again
                banksel T1CON
                bcf T1CON, TMR1ON       ; disable TMR1
                movlw   low t1maxdelay
                movwf   TMR1L
                movlw   high t1maxdelay
                movwf   TMR1H
                bsf     T1CON, TMR1ON   ; enable TMR1

                btfss   RCSTA,OERR
                  goto  rx_frame
                bcf     RCSTA,CREN      ; when overrun, uart will stop rx
                bsf     RCSTA,CREN      ; and CREN must be reset
```

```
rx_frame        movf    bytecnt, W
                  bnz   rx_2

                incf    rxe2cnt, F      ; count all headers
                btfsc   STATUS, Z
                  incf  rxe2cnt+1, F

                movf    RCREG, W        ; first byte
                movwf   rxbuf           ; save it

                movlw   0xf0            ; check devid for match
                andwf   rxbuf, W
                sublw   devid           ; FIXME?  check for addr 0 here?
                btfss   STATUS, Z
                  goto  rxb_ena         ; devid mismatch, restart

                bcf     RCSTA,ADDEN     ; open up bit 9 for rest of pkt
                movf    rxbuf, W
                movwf   chksum          ; initialize chksum
                movlw   0x0f            ; MASK bytecnt
                andwf   rxbuf, W
                addlw   3               ; account for header, cmd, cksum bytes
                movwf   bytecnt         ; store lsnybble as bytecnt
                goto    rx_3

rx_2            movf    RCREG, W        ; middle bytes
                movwf   INDF            ; simply accumulate in buffer
                addwf   chksum, F       ; and update chksum

rx_3            incf    FSR, F          ; point at next spot in buffer
                decfsz  bytecnt, F
                  return                ; packet still incomplete

rxb_done        ; packet received

                movf    chksum, W               ; check packet
                btfsc   STATUS, Z
                  goto  skip_6

                incf    rxe1cnt, F              ; 16 bit packet chksum error
                btfsc   STATUS, Z
                  incf  rxe1cnt+1, F
                goto    rxb_ena                 ; restart with new packet


skip_6          bsf     RCSTA, ADDEN    ; turn on address recognition

;************** Command handlers start here
; illegal commands with no response goto rxb_ena (restart with a new packet)
; good commands with resp. in txbuf goto txb_ena (TX a packet with checksum)
;**************

handle_cmd      ; got a good packet

                incf    rxe3cnt, F              ; keep stats
                btfsc   STATUS, Z
```

```
                  incf    rxe3cnt+1, F

                movlw   0xf8            ; check for higher 5 bits
                andwf   (rxbuf+1), W
                sublw   0x58
                btfss   STATUS, Z
                  goto  cext            ; not internal

                movlw   0x07
                andwf   (rxbuf+1), W
                goto    pbus_cmd_vectors

cping           movlw   RECHO
                movwf   txbuf+1         ; response code

                movlw   0x0f            ; mask off byte count
                andwf   rxbuf, W
                movwf   txbuf
                movwf   chksum          ; use as counter
                bz      txb_ena         ; data present?

                movlw   rxbuf+2         ; copy data to response
                movwf   FSR

                ; basically we increment FSR and add/subtract
                ; rmax (size of rxbuffer) to toggle between
                ; rxbuffer and txbuffer

cpingcopy       movf    INDF, W
                movwf   rxbuf+1         ; use as temp storage
                movlw   rmax
                addwf   FSR, F          ; goto transmit buffer
                movf    rxbuf+1, W
                movwf   INDF

                movlw   rmax
                subwf   FSR, F          ; goto receive buffer
                incf    FSR, F
                decfsz  chksum, F
                goto    cpingcopy

                goto    txb_ena

cver            movlw   2               ; num data bytes
                movwf   txbuf
                movlw   ROK             ; respond code
                movwf   txbuf+1
                movlw   PBUS_VER        ; first byte
                movwf   txbuf+2
                movlw   MAIN_VER        ; second byte
                movwf   txbuf+3
                goto    txb_ena

cstat           movlw   6
                movwf   txbuf
                movlw   ROK
                movwf   txbuf+1
```

```
                movf    rxe1cnt+1, W
                movwf   txbuf+2
                movf    rxe1cnt, W
                movwf   txbuf+3
                movf    rxe2cnt+1, W
                movwf   txbuf+4
                movf    rxe2cnt, W
                movwf   txbuf+5
                movf    rxe3cnt+1, W
                movwf   txbuf+6
                movf    rxe3cnt, W
                movwf   txbuf+7
                goto    txb_ena

cres            clrf    rxe1cnt
                clrf    rxe1cnt+1
                clrf    rxe2cnt
                clrf    rxe2cnt+1
                clrf    rxe3cnt
                clrf    rxe3cnt+1

cnoop           movlw   0
                movwf   txbuf
                movlw   ROK
                movwf   rxbuf+1
                goto    txb_ena

cext            movf    (rxbuf+1), W
                call    pbus_handler
                andlw   0xff            ; if at return W=0, respond
                btfss   STATUS, Z
                  goto  rxb_ena         ; otherwise, go back listening

clast           ; repeat last response (in the transmit buffer)

txb_ena         ; schedule a RX->TX switch after RXB to answer - txdelay

                bsf     DIRBIT          ; set RS485 port direction

                bcf     T1CON, TMR1ON ; disable TMR1
                movlw   low t1txdelay
                movwf   TMR1L
                movlw   high t1txdelay
                movwf   TMR1H
                bsf     T1CON, TMR1ON

                movlw   txbuf           ; get address of TX buffer
                movwf   FSR             ; set indirect register
                clrf    chksum
                movlw   0x0f
                andwf   txbuf, W        ; get data bytecount
                addlw   0x03            ; make it a packet count
                movwf   bytecnt         ; save it
                return

        END                             ; directive 'end of program''
```

Application Firmware Code (main88.asm)

The application code includes SPI routines for talking to the DAC and ADC converters and also implements two new user commands: SET and GET.

SET takes two bytes as an argument, treats them together as a 16-bit word and then uses the 12 least significant bits to set the DAC output for controlling the SCR power level. The GET command returns 12 hexadecimal byte values which are interpreted as eight packed 12-bit words, each word corresponding to the value of one of the 8 ADC channels.

```
;*********************************************************************
;   main routine for RS-485-like pbus node
;   data aqcuistion system
;
;*********************************************************************

        list       p=16F88              ; list directive to define processor
        __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _LVP_OFF & _CPD_OFF

#include <p16f88.inc>
#include "pbus88.inc"

freemem UDATA

;***** VARIABLES

w_save          RES 1           ; interrupt w save
s_save          RES 1           ; interrupt status save
temp            RES 1
channel         RES 1           ; tmp for doadc and dodac
adc_hi          RES 1
adc_lo          RES 1
dac_hi          RES 1
dac_lo          RES 1
spidata         RES 1           ; spi routine i/o buffer
count           RES 1           ; for spi routine
command         RES 1

#define         GET     0x10    ; get ADC value command
#define         SET     0x11    ; set DAC value command

vectors         CODE 0
                clrf    PCLATH
                goto    init            ; go to beginning of program
                nop
                nop
                goto    int_handler

int_handler     movwf   w_save
                swapf   STATUS, W
                clrf    STATUS
                movwf   s_save
                call    pbus_int_handler
                swapf s_save, W
                movwf STATUS
                swapf w_save, F
                swapf w_save, W
                retfie
```

```
init            banksel STATUS
                bcf     STATUS,RP0      ; select bank 0
                bcf     STATUS,RP1
                clrf    PORTA
                clrf    PORTB

                banksel TRISA
                movlw   b'00000001'     ; A(0) is SDI, 1:4 all outputs
                movwf   TRISA
                movlw   b'11110101'     ; RB2 RXin=1, RB5 TXout=1
                movwf   TRISB

                banksel ADCON0
                movlw   b'00000000'     ; ADC off
                movwf   ADCON0

                banksel ANSEL
                movlw   b'00000000'     ; all bits digital
                movwf   ANSEL

                call pbus_init
                bsf     PORTA,3         ; set CS/ high for dac,adc
                bsf     PORTA,4

main            nop
                goto main

;----------------------------------------------------------------------
; custom pbus commands
; we handle SET and GET commands here.
;----------------------------------------------------------------------

pbus_handler    ; arrives here w/cmd in W

                movwf   command         ; save the command
                sublw   GET             ; compare with SET command
                btfsc   STATUS,Z
                  goto  getcmd

                movf    command,w       ; get it back again
                sublw   SET             ; compare with SET command
                btfsc   STATUS,Z
                  goto  setcmd

                ; test as many CMDS as you want here ....

                retlw   0xff            ; didn't match any command

                ; getcmd packs 8 12-bit ADC conversion values into 12 bytes
                ; nothing fancy, just unwrapped code...

getcmd          movlw   .12             ; number of data bytes in response
                movwf   txbuf
                movlw   ROK
                movwf   txbuf+.1

                movlw   .0              ; ADC 0
```

```
call    doadc

movf    adc_hi,w
andlw   0x0f            ; isolate first nybble
movwf   txbuf+.2        ; move to first data reg
movf    adc_lo,w
andlw   0xf0            ; isolate hi nybble
iorwf   txbuf+.2,f
swapf   txbuf+.2,f      ; adjust nybble placement

movf    adc_lo,w
andlw   0x0f            ; isolate lo nybble
movwf   txbuf+.3        ; move to second data reg

movlw   .1              ; ADC 1
call    doadc

swapf   adc_hi,w
andlw   0xf0            ; isolate first nybble
iorwf   txbuf+.3,f      ; composite into reg
swapf   txbuf+.3,f      ; adjust nybble placement
movf    adc_lo,w
movwf   txbuf+.4        ; needs no twiddling

movlw   .2              ; ADC 2
call    doadc

movf    adc_hi,w
andlw   0x0f            ; isolate first nybble
movwf   txbuf+.5        ; move to first data reg
movf    adc_lo,w
andlw   0xf0            ; isolate hi nybble
iorwf   txbuf+.5,f
swapf   txbuf+.5,f      ; adjust nybble placement

movf    adc_lo,w
andlw   0x0f            ; isolate lo nybble
movwf   txbuf+.6        ; move to second data reg

movlw   .3              ; ADC 3
call    doadc

swapf   adc_hi,w
andlw   0xf0            ; isolate first nybble
iorwf   txbuf+.6,f      ; composite into reg
swapf   txbuf+.6,f      ; adjust nybble placement
movf    adc_lo,w
movwf   txbuf+.7        ; needs no twiddling

movlw   .4              ; ADC 4
call    doadc

movf    adc_hi,w
andlw   0x0f            ; isolate first nybble
movwf   txbuf+.8        ; move to first data reg
movf    adc_lo,w
andlw   0xf0            ; isolate hi nybble
```

```
                iorwf   txbuf+.8,f
                swapf   txbuf+.8,f      ; adjust nybble placement

                movf    adc_lo,w
                andlw   0x0f            ; isolate lo nybble
                movwf   txbuf+.9        ; move to second data reg

                movlw   .5              ; ADC 5
                call    doadc

                swapf   adc_hi,w
                andlw   0xf0            ; isolate first nybble
                iorwf   txbuf+.9,f      ; composite into reg
                swapf   txbuf+.9,f      ; adjust nybble placement
                movf    adc_lo,w
                movwf   txbuf+.10       ; needs no twiddling

                movlw   .6              ; ADC 6
                call    doadc

                movf    adc_hi,w
                andlw   0x0f            ; isolate first nybble
                movwf   txbuf+.11       ; move to first data reg
                movf    adc_lo,w
                andlw   0xf0            ; isolate hi nybble
                iorwf   txbuf+.11,f
                swapf   txbuf+.11,f     ; adjust nybble placement

                movf    adc_lo,w
                andlw   0x0f            ; isolate lo nybble
                movwf   txbuf+.12       ; move to second data reg

                movlw   .7              ; ADC 7
                call    doadc

                swapf   adc_hi,w
                andlw   0xf0            ; isolate first nybble
                iorwf   txbuf+.12,f     ; composite into reg
                swapf   txbuf+.12,f     ; adjust nybble placement
                movf    adc_lo,w
                movwf   txbuf+.13       ; needs no twiddling

                retlw   0x00

setcmd          movlw   0x0f            ; mask off byte count
                andwf   rxbuf, W
                sublw   0x02            ; set CMD must have two bytes
                btfss   STATUS,Z
                  goto  seterr
setok           movlw   ROK
                movwf   txbuf+1
                movf    rxbuf+2,w
                movwf   dac_hi
                movf    rxbuf+3,w
                movwf   dac_lo
                movlw   0               ; first DAC channel
                call    dodac
```

```
                goto     setdone

seterr          movlw    RBAD
                movwf    txbuf+1

setdone         clrf     txbuf            ; 0=addr, 0=cnt
                retlw    0x00

;------------------------------------------------
; debugging stub to return known adc values
;------------------------------------------------
doadc2          movwf    adc_hi
                addlw    0x01
                movwf    adc_lo
                return

;-----------------------------------------------------------------
; dodac
; called with channel number in w: (0,1)
; called with 12 bit value in: dac_hi, dac_lo
; uses spidata=SPI_RW(spidata)
; drives PORTA,CSD,SDO appropriately
;-----------------------------------------------------------------

dodac:  banksel PORTA
        bcf     PORTA,CSD          ; chip select/ DAC low

        andlw   0x01               ; mask channel in W
        movwf   channel            ; save channel #
                                   ; if 1, write dacB, if 0 write dacA
        movf    dac_hi,w
        andlw   0x0f               ; mask off high bits
        iorlw   b'00110000'        ; unbuffered, 1x-gain, power-on

        btfsc   channel,0
         iorlw  b'10000000'        ; set channel bit

        movwf   spidata
        call    SPI_RW             ; discard read-in

        movf    dac_lo,w
        movwf   spidata
        call    SPI_RW

        bsf     PORTA,CSD          ; chip select/ DAC hi
        return

;-----------------------------------------------------------------
; doadc
; called with channel number in w (0-7)
; uses spidata=SPI_RW(spidata)
; drives PORTA,CSA,SDO appropriately
; returns: adc_lo, adc_hi
;-----------------------------------------------------------------

doadc:  banksel PORTA
        bcf     PORTA,CSA          ; chip select/ ADC low
```

```
        andlw    0x07              ; mask channel in W
        movwf    channel           ; save channel #

        movlw    b'00000110'       ; pad[5],start,single,ch[2]
        btfsc    channel,2
          iorlw  b'00000001'       ; set msb of channel #

        movwf    spidata
        call     SPI_RW            ; discard read-in

        movlw    b'00000000'       ; ch[1], ch[0], pad[6]
        btfsc    channel,1
          iorlw  b'10000000'       ; set msb of channel #
        btfsc    channel,0
          iorlw  b'01000000'       ; set lsb of channel #

        movwf    spidata
        call     SPI_RW
        movf     spidata,w
        ANDLW    0x0F              ; mask off high byte
        movwf    adc_hi            ; save hi byte of conversion

        movlw    b'00000000'       ; TX data is a don't care
        movwf    spidata
        call     SPI_RW
        movf     spidata,w
        movwf    adc_lo            ; save lo byte of conversion

        bsf      PORTA,CSA         ; chip select/ ADC hi
        return

;-------------------------------------------------------------------
; SPI Routines
; globals: COUNT, DATA_OUT, DATA_IN
; assumes PORTA.[SDI,SDO,SCK]
; ------------------------------------------------------------------

SDI     equ     0       ; data into PIC
SDO     equ     1       ; data out from PIC
SCK     equ     2       ; clock
CSA     equ     3       ; chip select/ ADC
CSD     equ     4       ; chip select/ DAC

;-------------------------------------------------------------------
; SPI_RW: clock in/out byte in series, MSB first
; entry: datum to send in spidata,
; exit:  datum received in spidata,  count=0
;-------------------------------------------------------------------

SPI_RW:
        banksel PORTA
        bcf      PORTA,SCK         ; start clock low
        movlw    d'8'              ; init loop counter
        movwf    count

SPRWLP:
```

```
        bcf      PORTA,SCK          ; set clock lo
        ;------------------------------------------------
        ; TX bit setup
        ;------------------------------------------------
        bcf      PORTA,SDO          ; zero data bit
        btfsc    spidata,7          ; skip if MSB zero
        bsf      PORTA,SDO          ; else make data 1

        bsf      PORTA,SCK          ; set clock hi
        ;------------------------------------------------
        ; RX bit read
        ;------------------------------------------------
        bcf      STATUS,C           ; zero C flag
        rlf      spidata,f          ; shift C and datum left
        btfsc    PORTA,SDI          ; skip if SDI=0
        bsf      spidata,0          ; else set bit0 to 1

        decfsz   count,f            ; decrement count
        goto     SPRWLP             ; repeat till zero

        bcf      PORTA,SCK          ; leave clock lo
        return


;------------------------------------------------
; puthex(w) put a 2 character hex on rs232
; also updates cksum global variable
;------------------------------------------------

puthex:
        banksel PORTA
        movwf    temp                         ; save byte
        swapf    temp,W                       ; get hi-byte
        call     hex2asc
        movf     temp,W
hex2asc:
        ANDLW    0x0F
        ADDLW    0x36
        btfsc    STATUS,DC
        ADDLW    0x07
        ADDLW    0-6
        ;goto     putchar                              ; char in W
        return

                 end
```

0.2.3 Control Program Overview

The control program is written in two parts. The primary interface is a C-language command line program which talks to the serial port and handles the low-level manipulation for the 9-bit address protocol. When debugging the system, this is the primary interface for testing. Because it is a self-contained UNIX command line program, it can be used stand-alone, in shell scripts, or called by other programs.

The basic usage of the program is:

```
ask [-d<device> -n<count> -c<cmd> -h -b(force error) -v(verbose)] databytes

ask() is a program for talking to a p-bus+ device.
```

It sends packet of various types to the serial port at -d<device>.

You can either specify the command with -c<cmd_number> or you can sym-link to this program as

These correspond to command numbers of 0x5f, 0x5c, 0x5e, 0x5d, 0x5b, 0x58, 0x10, and 0x11. Th

You are encouraged to look at the code for precise details of the packet format, but it is bas
The checksum has the property that if it is added to all the bytes in the packet the result wi
absence of errors.

0.2.4 C Language control program listing

```
// Jpsoft (c) 2001 Jap, http://jap.hu
//        (c) 2008 RCW, walker@omnisterra.com

// should optionally timestamp output with gettimeofday()

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <sys/time.h>
#include <termios.h>
#include <strings.h>
#include <sys/select.h>
#include <limits.h>
#include <errno.h>

//#define BAUD  B115200         /* baudrate */
//#define BAUD  B9600           /* baudrate */
#define BAUD    B19200          /* baudrate */
#define TIMEOUT 10000           /* read() timeout in usec */

#define MODEM   "/dev/ttyS0"
#define _BSD_SOURCE 1
#define TRUE    1
#define FALSE   0

typedef unsigned char byte;

typedef struct pkt {
    byte b0;    // address + size
    byte cmd;   // cmd
    byte data[20];
} PKT;

#define CGET    0x10
#define CSET    0x11
#define CLAST   0x5b
#define CPING   0x5f
#define CNOOP   0x58
#define CVER    0x5e
```

```
#define CRST    0x5c
#define CSTAT   0x5d
#define ROK     0x60
#define RBAD    0x61
#define RECHO   0x6f

void pkt_dump(PKT *p, int p12);
void dump_stats();
int  pkt_write(int fd, PKT *p);
void pkt_fill(PKT *p, int cmd, int dest, int count);
void setparity(int fd, int value);

struct termios oldtio, newtio;

int rstats[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

char *progname;
int badck=0;            // used to force a bad tx cksum
int p12=0;              // flag used to force 12 bit packet printing
int verbose=0;

byte u(byte a) {                // return upper nybble
   return((a&(~15))>>4);
}

byte l(byte a) {                // return lower nybble
   return(a&15);
}

pradc(byte *x) {
    printf("%d ",  256*u(*(x+0))+ 16*l(*(x+0))+ u(*(x+1)));
    printf("%d ",  256*l(*(x+1))+ 16*u(*(x+2))+ l(*(x+2)));
    printf("%d ",  256*u(*(x+3))+ 16*l(*(x+3))+ u(*(x+4)));
    printf("%d ",  256*l(*(x+4))+ 16*u(*(x+5))+ l(*(x+5)));
    printf("%d ",  256*u(*(x+6))+ 16*l(*(x+6))+ u(*(x+7)));
    printf("%d ",  256*l(*(x+7))+ 16*u(*(x+8))+ l(*(x+8)));
    printf("%d ",  256*u(*(x+9))+ 16*l(*(x+9))+ u(*(x+10)));
    printf("%d\n", 256*l(*(x+10))+16*u(*(x+11))+l(*(x+11)));
}

pradch(byte *x) {
    printf("%x%x%x ",   u(*(x+0)), l(*(x+0)), u(*(x+1)));
    printf("%x%x%x ",   l(*(x+1)), u(*(x+2)), l(*(x+2)));
    printf("%x%x%x ",   u(*(x+3)), l(*(x+3)), u(*(x+4)));
    printf("%x%x%x ",   l(*(x+4)), u(*(x+5)), l(*(x+5)));
    printf("%x%x%x ",   u(*(x+6)), l(*(x+6)), u(*(x+7)));
    printf("%x%x%x ",   l(*(x+7)), u(*(x+8)), l(*(x+8)));
    printf("%x%x%x ",   u(*(x+9)), l(*(x+9)), u(*(x+10)));
    printf("%x%x%x\n",  l(*(x+10)),u(*(x+11)),l(*(x+11)));
}

void setparity(int fd, int value) {

    struct termios tio;
    static int current_parity_mode=2;

    if (value != current_parity_mode) {
```

```
        bzero(&tio, sizeof(tio));
        tcgetattr(fd, &tio);
        if (value == 0) {
            tio.c_cflag &= ~PARODD;      /* space parity = 0 */
            current_parity_mode=0;
        } else {
            tio.c_cflag |= PARODD;       /* mark  parity = 1 */
            current_parity_mode=1;
        }
        tcsetattr(fd, TCSADRAIN, &tio);
    }
}

int main(int argc, char **argv)
{
    int fd;
    int d;
    int val;
    int i=0;
    int retry=2;
    PKT ps, pd;
    int retval=0;

    int c;

    int device=0;
    int cmd=CPING;
    int count=1;
    char *str;
    char *endptr;

    extern int optind;
    extern int opterr;
    extern char *optarg;
    int errflag = 0;
    opterr = 0;         /* disables getopt's error msg */

    progname = argv[0];

    if (strcmp(progname, "cping") == 0) {
        cmd = CPING;
    } else if (strcmp(progname, "cadc") == 0) {
        verbose=0;
        p12++;
        cmd = CGET;
    } else if (strcmp(progname, "cget") == 0) {
        cmd = CGET;
    } else if (strcmp(progname, "cset") == 0) {
        cmd = CSET;
    } else if (strcmp(progname, "creset") == 0) {
        cmd = CRST;
    } else if (strcmp(progname, "cstat") == 0) {
        cmd = CSTAT;
    } else if (strcmp(progname, "cver") == 0) {
        cmd = CVER;
    } else if (strcmp(progname, "clast") == 0) {
        cmd = CLAST;
```

```
    } else if (strcmp(progname, "cnoop") == 0) {
        cmd = CNOOP;
    } else {
        // printf("%s: bad program name\n", progname);
        cmd = CPING;
    }

    if (argc == 1)
        errflag++;  /* print usage if no com args    */
    while ((c = getopt(argc, argv, "bd:c:n:v")) != EOF)
        switch (c) {
        case 'b':
            badck++;
            break;
        case 'd':
            fflush(stdout);
            device=atoi(optarg);
            break;
        case 'c':
            cmd=atoi(optarg);
            break;
        case 'n':
            count=atoi(optarg);
            break;
        case 'v':
            verbose++;
            break;
        }
    if (errflag || device==0) {
        fprintf(stderr, "usage: %s [-d<device> -n<count> -c<cmd> -h(help) -b(force cksum error
        exit(2);
    }

    // collect numerical arguments

    int error=0;
    int index=0;
    for (; optind < argc; optind++) {
         str = argv[optind];
         errno = 0;
         val = (int) strtol(str, &endptr, 0);

         if ((errno == ERANGE && (val == LONG_MAX ||
                val == LONG_MIN)) || (errno != 0 && val == 0)) {
             printf("bad numerical argument: %s\n",str);
             error++;
         }
         if (endptr == str) {
             printf("couldn't convert number: %s\n",str);
             error++;
         }
         if (val < 0 || val > 255) {
             printf("number won't fit in a byte: %s\n",str);
             error++;
         }
         ps.data[index++]=(byte) val;
         // printf("%d\n",val);
```

```
    }

    if (index > 15) {
         printf("max payload is 15 bytes\n");
         error++;
    }

    if (error != 0) {
        exit(0);
    }

    fd = init_pbus(MODEM);

    pkt_fill(&ps, cmd, device, index);

    for (i=0; i<count; i++) {
        if (i > 0) {
            printf("----------------------\n");
        }

        if (pkt_write(fd, &ps) == 0) {
            if (verbose) {
                printf("%04d: sent: ", i);
                pkt_dump(&ps,0);
            }
        } else {
            retval++;
            printf("packet transmission error!\n");
        }

        usleep(10000);   // 22000

        if (pkt_read(fd, &pd)) {
            if (verbose) printf("%04d: rcvd: ", i);
            pkt_dump(&pd,p12);
        } else {
            printf("TIMEOUT - ");
            if (retry != 0) {
                printf("resending %d\n", retry);
                retry--;
                i--;
            } else {
                printf("giving up %d\n", retry);
                retry=2;
                retval++;
            }
            // rstats[d]++;
        }
        fflush(stdout);
    }                              // while
    close_pbus(fd);
    return(retval);
}

void pkt_fill(PKT *p, int cmd, int dest, int index) {
    (*p).b0 = (byte) ((dest << 4) + index);
    (*p).cmd = (byte) cmd;
```

```
}

static char* cmd_to_str(int cmd) {
    switch(cmd) {
        case CGET:
           return ("cget");
        case CSET:
           return ("cset");
        case CNOOP:
           return ("cnoop");
        case CLAST:
           return ("clast");
        case CRST:
           return ("creset");
        case CSTAT:
           return ("cstat");
        case CVER:
           return ("cver");
        case CPING:
           return ("cping");
        case ROK:
           return ("rok");
        case RBAD:
           return ("FMTERR");
        case RECHO:
           return ("recho");
        default:
           return ("UNDEF");
    }
}

int init_pbus(char *tty) {
    int fd;
    int d;
    struct termios oldtio, newtio;
    PKT ps, pd;

    fd = open(tty, O_RDWR | O_NOCTTY | O_NONBLOCK | O_SYNC);
    if (fd < 0) {
        perror(MODEM);
        exit(-1);
    }

    tcgetattr(fd, &oldtio);      /* save current settings */

    bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = BAUD|CS8|CLOCAL|CREAD|PARENB|CMSPAR|PARODD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;
    newtio.c_lflag = 0; /* non-canonical, no echo, ... */

    newtio.c_cc[VTIME]=10;       /* inter-char timer decisecs */
    newtio.c_cc[VMIN]=0;         /* block til n chars read */

    tcflush(fd, TCIFLUSH);
    tcsetattr(fd, TCSANOW, &newtio);
```

```
    return(fd);
}

close_pbus(int fd) {
    tcsetattr(fd, TCSANOW, &oldtio);    // restore settings
    close(fd);
}

void dump_stats() {
    int d;
    printf("-- STATS: ");
    for (d = 2; d < 8; d++) {
        printf("%d:%03d,%03d ", d, rstats[d]);
    }
    printf("\n--\n");
}

void pkt_dump(PKT *p, int p12)
{
    int dlen = (p->b0) & 15;
    int dest = ((p->b0) & 240) >> 4;
    int cmd = p->cmd;
    int cksum = p->data[dlen];
    int i;
    int myck = p->b0 + p->cmd;

    if (verbose) {
        printf("D=%02x L=%02x C=%02x S=%02x ", dest, dlen, cmd, cksum);
    }

    if (dlen > 0) {
        printf("DATA= ");
        for (i = 0; i < dlen; i++) {
            if (!p12) {
                printf("%02x ", p->data[i]);
            }
            myck += p->data[i];
        }
    }

    if (p12) {
        pradc(p->data);
    }

    myck = (byte) (256 - (myck & 255));

    if (myck == cksum) {
         if (verbose) printf(" %s\n", cmd_to_str(cmd));
    } else {
         printf("!!! ERROR ck=%02x !!!\n", myck);
    }
}

int serial_read(int fd)
{
    unsigned int ch = 0;
    fd_set readfds;
```

```
    struct timeval timeout;

    FD_ZERO(&readfds);
    FD_SET(fd, &readfds);
    timeout.tv_sec = 0;
    timeout.tv_usec = TIMEOUT;   /* timeout in usec */

    if (select(fd+1, &readfds, NULL, NULL, &timeout) == 1) {
        if (read(fd, &ch, 1) > 0) {
            return (ch);
        }
    }
    return (-1);                  // timeout or read error
}

int pkt_read(int fd, PKT *p)
{
    int i;
    int c;
    int myck=0;

    if ((c=serial_read(fd)) == -1) return (0);
    p->b0 = (byte) c;    // address(4:7), bytecount(0:3)
    myck += c;

    if ((c=serial_read(fd)) == -1) return (0);
    p->cmd = (byte) c;   // command
    myck += c;

    if ((c=serial_read(fd)) == -1) return (0);
    p->data[0] = (byte) c;
    myck += c;

    int dlen = (p->b0) & 15;
    int dest = ((p->b0) & 240) >> 4;
    int cmd = p->cmd;

    if (dlen > 0) {
        for (i = 1; i <= dlen; i++) {
            if ((c=serial_read(fd)) == -1) {
                return (0);
            }
            p->data[i] = (byte) c;
            myck += c;
        }
    }

    int cksum = p->data[dlen];

    if ((byte) myck) {
       printf(" parity error ");
       return(0);                 // parity error
    }
    return (1);
}

int pkt_write(int fd, PKT *p)
```

```
{
    unsigned int ch;
    int dlen = (p->b0) & 15;
    int cksum = 0;
    int i;
    int err=0;

    for (i = 0; i < 2; i++) {
        setparity(fd,i==0);      // first byte of pkt parity=1
        ch = *((byte *) p + i);
        cksum += ch;
        if (write(fd, &ch, 1) == -1) {
            err++;
        }
    }

    if (dlen > 0)
        for (i = 0; i < dlen; i++) {
            ch = p->data[i];
            cksum += ch;
            if (write(fd, &ch, 1) == -1) {
                err++;
            }
        }

    cksum = 256 - (cksum & 255) + badck;
    p->data[dlen] = cksum;

    if (write(fd, &cksum, 1) == -1) {
        err++;
    }
    return(err);
}

unsigned long getstamp(void)
{
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return (tv.tv_sec * 1000000 + tv.tv_usec);
}
```

0.2.5 Tcl/Tk GUI program for datalogging

The second portion of the program is a graphical user interface (GUI) written in Tcl/Tk which uses the C-interface for I/O.

```
#!/bin/sh
# the next line restarts using wish \
exec wish "$0" "$@"

package require Tk

proc notebook {w args} {
   frame $w
   pack [frame $w.top] -side top -fill x -anchor w
   rename $w _$w
   proc $w {cmd args} { #-- overloaded frame command
```

```
        set w [lindex [info level 0] 0]
        switch -- $cmd {
            add     {notebook'add   $w $args}
            raise   {notebook'raise $w $args}
            default {eval [linsert $args 0 _$w $cmd]}
        }
    }
    return $w
}

proc notebook'add {w title} {
    set btn [button $w.top.b$title -text $title -command [list $w raise $title]]
    pack $btn -side left -ipadx 5
    set f [frame $w.f$title -relief raised -borderwidth 2]
    pack $f -fill both -expand 1
    $btn invoke
    bind $btn <3> "destroy {$btn}; destroy {$f}" ;# (1)
    return $f
}

proc notebook'raise {w title} {
    foreach i [winfo children $w.top] {$i config -borderwidth 0}
    $w.top.b$title config -borderwidth 1
    set frame $w.f$title
    foreach i [winfo children $w] {
        if {![string match *top $i] && $i ne $frame} {pack forget $i}
    }
    pack $frame -fill both -expand 1
}

#---------------------------------------------------- test and demo code

set widgettab {
    {"run time" "RT"    #f5f "h:m:s"    -1      1.0      0.0}
    {"PHV A-B"  "PAB"   #ff5 "volts"     0      0.001    0.0}
    {"PHV B-C"  "PBC"   #ff5 "volts"     1      0.001    0.0}
    {"PHV C-A"  "PCA"   #ff5 "volts"     2      0.001    0.0}
    {"CTI A"    "CTA"   #5f5 "amps"      7      0.1053  30.0}
    {"CTI B"    "CTB"   #5f5 "amps"      6      0.1053  30.0}
    {"CTI C"    "CTC"   #5f5 "amps"      5      0.1053  30.0}
    {"TEMP"     "TEMP"  #ff5 "celcius"   3      0.001    0.0}
    {"HV"       "HV"    #ff5 "volts"     4      0.0098   2.0}
    {"KVA"      "KVA"   #5ff "kwatts"   -1      1.0      0.0}
}

set nwidget [llength $widgettab]

pack [notebook .n] -fill both -expand 1 ;# create notebook with tabs

set p1 [.n add Control]                 ;# create page 1
set p2 [.n add Options]                 ;# create page 2

set delta 2

grid [label $p1.cl1 -text "parameter" -padx 20 ] -column 0 -row 0
grid [label $p1.cl2 -text "value" -padx 20 ]     -column 1 -row 0
```

```
grid [label $p2.cl1 -text "gain" -padx 20 ]       -column 2 -row 0
grid [label $p2.cl2 -text "offset" -padx 20 ]     -column 3 -row 0


for {set i 0} {$i < $nwidget} {incr i 1} {
    set index [expr int($i+$delta)]

    ;# ------------ page 1 widgets
    set name [lindex [lindex $widgettab $i] 0]  ;# get label
    grid [label $p1.$i -text $name -padx 20 ] -column 0 -row $index
    set name [lindex [lindex $widgettab $i] 1]  ;# get varname
    set color [lindex [lindex $widgettab $i] 2] ;# get color
    global $name
    set $name 0
    grid [label $p1.v$i -textvar $name -bg $color -width 8] -column 1 -row $index
    set unit [lindex [lindex $widgettab $i] 3] ;# get units
    grid [label $p1.l$i -text $unit -width 8] -column 2 -row $index

    ;# ------------ page 2 widgets
    set ch [lindex [lindex $widgettab $i] 4]
    if {$ch >= 0} {
       set index [expr int($ch+$delta)]
       set channel CH$ch
       grid [label $p2.c$i -text $channel -padx 20 ] -column 0 -row $index
       set name [lindex [lindex $widgettab $i] 1]
       grid [label $p2.l$i -text $name -padx 20 ] -column 1 -row $index
       set gain [lindex [lindex $widgettab $i] 5]
       set color "#ff5"
       set $channel.gain $gain
       grid [entry $p2.g$i -textvar $channel.gain -width 8 -bg $color] -column 2 -row $index
       set offset [lindex [lindex $widgettab $i] 6]
       set $channel.offset $offset
       grid [entry $p2.o$i -textvar $channel.offset -width 8 -bg $color] -column 3 -row $index
    }
}

set logstat 0
set logmsg "start log"
set logfile "not logging"
set logfd ""

grid [button $p1.log -textvar logmsg -command startlog ] -column 0 -row 15
grid [label  $p1.logmsg -textvar logfile ] -column 1 -row 15

proc startlog {} {
    global logstat
    global logmsg
    global logfile
    global logfd

    if {$logstat} {
        set logstat 0
        set logmsg "start log"
        set logfile "not logging"
        close $logfd
        set logfd ""
    } else {
```

```
        set logstat 1
        set logmsg "stop log"
        set logfile [exec date +%Y-%m-%d-%H:%M:%S.log]
        set logfd [open $logfile a+]
    }
}

set runtime 0
set startdate [exec date +%s]

proc loop {} {
   global widgettab
   global runtime
   global startdate
   global logfd

   set prefix ""
   set date [exec date +%s.%N]

   ;# swap these next two lines for debugging:

   ;# set out [exec cadc -d5]
   set out "DATA: 1 2 3 4 5 6 7 8"

   ;# only accept data lines prefixed with DATA:
   if { [regexp "^DATA:" $out] == 1 } {
       set out [lreplace $out 0 0]

       set nwidget [llength $widgettab]
       for {set i 0} {$i < $nwidget} {incr i 1} {
          set name [lindex [lindex $widgettab $i] 1]
          set ch [lindex [lindex $widgettab $i] 4]
          global $name
          if {$ch >= 0} {
             global CH$ch.offset
             global CH$ch.gain
             set gain CH$ch.gain
             set offset CH$ch.offset
             set $name [expr [lindex $out $ch]*[set $gain]+[set $offset]]
          }
       }
   } else {
       ;# failed to get valid data
       ;# simply log the error, but don't update
       ;# any internal variables
       set prefix "#"
   }

   incr runtime
   set hour [expr floor($runtime/3600.0)]
   set min  [expr floor(($runtime-($hour*3600))/60)]
   set sec  [expr floor(($runtime-($hour*3600)-($min*60)))]
   set RT [format "%02g:%02g:%02g" $hour $min $sec]

   set KVA [format "%4g" [expr (sqrt(3)*12*(($CTA+$CTB+$CTC)/3)*$HV)/1000]]

   if { $logfd != "" } {
```

```
        puts $logfd "$prefix[format %.4g [expr $date-$startdate]] $PAB $PBC\
$PCA $CTA $CTB $CTC $TEMP $HV $KVA $out"
        flush $logfd
    }

    after 1000 loop
}

.n raise Control
wm geometry . 600x300

loop
```