ENGINEERING CHANGE ORDER

Number 03-		
Project Engineer	Stephen L. Robinson	Cross Ref. Doc. Type & Number: None
Change Requested By	Stephen L. Robinson	Page 1 of 1

Description of Change

Release of technical documentation for archive:

VideoWise Protocol , June 1997

Reason for Change None

Scope of Change

- ☐ Changes Form, Fit, or Function
- ☐ Other performance enhancement
- ☒ Internal

Type of Change

- ☐ New Product
- ☐ Error
- ☐ Design Improvement
- ☒ Additional Info
- ☐ Cost Reduction
- ☐ Conform to Present Practices

Material Disposition

- ☒ None
- ☐ Scrap
- ☐ Rework
  - ☐ Finished Goods
  - ☐ Work In Progress
  - ☐ Stock
- ☐ Running Change

Documentation Affected

Product Model Number: None

Drawing Number	Old Rev	New Rev
None		

Approvals	Engineering Signature	Date
	Materials Signature	Date
Cost Impact	0	New Comp. Cost: 0
Obsol. Impact	0	New Comp. Lead Time: 0

Manager's Initials in Appropriate Box

☐ EWS ☐ Hot ☒ Normal

Required Tasks (use attachments if necessary)	Initials	Date
Manufacturing None		
Production None		
Materials None		
Stock Room None		
Sales / Marketing None		
Repair None		
Quality Assurance None		
Other None		

From: Phillip McInnes <videowiz@icon.co.za>
To: STEPHEN; L.; ROBINSON
Subject: Videowise Communications Protocols
Date: Wednesday, June 25, 1997 11:18 AM

Hi Stephen

My name is Sean Kinghan, I will be handling this for Phillip and you can contact me at videowiz@icon.co.za

The documents dealing with the protocols comprise 4 or 5 separate sections - If you only have the one then you are definitely missing something. I will email the complete set directly to you tomorrow when I get back to the office.

As regards the use of equipment for testing, we will supply an exe file which runs under win3.1 or win95 and drives a PTZ using our protocol through one of the PC comm ports. If your dome uses RS485/RS422 you will need the appropriate convertor.

When do you expect to be ready for testing?

If I think of anything else I'll email you.

Regards

Sean

From: wklampfl@ix.netcom.com
To: tcovacev@pelco.com; dmartin@pelco.com; djmartin@prodigy.com
Cc: slrob@lightspeed.net; intl@pelco.com; ghurenka@pelco.com; dsmith@pelco.com; cynthia@cybergate.com; ctuttle@pelco.com; 75102@ix.netcom.com; 2717@compuserve.com
Subject: Fwd: RE: PhilTech and the Videowise Communications Protocols
Date: Friday, July 11, 1997 8:15 AM

Hello All,

The following is a response from Mike Cox at Specialised Control Technologies (South Africa) regarding the requirement for Spectra to communicate with others' matrices. This is self explanatory.

Basically, Sun International has a substantial exisiting investment in others' matrices and will not pull these out to substitute with our CM9760. This is primarily due to the substantial cost for replacement and second, due to features not yet supported by CM9760 (i.e. interface to coin counters, active GUI, compatibility with others' receivers and domes, etc.). Therefore it's critical that all new domes going in should be compatible to existing controller. Our competition is already doing this and, in doing so, specifically point out that Pelco DOES NOT comply. Need more be said?

Steve Robinson has been actively trying to get additional information on the Philtech protocol, and per the accompanying e-mail is awaiting an updated response from VideoWise. Any help in getting the needed software protocol to Steve would be greatly appreciated.

It should be noted that Sensormatic have already set up shop in South Africa, having already been successful in nailing down some heavy retail projects (i.e. Edgars supermarket chain) and are actively pursuing the casino market. In addition, they will be using their reference to the 1996 Olympics to gain ground on supplying for the upcoming 2004 Olympics in Cape Town.

Baxall is owned by Norbain, who several months ago acquired Reditron, the largest single security distributor on the African continent. Reditron has for a number of years been the sole Burle agent and has supplied the lion's share of systems to the commercial and industrial sector in South Africa, and have been supplying AutoDomes in increasing numbers to Sun International. Given Norbain's involvement, you can expect pricing and promotion to become even more aggressive from that camp. As an example, Norbain purchases approximately 40,000 cameras from Sony at a shot, and is thereby able to offer them to the market at a cost below the South African Sony agent's (Tedelex) purchase cost.

I trust you are all familiar with Betetech. This company, managed by Alan Armstrong, is the manufacturer of GYYR's new ControLink.

Diamond and Maxpro are fairly new to the market in South Africa, but you can expect them to take advantage of Pelco's relative weakness there. Again, price and cross compatibility will be the key considerations for purchase.

Pelco has an excellent opportunity to make a name for itself in South Africa, but it will require a dedicated investment in time to get products compatible, as well as a monetary investment to make the products price competitive. Bear in mind that Sensormatic, Norbain/Baxall and Burle all have offices established in South Africa and transfer their goods at manufacturer cost, thereby avoiding heavy import duties. Additionally, Norbain's bulk buying power lends itself to even further price reductions. To be a player in the market there (as in England) you need a corporate presense that can take advantage of lower transfer pricing. Normal distribution price markups will keep you out.

That's it for my soapbox for now. Let me know if there's anything I can do to help.

Best regards,
Werner

------Begin forward message------------------------

Return-Path: <cctv@icon.co.za>
Received: from oak.zilker.net (rsmitch@oak.zilker.net [198.252.182.129]) by ixmail7.ix.netcom.com (8.7.5/SMI-4.1/Netcom)
    id HAA01010; Thu, 10 Jul 1997 07:35:04 -0700 (PDT)
Received: by oak.zilker.net (8.8.5/zilker.1.126)
    id JAA08919; Thu, 10 Jul 1997 09:34:52 -0500 (CDT)
Received: from mail1.dial-up.net by oak.zilker.net (8.8.5/zilker.1.126)
    id JAA08915; Thu, 10 Jul 1997 09:34:48 -0500 (CDT)
Received: from .icon.co.za (m2-4-ndf.dial-up.net [196.34.156.68]) by mail1.dial-up.net (8.8.5/8.7.5/ICONfront#1) with SMTP id QAA12824 for <wklampfl@crockettint.com>; Thu, 10 Jul 1997 16:36:20 +0200 (GMT)
Date: Thu, 10 Jul 97 16:15:10 PDT
From: Mike Cox <cctv@icon.co.za>
Subject: RE: PhilTech and the Videowise Communications Protocols
To: Werner Klampfl <wklampfl@crockettint.com>
X-PRIORITY: 3 (Normal)
X-Mailer: Chameleon 4.6, TCP/IP for Windows, NetManage Inc.
Message-ID: <Chameleon.970710163130.cctv@.icon.co.za>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=us-ascii

To answer some of your questions.

Philtech / Videowise is a South African company who started up in the 1980's
ostensibly to service the casino market.
The members are Philip McInnes, Sean Kingham and Raymond Parfit.
They made for themselves a niche market supplying matrices after Teljoy
dropped out of the casino market.
They make 24 by 8, 48 by 16, 56 by 16 and a 156 by 32 matrix.
All matrices have time, date , camera titles, tours, sequencing, password
access, P.C interface, multiples of 16 alarm inputs, either n/o, n/c or a
change of status.
They also make coin/note counters for Glory and other similar machines for
casinos.
They provide displays for Jackpots and interface the coin/note/ jackpot info
into the matrix to super-impose data onto the cctv screen.
Many other items are manufactured such as video amps, P.C controlled audio
products.
That's a small selection of the equipment they have developed and sell to
the trade.

I estimate 40% of existing S.I casino;s use Philtech matrices now. Betatech
have sold 2 or 3 matrices, the remainder are old Teljoy matrices.

There will be 41 casino licenses issued in the very near future with 3 of the 41 already issued, Philtech matrices are waiting to be installed in these 3.
We can offer Spectra domes or Intercept if we can interface to the Philtech matrix, Teljoy's matrices would be a waste of time.
As to how many domes we could sell, well already Diamond has sold domes to S.I through Diamond supplying protocol info to Philtech who have developed a interface unit for $200.00 retail.

Should dealers win tenders for C.B.d systems and will not purchase Pelco matrices then we would still have some of the business through the dome sales.

How many could we sell ?
Very difficult to say.
With Norbain, Diamond, Sensormatic, Maxpro and others already selling domes I think we could sell +- 50 domes this year.
I understand all the other dome suppliers have given or Philtech have 'worked out' the protocol to drive the respective domes.

please keep me informed as to your decisions. S.I has asked on numerous occasions what's happening, and I have had to cloud the issue.

Mike

Name: CCTV
E-mail: CCTV <cctv@icon.co.za>
Date: 10/07/1997
Time: 16:15:10

This message was sent by Chameleon

------End forward message--------------------------

PELCO
300 WEST PONTIAC WAY
CLOVIS, CALIFORNIA, USA 93612

June 25, 1997

From: Stephen L. Robinson
Senior Software Engineering Manager
Voice (209) 292-1981 x 2344
Fax (209) 294-2697
slrob@lightspeed.net

Mr. Philip McInnes:

I have been talking with Werner Klampfl about doing some work to make Pelco equipment work with the "VideoWise Communications Protocols". Werner has provided me with a copy of the "Abridged protocol reference, Release 1.10 (97.03.18)".

He also tried to send me another document in a file he called "Philtech2" but that didn't come through the net correctly. The reference I have describes the message framing, check sum, etc. but does not describe any of the commands and data. Is that information is contained in the other document?

When I asked Werner for the other document he suggested (for a variety of reasons) that I make contact with you. I am trying to get this project going and would like the following information:

- Do you have an Internet address?
- Can I get a copy of the manual that details the command set?
- Can we make arrangements for the use of some of your equipment for testing?
- Do you have any suggestions or specific targets that we should be looking at?

I'm looking forward to hearing from you and working on this project.

Sincerely,

Stephen L. Robinson

From: wklampfl@ix.netcom.com
To: wklampfl@ix.netcom.com; slrob@lightspeed.net
Cc: ctuttle@pelco.com; cctv@icon.co.za
Subject: Re: PhilTech Protocol
Date: Tuesday, June 24, 1997 9:09 AM

Hello Steve,

Regarding the Philtech protocol, PHILTECH2, I think the best bet would be for you to contact Philip McInnes of Philtech directly for the protocol. I don't have an e-mail address for him, but the address, phone and fax numbers are as follows:

Philtech
141 8th Avenue
P.O. Box 59904, Kengray 2100
Bezvalley, Johannesburg 2094, South Africa
Tel.: (27-11) 622-4188
Fax: (27-11) 622-4174

Please let me know if this works out for you.

Thanks,
Werner

On 06/23/97 10:19:48 you wrote:
>
>Werner:
>
>I've checked my records and I've found that you gave me two files from
>PhilTech. The first file, titled "PHILTECH1", contains "VideoWise
>Communications Protocols, Abridged protocol reference, Release 1.10
>(97.03.18)". The second file, titleled "PHILTECH2", is corrupted or
>something. I can't read it.
>
>I recently sent you some notes on how our protocols handle patterns,
>presets, and zones. Did you receive it and was it what you needed?
>

AMAZON
800-201-7575

From: Werner Klampfl <wklampfl@crockettint.com>
To: 'Tuttle, Cynthia' <CTuttle@firewall1.pelco.com>
Cc: 'Steve Robinson' <slrob@lightspeed.net>; 'Mike Cox' <cctv@icon.co.za>
Subject: RE: PhilTech and the Videowise Communications Protocols
Date: Sunday, July 06, 1997 4:08 PM

Hello Cynthia,

Many thanks for your e-mail and ohone call. With regards to VideoWise, I am not familiar with this firm. However, being based in South Africa I can only assume they are wanting to make inroads to the vast casino market there. Accordingly I am copying this message to Mike Cox at Specialised Controls for his comments.

With regards to Philtech, please be aware that many of the casinos presently use Philtech control systems. A requirement of the casinos is that additional domes MUST be compatible with existing control/switching systems - hence the request.

I suppose the most important question should be "How many domes can we sell to Sun International and the rest of the South African market in the next 6-12 months if we can make Spectra (and Intercept) compatible with Philtech, with VideoWise, and with Baxall". I hope Mike can shed some light on this for us via his contacts there.

Best regards,
Werner

-----Original Message-----
From: Tuttle, Cynthia [SMTP:CTuttle@firewall1.pelco.com]
Sent: Monday, June 30, 1997 10:29 AM
To: 'wklampfl@ix.netcom.com'
Cc: Martin, David; 'slrob@lightspeed.net'
Subject: FW: PhilTech and the Videowise Communications Protocols

Hi Werner,

Per my phone mail, please help Stephen answer the questions he has
below.

For your information, Stephen estimates that this project alone could
require months worth of work. We need to do our due diligence with
respect to the market potential for our dome with this protocol, and
prioritize the development of this translator against Baxall for the UK.

Can you please call/respond via e-mail to help answer these questions?

Thanks
Cynthia

> ----------
> From: Stephen L. Robinson[SMTP:slrob@lightspeed.net]

> Sent: Friday, June 27, 1997 11:21 AM
> To: Cynthia Tuttle
> Subject: PhilTech and the Videowise Communications Protocols
>
> Greetings:
>
> I have just received a message from Sean Kinghan who will "be handling
> this
> for Phillip". I believe that Sean works for VideoWise since his
> address is
> "videowiz@icon.co.za".
>
> What I need is some sort of specifications and someone who can tell me
> if
> they think VideoWise is OK. For all I know, they're working with or
> for
> Sensormatic.
>
> I would like to know what products / systems PhilTech wants to work
> with,
> what the projected sales are, etc. Should we be working on this? No
> one's
> answered this question yet.
>
> Thanks, SLR
>

VideoWise Communications Protocols

Abridged protocol reference
Release1.10 (97.03.18)

Hardware and Signal Standards

The communications standards to which all Philtech devices conform are as follows :

> Inter-device connection is virtually always either RS-485 or RS-422. In those instances where a device has only a single-channel link, the documentation will specify whether the link is implemented as half-duplex (bi-directional shared) or simplex (uni-directional) only.
>
> The baud rate is variable, but typically 9600 BPS. Some systems will attempt to determine the maximum baud rate common to all devices tied together. This is done by having the master attempt communications at increasing rates with individual devices, and then using the highest rate *before* ACK failure as the system standard. The highest value attempted is commonly 57600 BPS. Note that some devices may be able to run at higher rates than others, due to shorter cable lengths etc. but must still conform to the system rate. It is possible to run various devices at differing baud rates, but this practice leads to problems with spurious characters, and unidentifiable packets at those devices which are just listening in.

Serial communication standards

On those devices which may possibly need to perform unplanned interfacing, the serial protocols used by all Philtech devices follow the following format :

> Byte transfer is asynchronous, each byte consists of 10 bits - 1 start bit, 8 data bits and 1 stop bit. Although many microcontrollers support a 9-bit addressing scheme, a PC cannot, which basically forces the use of an 8-bit format in those applications where inter-system computability must be maintained.
>
> The serial communications format therefore has a twist - this is the use of a flag byte to indicate a following address byte. The flag used is 0xFE - always. ***This value is reserved as an address - no device may ever have an address of 0xFE.*** The serial handler accepts requests to send data bytes, but also accepts requests to send addresses. A request to send an address results in a flag byte (0xFE) being shoved out before the address. The receiving device therefore knows that a given byte is an address, and can ignore all data following an address byte which is not its own. Of course, this means that any real data of value 0xFE is in trouble - the RXR thinks the next byte is an address. So the portion of the serial handler of the TXR that handles data transmission (as opposed to address transmission) sticks a second tag byte on in front of the real data byte, and the receiver knows when it gets two consecutive tag bytes that these in fact represent a single data byte. This is why no address may be equal to 0xFE - the receiver would interpret the tag + address pair of 0xFE+0xFE as a single data byte. (This technique is common in the C programming language where '\' represents the escape character in a string, but '\\' is a true '\'.)

On those applications which make use of closed system components (i.e. excluding a PC), and therefore will never need to interface to anything but well-known (proprietary) devices, the processors typically communicate in 9-bit mode. The master device uses the high order bit as an address flag mechanism in a similar manner to the flag byte scheme discussed above. (This method is supported in hardware by the 8051 processor, and takes a load off the software). See an 8051 reference for details.

On those applications in which multiple processors exist on a single board, the master processor may communicate with the slaves in either asynchronous or synchronous mode (in either 8 or 9-bit mode). The transaction is then usually at very much higher baud rates than are normally available by wire (up to 1 MHz maximum with an asynchronous schema).

Supported packet formats

The packet formats used by all Philtech devices have the following structure :

A packet consists of a header, followed by a data stream, and a tailing check-byte.
The header is a block of 5 bytes in the following order :

Destination address

This is the 8-bit address of the device intended to receive the packet. Each device must be uniquely identifiable by means of its address and device type. This means that while two devices of differing types may share the same address, two devices of the *same type* may not do so (unless this is deliberately done, of course). There are some limitations on the allowable addresses due to the use of certain addresses as flags or global addresses - these limitations are discussed in detail later in the document. In brief, all addresses up to but not including 0xF0 may be used without fear of conflict. All addresses above 0xF0 are either used, or reserved for future use by extended system commands.

Source address

This is the address of the transmitting device - it is typically used by the receiving device as confirmation of a valid source, and as a means of identifying the address to which a response should be sent. Many receivers will accept commands or data only from specified device addresses or a range of addresses - this is often a system requirement in secure applications.

Device types

The destination/source types are packed into the hi/lo-nibbles of this byte. The destination device type occupies the upper 4 bits of the byte, and is used by the receiving device to check whether it is being addressed - or some other device of the same address. The source device type occupies the lower 4 bits, and is used in the same way as the source address - both for identifying a return type, and to ensure that only valid devices may communicate with the receiver.

Primary command

This is the *required* primary command byte. All commands make use of this byte - a complete list of possible commands follows later. The primary command specifies the nature of the command packet, and allows the RX device to correctly interpret the following data (if any).

A special case is when this byte has the value 0x00. This will invoke system commands for the RXN device, for all device types. Other values of the primary command will have different meanings for different devices, but all devices respond to 0x00 as a system command invocation.

Secondary command

This is the optional secondary command byte. It must be sent, even if it has a null value, but is called optional because it need not necessarily contain valid data if the nature of the primary command is such that only one byte is required.

The packet data stream is of variable length. This does require rather more sophisticated code with respect to packet processing on the part of both TXN and RXN devices than is required with fixed-length packet handlers, but the benefits to a high-level process are huge.

> Note that all packets have the same header and checksum requirements, no matter what their data length.
>
> The data stream of a variable length packet has a preliminary byte which specifies the number of data bytes following. The receiver makes use of this byte to locate the last expected byte of this packet (the checksum), and tests the packet for correctness. The TXR constructs a packet with header, data length, data stream and checksum, and transmits the packet. The RXR then tests the packet on the basis of expected length. This implies that a faulty packet length will place the check byte at an incorrect location, leading to a bad CRC. The packet will fail, and communications cannot take place.

The design trade-off against a fixed-length packet is of an extra byte permanently embedded in the packet (the length byte), versus the ability to switch packet sizes instantaneously. This is often required in systems which regularly send messages of varying nature.

The checksum byte generated uses a cyclic redundancy check, which is more efficient at error detection than an XOR checksum. The generating polynomial byte used is 0xCC. Unfortunately, there is rather a lot of theory, and some strange code attached to this process, so a complete explanation is impossible. Most introductory texts to communications theory have a decent description. Both C and 8051 assembly language code for a CRC generating module is included in Appendix F, as well as a further discussion of CRCs and their properties..

> The checksum byte is a CRC of all preceding bytes in the packet (excluding itself). One of the peculiarities of this thing is that a CRC of the complete packet (*including* the packet CRC) should be equal to zero. So whether you generate a CRC for all data before the CRC and then test for equality, or generate for all data including the CRC and then test for zero is a matter of personal preference.
>
> It is important to note that the stuffing of 0xFE bytes (address & data flags) takes place ***after*** the CRC has been calculated, by the low-level serial drivers. Similarly, the process of picking out the extra flag bytes must take place before the CRC is calculated by the RXR. (This implies that a CRC of 0xFE will also equal two consecutive 0xFE bytes.)

Appendix C : Communications handling

The following code is a set of sample drivers for the communications handler, including serial and packet handlers.

SERIAL.C

```
#include <stdmac.h>
#include <reg51.h>
#include <board.h>
#include <command.h>
#include <serial.h>
#include <xtaldef.h>

#ifndef EOF
#define EOF -1
#endif

/* test for unit's own address */
/* code must use registerbank SER_BANK */
extern bool serAddressed(byte test);

extern void serCloseTX(void);
extern void serOpenTX(void);
extern void serDisableTX(void);
extern void serEnableTX(void);
extern void serInhibitTX(byte delay);
extern void serInhibitTXNext(void);
extern void serRelaxTX(void);

extern bool serClosedTX(void);
extern bool serRetardedTX(void);

/* transmission inhibit service routines */
extern byte cfgTXInhibit(void);
extern void serInhibSrv(byte ticks);

static mdata bool TX_Busy = FALSE;
static mdata bool RX_Quiet = TRUE;
public bool serSendActive(void) { return TX_Busy; }

public bool serAvailData(void) { return !inStreamEmpty(); }

public char serGetData(void)

        {
        while (inStreamEmpty()); /* wait */
        return inStreamGetChar();
        }

public char serPutData(char put)

        {
        if (!outStreamFull() && !serClosedTX())
                {
                outStreamPutChar(put);
                if (put == MSG_HEADER) outStreamPutChar(MSG_HEADER);
                if (!TX_Busy && !serRetardedTX()) TI = TRUE;
                return put;
                }
        return EOF;
        }
```

```
public char serPutAddr(char addr)

        {
        if (!outStreamFull() && !serClosedTX())
                {
                outStreamPutChar(MSG_HEADER);
                outStreamPutChar(addr);
                if (!TX_Busy && !serRetardedTX()) TI = TRUE;
                return addr;
                }
        return EOF;
        }


public char serPutBuf(char *buf, byte len)

        {
        if (!serClosedTX())
                {
                xdata byte i;
                for (i = 0; i < len; i++)
                if (outStreamFull()) break;
                   else
                        {
                        xdata byte put = buf[i];

                        outStreamPutChar(put);
                        if (put == MSG_HEADER)
                                outStreamPutChar(MSG_HEADER);
                        }
                if (!TX_Busy && !serRetardedTX()) TI = TRUE;
                return i;
                }

        return 0;
        }

public char serPutStr(char *putStr)

        {
        if (!serClosedTX())
                {
                xdata byte i, put;
                for (i = 0; (put = putStr[i]) != 0; i++)
                        if (outStreamFull()) break;
                            else
                                {
                                outStreamPutChar(put);
                                if (put == MSG_HEADER)
                                        outStreamPutChar(MSG_HEADER);
                                }

                if (!TX_Busy && !serRetardedTX()) TI = TRUE;
                return i;
                }

        return 0;
        }
```

```
#pragma registerbank(SER_BANK)

public void serIntHandler(void)

        {
        if (RI)
                {
                RI = FALSE;
                inStreamPutChar(SBUF);
                RX_Quiet = FALSE;
                }

        if (TI)
                {
                TI = FALSE;
                if (!outStreamEmpty() && !serRetardedTX())
                        {
                        ETXRX_ = HI;
                        SBUF = outStreamGetChar();
                        TX_Busy = TRUE;
                        }
                 else TX_Busy = ETXRX_ = FALSE;
                }
        }

#pragma registerbank(DEF_BANK)

public void serSetMode(byte mode)

        {
        SCON = 0x40;
        TMOD = (TMOD & 0x0f) | 0x20;
        }

public void serSetBaud(word baud)

        {
        PCON &= ~0x80;
        switch(baud)
                {
                case 19200: PCON |= 0x80;
                case 9600: TH1 = TL1 = SER_9600_BAUD; break;
                case 4800: TH1 = TL1 = SER_4800_BAUD; break;
                case 2400: TH1 = TL1 = SER_2400_BAUD; break;
                default : TH1 = TL1 = SER_1200_BAUD; break;
                }
        }


public void serInit(void)

        {
        inStreamFlush();
        outStreamFlush();

        serSetBaud(9600);
        serSetMode(1);

        serOpenTX();
        serEnableTX();
        serRelaxTX();

        TF1 = PS = LO;
        TR1 = TRUE;
        REN = ES = TRUE;
        ETXRX_ = LO;
        }
```

```
public void serService(byte ticks)

            {
            if (RX_Quiet) serInhibSrv(ticks);
                  else
                        {
                        RX_Quiet = TRUE;
                        serInhibitTXNext();
                        }
            if (!outStreamEmpty() && !TX_Busy)
                        if (!serRetardedTX()) TI = TRUE;
            }
```

TXCONTROL.C

```
#include <stdmac.h>
#include <reg51.h>
#include <serial.h>

// This flags access denial to the TX buffer for caller
static mdata bool TX_Closed = FALSE;

// This flags an ON/OFF transmit inhibit of data in TX Buffer
static mdata bool TX_Retarded = FALSE;

// This flags a timer countdown transmit inhibit of data in TX Buffer
// It is in effect a temporary, timed version of TX_Retarded
static mdata byte TX_Delay = 50;
static mdata byte TX_NextDelay = 30;

// This closes access by the caller to the TX buffer
public void serCloseTX(void) { TX_Closed = TRUE; }

// This opens access by the caller to the TX buffer
public void serOpenTX(void) { TX_Closed = FALSE; }

// This halts transmission of data in the TX buffer
public void serDisableTX(void) { TX_Retarded = TRUE; }

// This re-enables transmission of data in the TX buffer
public void serEnableTX(void) { TX_Retarded = FALSE; }

// This re-enables transmission of data in the TX buffer
// It also immediately invokes the interrupt routine for TX
public void serInvokeTX(void)

            {
            TX_Retarded = FALSE;
            if (!outStreamEmpty() && !serSendActive()) TI = TRUE;
            }

// This temporarily halts transmission of data in the TX buffer
public void serInhibitTX(byte delay) { TX_Delay = delay; }

// This loads the next precalculated delay
public void serInhibitTXNext(void) { TX_Delay = TX_NextDelay; }

// This re-enables transmission of data in the TX buffer
public void serRelaxTX(void) { TX_Delay = 0; }

// These return the state of the inhibit functions
public bool serClosedTX(void) { return TX_Closed; }
public bool serRetardedTX(void) { return TX_Retarded || TX_Delay; }
public bool serDelayedTX(void) { return TX_Delay; }
```

```
extern byte cfgTXInhibit(void);

public void serInhibSrv(byte ticks)

        {
        if (TX_Delay)
                if (TX_Delay > ticks) TX_Delay -= ticks;
                    else
                        {
                        TX_Delay = 0;
                        TX_NextDelay = cfgTXInhibit();
                        }
        }
```

PACKET.C

```
#include <stdmac.h>
#include <stdlib.h>
#include <string.h>
#include <command.h>
#include <serial.h>
#include <crc.h>

#include "pak.h"

/* these MUST be supplied by MAIN.C */
extern bool unitIsDevice(byte dev);
extern byte unitDevice(void);
extern bool unitIsAddress(byte addr);
extern byte unitAddress(void);

#define MAX_DLEN        32
#define VHDR_LEN        6
#define MAX_PLEN        (MAX_DLEN+VHDR_LEN)

static xdata byte cmdLen;
static xdata byte cmdBuffer[MAX_PLEN];
static xdata byte inBuffer[MAX_PLEN], outBuffer[MAX_PLEN];

static xdata byte pakTimeout = 0;
static xdata byte pakCount = 0;

static mdata bool msgRXD = FALSE;
static mdata bool checkAddr = FALSE;
static xdata byte expectCnt;

static byte pakChecksum(void *cptr, byte len)

        {
        xdata byte i;

        crcClear();
        for (i = NIL; i < len; i++)
                crcCalculate(((byte *)cptr)[i]);
        return crcResult();
        }


public bool pakIsVariable(void) { return TRUE; }
public void pakSetVariable(void) { pakClear(ON); }
public void pakClear(bool state)

        {
        pakCount = 0;
        pakTimeout = 0;
        msgRXD = FALSE;
        checkAddr = FALSE;
        }

/* used this order to maximise compiler register optimisation */
static void pakLoadHdr(byte dest, byte dev, byte prm, byte sec)

        {
        outBuffer[PDST] = dest;
        outBuffer[PCMD] = prm;
        outBuffer[PSCMD] = sec;
        outBuffer[PDVC] = ((dev & 0x0f) << 4) | unitDevice();
        outBuffer[PSRC] = unitAddress();
        }

static void pakDumpSer(byte len)

        {
        xdata byte i;

        serPutAddr(outBuffer[PDST]);
        for (i = 1; i <= len; i++)
                serPutData(outBuffer[i]);
        }


public void pakShort(byte dest, byte dev, byte prm, byte sec)

        {
        pakLoadHdr(dest,dev,prm,sec);
        outBuffer[PDL] = 0;
        outBuffer[VHDR_LEN] = pakChecksum(outBuffer,VHDR_LEN);
        pakDumpSer(VHDR_LEN);
        }


public void pakSingle(byte dest, byte dev, byte prm, byte sec, byte data0)

        {
        xdata byte len;

        pakLoadHdr(dest,dev,prm,sec);
        outBuffer[VHDR_LEN] = data0;
        len = VHDR_LEN+(outBuffer[PDL] = 1);
        outBuffer[len] = pakChecksum(outBuffer,len);
        pakDumpSer(len);
        }


public void pakPair(byte dest, byte dev, byte prm, byte sec, byte data0, byte data1)

        {
        xdata byte len;

        pakLoadHdr(dest,dev,prm,sec);
        outBuffer[VHDR_LEN] = data0;
        outBuffer[VHDR_LEN+1] = data1;
        len = VHDR_LEN+(outBuffer[PDL] = 2);
        outBuffer[len] = pakChecksum(outBuffer,len);
        pakDumpSer(len);
        }
```

```
public void pakArray(byte dest, byte dev, byte prm, byte sec, byte *dptr, byte len)

        {
        /* load header data to packet header */
        pakLoadHdr(dest,dev,prm,sec);

        /* @len = data length to be copied */
        if ((outBuffer[PDL] = len = min(len,MAX_DLEN)))
                memcpy(&outBuffer[VHDR_LEN],dptr,len);
        /* now @len = full buffer length (exc. CHK)*/
        len += VHDR_LEN;

        /* @outBuffer[PCHK] = checksum */
        outBuffer[len] = pakChecksum(outBuffer,len);
        /* dump to serial buffers */
        pakDumpSer(len);
        }


public void pakService(byte ticks)

        {
        if (pakCount && (pakTimeout += ticks) > PAK_TMOUT) pakCount = NIL;

        while (serAvailData())
                {
                xdata byte tempData;

                pakTimeout = NIL;

                if ((tempData = serGetData()) == MSG_HEADER)
                        {
                        if (checkAddr) inBuffer[pakCount] = MSG_HEADER;
                         else { checkAddr = TRUE; continue; }
                        }
                    else
                        {
                        if (checkAddr) inBuffer[(pakCount = PDST)] = tempData;
                           /* else can't be data if waiting for header byte so throw away */
                              else if (pakCount == PDST) continue;
                              else inBuffer[pakCount] = tempData;
                        }

                checkAddr = FALSE;

                if(++pakCount < VHDR_LEN) continue;
                    else if (pakCount == VHDR_LEN)
                        expectCnt = VHDR_LEN+min(inBuffer[PDL],MAX_DLEN)+1;
                    else if (pakCount >= expectCnt)
                        {
                        if (pakValid())
                                {
                                memmove(cmdBuffer,inBuffer,min(pakCount,MAX_PLEN));
                                msgRXD = TRUE;
                                cmdLen = cmdBuffer[PDL];
                                }
                        pakCount = NIL;
                        break;
                        }
                }
        }
```

```
/* Place BEFORE pakService in code */
static bool pakValid(void)

              {
              if (!unitIsAddress(inBuffer[PDST])) return FALSE;
                    else if (!unitIsDevice(inBuffer[PDVC] >> 4)) return FALSE;
                    else if (pakChecksum(inBuffer,pakCount) != 0) return FALSE;
              return TRUE;
              }


 public bool pakReceived(void) { return msgRXD; }
public void pakLoadCmd(void) { msgRXD = FALSE; }

/* destination address */
public byte cmdDestAddress(void) { return cmdBuffer[PDST]; }

/* destination device type */
public byte cmdDestDevice(void) { return (cmdBuffer[PDVC] >> 4) & 0x0f; }

/* source address */
public byte cmdSrcAddress(void) { return cmdBuffer[PSRC]; }

/* source device type */
public byte cmdSrcDevice(void) { return cmdBuffer[PDVC] & 0x0f; }

/* primary command */
public byte cmdPrimary(void) { return cmdBuffer[PCMD]; }

/* secondary command */
public byte cmdSecondary(void) { return cmdBuffer[PSCMD]; }

/* number of extended data bytes */
public byte cmdLength(void) { return cmdLen; }

/* individual data bytes */
public byte cmdData(byte pos) { return cmdBuffer[VHDR_LEN+pos]; }

/* individual data bytes */
public byte *cmdDataStr(void) { return &cmdBuffer[VHDR_LEN]; }

/* packet checksum */
public byte cmdChecksum(void) { return cmdBuffer[VHDR_LEN+cmdLen]; }

#undef __PAK_C__
```

Appendix F : Checksum generation

Cyclic redundancy check (CRC) error-checking methods are based on the behaviour of polynomials under two's complement arithmetic. There is extensive literature available on the theory and practice of CRCs. It is unnecessary to use 16-bit CRCs with an 8-bit micro, and we use an 8-bit code (0CCH) as recommended by Dallas and used in one of their chips.

CRCs are far more efficient than XOR schemes in detecting errors, although they take longer to generate. The XOR method is susceptible to failure under any number of error conditions. A 2-bit error burst is a typical example - the XOR will fail to detect any even-numbered inversion string of consecutive individual bits, say bit position 5. And that's not all it loses. This is basically because the XOR loses information in its operation - it cannot encode the two-dimensional data stream into a similar two-dimensionally coded check byte. Instead it can only form 8 one-dimensional checks. (The data stream is 2-D in the sense that it can be seen as an Nx8 array of bits. The CRC is 2-D in the sense that the *order* of bits in the checksum reflects, in part, the order of the bits in the data stream.) CRCs are guaranteed to detect a whole range of possible errors, with the actual efficiency of detection dependent on the length of the polynomial used and the nature of the polynomial's behaviour in arithmetic space. Note that they are not used in any way for error-correction (as in Reed-Solomon codes etc.).

```
#define byte unsigned char
#define POLY 0xCC /* CRC calculation polynomial */

static byte crcScratch; /* scratchpad register variable */

/* clear scratchpad before starting calcs */
/* public */ void crcClear(void) { crcScratch = 0x00; }

/* run calcs on consecutive data bytes */
/* public */ void crcCalculate(byte calc)

        {
        byte loop, work, flag;
        for (loop = 0, work = calc; loop < 8; loop++)
                {
                calc = work; /* hold most recent bit-shifted value in 'calc' */
                work ^= crcScratch; /* XOR against current scratch-pad value */
                flag = work & 0x01; /* check on XORed LSB state */
                /* get 'scratch' or 'scratch XOR POLY' depending on LSB state */
                work = (flag ? crcScratch ^ POLY : crcScratch) >> 1;
                crcScratch = work |= flag ? 0x80 : 0x00; /* rotate right with carry */
                work = calc >> 1; /* start looking at next bit of data byte */
                }
        }

/* return result of calcs */
/* public */ byte crcResult(void) { return crcScratch; }
```

The assembly code used in this module was copied from a Dallas application note, converted to 8051 assembly language, tightened up quite a bit, and made slightly more readable.

```
; This module coded for the REGISTER PARAMETER model
; Data is transferred in R7 - both incoming & outgoing
PUBLIC crcClear, crcCalculate, crcResult

?DT?crcDataSeg?CRC        SEGMENT DATA   ; scratch-pad data area
?PR?crcClear?CRC          SEGMENT CODE   ; crcClear() code segment
?PR?crcCalculate?CRC      SEGMENT CODE   ; crcCalculate() code segment
?PR?crcResult?CRC         SEGMENT CODE   ; crcResult() code segment

POLY            EQU 0CCH                 ; value of generator polynomial

RSEG ?DT?crcDataSeg?CRC
crcScratch :    DS 1                     ; assign single byte scratch-pad

RSEG ?PR?crcClear?CRC
crcClear:
        mov     crcScratch,#00H          ; clear CRC scratchpad register
        ret                              ; return to caller

RSEG ?PR?crcCalculate?CRC
crcCalculate:                            ; entered with data byte in R7
        mov     R0,#08H                  ; set loop for 8 bits
        mov     A,R7                     ; load data byte into ACC
calcNextBit:                             ; generate CRC for lowest bit
        mov     R7,A                     ; save bits to be shifted
        xrl     A,crcScratch             ; xor data byte with CRC
        rrc     A                        ; move to carry bit
        mov     A,crcScratch             ; get last CRC value
        jnc     calcNoXOR                ; skip if xor == 0
        xrl     A,#POLY                  ; update CRC value
calcNoXOR:                               ; prepare next bit from data byte
        rrc     A                        ; setup new CRC
        mov     crcScratch,A             ; store new CRC
        mov     A,R7                     ; get remaining bits
        rr      A                        ; position next bit in LSB
        djnz    R0,calcNextBit           ; test if another bit
        ret                              ; return to caller

RSEG ?PR?crcResult?CRC
crcResult:
        mov     R7,crcScratch            ; place scratch value into return register
        ret                              ; return value to caller in R7
```

Both preceding pieces of code have been assembled/compiled and tested with real data. As a test example, the sequence crcClear(), crcCalculate(0x5E), crcCalculate(0x3F), ... should return crcResult() equal to 0xAC.

i.e. CLR, 0x5E, 0x3F, 0x01 = 0xAC