Communications Layout for the TXB-S422

17 July 2002

TABLE OF CONTENTS

Section **Page**

[1] `$Header: d:/txb-s422/RCS/commlay.tex,v 1.7 2001-10-17 11:56:03-07 Hamilton Exp Hamilton $`

This page intentionally left blank

1 General

Part numbers for this project have not been established. When they are the information will be placed here.

The following Pelco part numbers apply to this project. Each time a software rev is made, the rev number on the IC51, PG51, BH51 and FW00 parts, must be updated.

This listing must be updated before releasing the code.

Part number	Use/Description
`PA05-xxxx-00x0`	Is the fully assembled board with many other things on it.
`IC51-xxxx-0100`	The programmed IC that gets stuck on the board.
`IC01-xxxx-00xx`	`PIC16F873` chip that gets stuff written into.
`PG51-xxxx-0100`	All combined binary hex files. There is only one `.HEX` file on this project.
`BH51-xxxx-0100`	Single binary hex file in `.HEX` format
`FW00-xxxx-0100`	Source code for this project. Consists of four files: The main source file, `TXBS422.ASM` MicroChip's chip include file D protocol include file S protocol include file

To operate correctly the crystal running this chip must be cut for 11.0592 MHz

1.1 Debug Defines

To enable this group of defines, `debug` must be defined.

DEBUGADDRESS 6 Makes it so that when debugging it is possible to avoid asking the Spectra its address. This specifies what address to use.

If `debug` isn't specified and real debugging is being done, then it is very unlikely that the system will boot up properly. During a normal boot up sequence this software looks at the Spectra and expects several events to happen in sequence. If the sequence of initial events is improper, then this software will hang-up, mostly waiting for a reply from the Spectra with the Spectra's address in it. As a solution to this problem, when in `debug` mode it is possible to specify an address for the Spectra. This gets around the whole boot up problem. If you are not running in `debug` mode, then each time that the software gets changed there must be a complete power cycle of the Spectra done with careful starting of the TXB-S422's software. (Not a fun activity!)

TESTPROPORTINAL is used to enable testing of proportional speed commands from an RC216 controller when we only have an AD2083/02. (The RC216 sends different speed values than the AD2083/02 does. This forces a test for controller type to always report than an RC216 is installed when any variable speed pan command comes in.)

Modes The only difference in modes between `debug` and "normal" configuration settings is that in `debug` mode I do not enable code protection. Code protection is only enabled in released code. MicroChip does not recommend enabling it on EPROM type devices.

The mode string for "normal" mode code: `__config _boden_on&_hs_osc&_pwrte_on&_wdt_off&_cp_off&_wrt_enable_off`

The mode string for `debug` mode code: `__config _boden_on&_hs_osc&_pwrte_on&_wdt_off&_cp_all&_wrt_enable_off`

Processor include files automatically change depending if a PIC16F873 or PIC16F876 is externally specified.

1.2 General Defines

listincludes when defined, makes all included files list themselves out.

Processor include files automatically change depending if a PIC16F873 or PIC16F876 is externally specified.

D Protocol include file named `dproto.inc` is used.

S Protocol include file named `sdefs.inc` is used.

1.3 Memory Reference Defines

RESETVECTOR	0x0000	Reset vector entrance
PROGRAM_PAGE0	0x0010	Program area 0
PROGRAM_PAGE1	0x0800	Program area 1
RAM0	0x020	RAM Bank 0
RAM1	0x0A0	RAM Bank 1
RAM2	0x120	RAM Bank 2
RAM3	0x1A0	RAM Bank 3

1.4 Internal EEPROM defines

This memory must be initialized before use. This is done transparently to the user by setting a flag in four consecutive bytes. If the flag is set then there is no initialization is needed, but if it is missing then the preset ID is set to `0x00` and the pan/tilt speeds are set to their default values.

When `EEPROM` has been initialized the four `EESETMEMx` bytes will have `0xDEADBEEF` written into them, i.e. `EEPROM0` = `0xDE` and `EEPROM3` = `0xEF`

EEPRESET	0x22	Handy place in EEPROM to use for presets IDs
EEPANSPEED	0x23	EEPROM version of fixed pan speed
EETILTSPEED	0x24	EEPROM version of fixed tilt speed
EESETMEM0	0x40	EEPROM flag for being initialized, part 1 0xDE
EESETMEM1	0x41	EEPROM flag for being initialized, part 2 0xAD
EESETMEM2	0x42	EEPROM flag for being initialized, part 3 0xBE
EESETMEM3	0x43	EEPROM flag for being initialized, part 4 0xEF

1.5 Misc defines

BADTRY	10	Number of bad inputs before flipping
BLANK	0x20	Blank character
CR	0x0D	Carriage return
DEFAULTPANSPEED	50	24.1/24^o/sec
DEFAULTTILTSPEED	54	24.8/24^o/sec
IDLELOOKS	216	How many times to look for an idle IDLELOOKS must be a multiple of 9
IGNOREADDRESS	64	Messages to address 64 are usually ignored
LF	0x0A	Line feed
LONGLOOK	30	Make the idle looking take longer
MAXPAN	0x40	Maximum legal pan speed
MAXTILT	0x3F	Maximum legal tilt speed
PAN48	57	46.0/48^o/sec
PAN72	62	72.9/72^o/sec
PAN96	64	150/96^o/sec
PELCODOME	0xF8	Dome type to send to the controller
	0xE5	SpeedDome 2000 2-board response dome 2
	0xF5	SpeedDome Ultra response dome 4
	0xF8	SpeedDome 2000 2-board response dome 5
	0xF8	SpeedDome 2000 1-board response dome 1
SGOTOPOSITIONSIZE	13	0xA6 Command length
SPROPSPEEDSIZE	4	0xC3 Command length
SPROTOLENGTH	0	Fake length of an S protocol command
SUNKNOWNC1SIZE	12	0xC1 Command length
SVARIABLESPEEDSIZE	5	0xC0 Command length
TILT48	0x20	48^o/sec
TILT72	0x30	72^o/sec
TILT96	0x40	96^o/sec

1.6 IO defines

spare2	porta,bit0	Pin 2	spare spare spare
ENABLEINPUTS	porta,bit1	Pin 3	RS-422/485 input enable Control inputs from the head-end 0 enables inputs 1 disables inputs
INVERTIO	porta,bit2	Pin 4	Used to invert input/output data 0 = normal input levels 1 = input data is inverted controls an XOR gate. INVERTIO Input Output Format 0 1 1 normal 0 0 0 normal 1 1 0 inverted 1 0 1 inverted
SPECTRAOUT	porta,bit3	Pin 5	Serial data to Spectra
spare6	porta,bit4	Pin 6	spare spare spare
SPECTRAIN	porta,bit5	Pin 7	Serial data from Spectra
notexist1	porta,bit6	Pin -	IO pin does not exist
notexist2	porta,bit7	Pin -	IO pin does not exist
ADDRESS0	portb,bit0	Pin 21	One bit of address data SW1-1
ADDRESS1	portb,bit1	Pin 22	One bit of address data SW1-2
ADDRESS2	portb,bit2	Pin 23	One bit of address data SW1-3
ADDRESS3	portb,bit3	Pin 24	One bit of address data SW1-4
PERMIT64	portb,bit4	Pin 25	Permit address 64 to be used
DEBUGMODEON	portb,bit5	Pin 26	Enables debug outputs SW1-6
DATAOUT	portb,bit6	Pin 27	ICSP and RS-232 to debug monitor
spare28	portb,bit7	Pin 28	ICSP and spare otherwise
spare11	portc,bit0	Pin 11	spare spare spare
spare12	portc,bit1	Pin 12	spare spare spare
spare13	portc,bit2	Pin 13	spare spare spare
spare14	portc,bit3	Pin 14	spare spare spare
spare15	portc,bit4	Pin 15	spare spare spare
ENABLEOUTPUTS	portc,bit5	Pin 16	RS-422/485 output enable Controls outputs to the head-end 0 disables outputs 1 enables outputs
UARTOUT	portc,bit6	Pin 17	Output of the USART
UARTIN	portc,bit7	Pin 18	Input to the USART

1.7 Global byte definitions

1.7.1 mode1_M

NORMALPOLARITY	mode1_M,bit0	0 = normal polarity, 1 = reversed
RESERVEDBIT	mode1_M,bit1	MUST remain at 0
DEBUGMODE	mode1_M,bit2	Indicates that we are in debug mode 0 = No debug output 1 = Yes debug output
INPUTERROR	mode1_M,bit3	Input error flag
CLEARDISPLAY	mode1_M,bit4	Should the Spectra display be cleared? 0 = No 1 = Yes
RC216MODE	mode1_M,bit5	Indicates the mode for speed commands 0 = AD2083/02, or other 8 speed source 1 = RC216, or other many speed device

1.7.2 mode2_M

BYTE1	mode2_M,bit0	Used in receiving while transmitting
BYTE2	mode2_M,bit1	. . to the Spectra
BYTE3	mode2_M,bit2	. . . . indicates that all are in

1.7.3 mkmode1_M

In the land of Sensormatic, several camera options are selected by using several buttons simultaneously. These bits are used to control the detecting of these special simultaneous button depressing.

STEP1RESET	mkmode1_M,bit0	A dome reset consists of two commands
STEP2RESET	mkmode1_M,bit1	First a zoom out followed by a FOCUS FAR and last by an IRIS OPEN command
LASTFAST	mkmode1_M,bit2	Set when FAST (0x8D) is received 0 = Normal command 1 = Last command was a FAST command
STEP1MENU	mkmode1_M,bit3	Set when IRIS OPEN (0x90) is received 0 = Normal command 1 = Last command = IRIS OPEN
STEP2MENU	mkmode1_M,bit4	Set when a FOCUS X command is received (X = 0x87, NEAR or 0x88, FAR). 0 = Normal command 1 = Last command = FOCUS X

1.8 Baud rate defines

1.8.1 Spectra IO defines

DBAUDBASIC and DBAUDREPEAT are used to control how long to delay when sending each bit, of bit-banged, data to the Spectra at 2400 baud. The original plan was to take baudperiod and multiply it by 8 ($19200 = 2400 * 8$) to get a delay time. However when this is done the result is greater than an eight bit number can hold (376) so a pair of values were needed. One for a major loop and one for a minor inside loop.

DBAUDBASIC	47
DBAUDREPEAT	8

1.8.2 Debug output defines

Note as these get to higher speeds that more compensation is needed for what otherwise is minor overhead introduced elsewhere.

DEBUGPERIOD	4	Bit duration of debug data	115200
DEBUGPERIOD	12	Bit duration of debug data	57600
DEBUGPERIOD	20	Bit duration of debug data	38400
DEBUGPERIOD	44	Bit duration of debug data	19200
DEBUGPERIOD	90	Bit duration of debug data	9600
DEBUGPERIOD	180	Bit duration of debug data	4800

1.8.3 Controller communications defines

The Baud Rate Generator is used to provide a clock to the on-board USART which is used for output communications with the head end. All other serial IO is bit-banged.

BAUD1200, BAUD2400, BAUD4800, BAUD9600 and BAUD19200 are used to load the baud rate generator for the on board USART. The data for all five of these equates assumes that BRGH is 0 (for low speed operation which gives an internal divide of 64 instead of 16.) It is assumed that the xtal is cut for 11,059,200 Hz (AKA 11.0592 MHz).

Baud Rate Generator formula from page 101 of one of the MicroChip is:

$$baudrate = Fosc/(64(BRG + 1))$$

Solving for BRG gives us:

$$BRG = (Fosc/(baudrate * 64)) - 1$$

or

$$BRG = ((Fosc/64)/baudrate) - 1$$

$$BRG = ((11095200/64)/baudrate) - 1$$

$$BRG = (173362.5/baudrate) - 1$$

The above calculations got me to an approximate starting value. I then fooled around with the numbers until they worked.

`BAUD1200`	143	1,200 baud
`BAUD2400`	68	2,400 baud
`BAUD4800`	34	4,800 baud
`BAUD9600`	17	9,600 baud
`BAUD19200`	8	19,200 baud

1.9 Bit position defines for bit testing

`bit0`	`0x00`	Bit position '0'
`bit1`	`0x01`	Bit position '1'
`bit2`	`0x02`	Bit position '2'
`bit3`	`0x03`	Bit position '3'
`bit4`	`0x04`	Bit position '4'
`bit5`	`0x05`	Bit position '5'
`bit6`	`0x06`	Bit position '6'
`bit7`	`0x07`	Bit position '7'

2 Naming conventions

Each RAM location will be have a name of some type that ends with an underscore "_" and the area of RAM that the variable is located in. There are the following four areas of memory: 0 = Last 96 locations in RAM bank 0, 1 = All 96 bytes of RAM bank 1. In case of doubt always assume that the item is globally accessible, most of the unmarked items are defined by the MicroChip assembler and I do not want to change their names.

Macros are in all upper case. (Just like in well written C programs.)

Note that the following RAM locations are non-functional. That is why there are "holes" in the memory allocations as they represent unused peripheral options.

`0x08`, `0x09`, `0x88`, `0x89`, `0x8F`, `0x90`, `0x95`, `0x97`, `0x9A`, `0x9B`, `0x9C`, `0x9D`

`RAM 0` is used to store almost all variables.

The number following the name is the subroutine level that uses that RAM location. A RAM location may be used only by that level of subroutine, or the main line code. However a higher level routine may READ but NOT modify a lower level byte. There is a special suffix allowed and that is `_M` for items only accessed from the main line code.

There is a special convention adopted to indicate which bank of RAM is active. That convention is to always select RAM bank 0, UNLESS a different bank is needed for some reason AND when a different RAM bank is selected to indicate it in the right hand column.

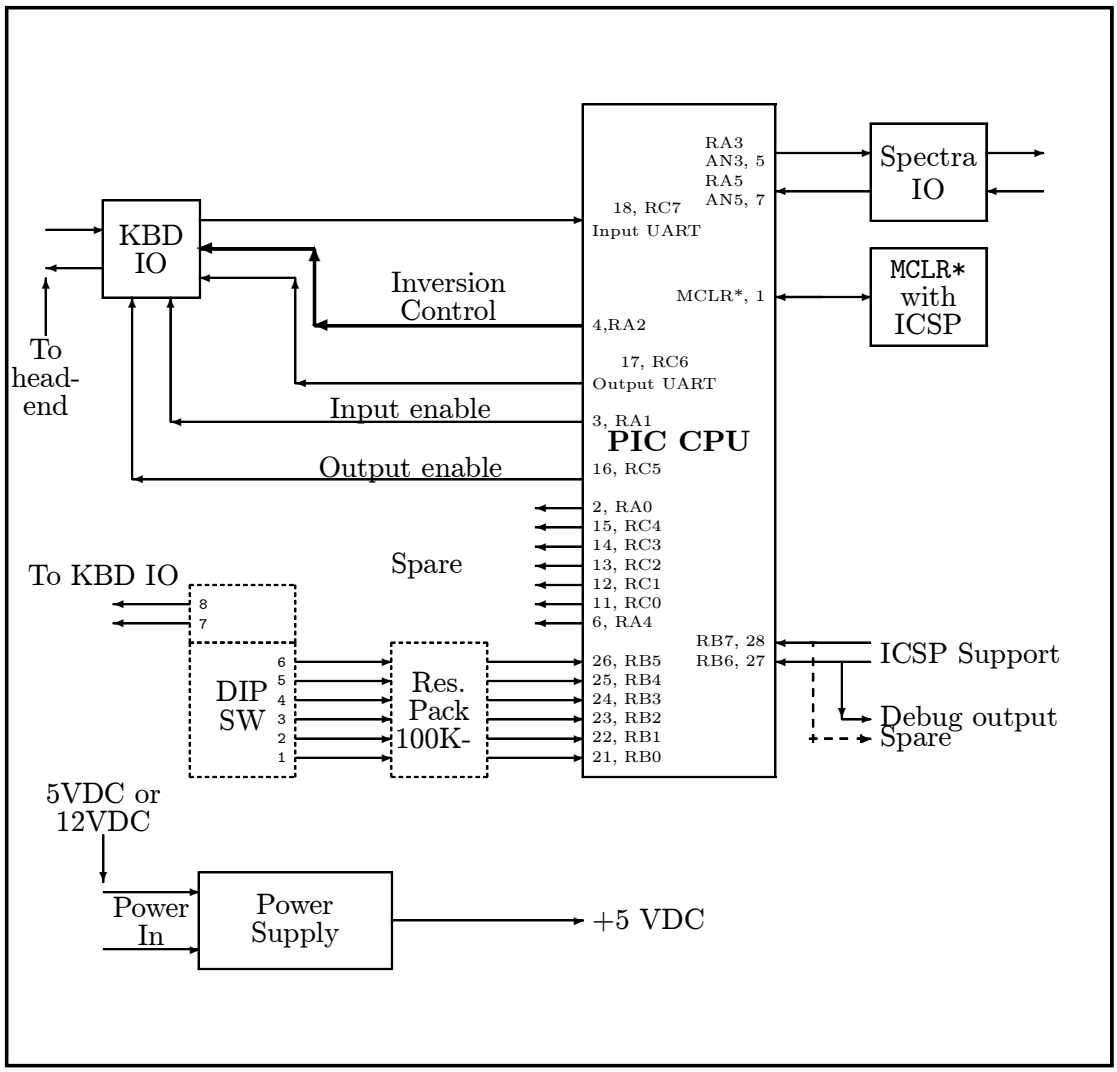
2.0.1 Contents of RAM0

bytebits_1	Number of bytes to bitbang out
dproto1_M	D proto sync byte
dproto2_M	D proto address byte
dproto3_M	D proto command 1
dproto4_M	D proto command 2
dproto5_M	D proto data 1
dproto6_M	D proto data 2
dproto7_M	D proto checksum
eewritehere_0	Where to write into in EE memory
extratimeout_0	For long timeouts that are too big
hexlower_0	Debug lower half of a byte in ASCII
hexupper_0	Debug upper half of a byte in ASCII
lastchecksum_M	Most recent checksum
messcount_4	Debug message counter/indicator thing
mkmode1_M	Command bits relating to multiple simultaneous key depressions
mode1_M	System mode part 1
mode2_M	System mode part 2
overlong1_0	Used in communicating with the Spectra
overlong2_0	. these are needed because 2400 baud . . is real slow and we can't delay . . . long enough with just 8-bits
panspeed_M	Pan speed to use in commands
panstatus_M	Current pan type command
patternwas_M	Last pattern worked on

save3_M	D Protocol saving space
save4_M	D Protocol saving space
save5_M	D Protocol saving space
save6_M	D Protocol saving space
savedpanspeed_M	Saved pan speed after a FAST command
savedtiltspeed_M	Saved tilt speed after a FAST command
sentbyte_1	Byte being bit banged out
spectraaddress_M	Address for the Spectra
sprotolength_M	Length of this S protocol message
sproto1_M	S protocol address
sproto2_M	S protocol command
sproto3_M	S protocol checksum
sproto4_M	S protocol long message byte 4
sproto5_M	S protocol long message byte 5
sproto6_M	S protocol long message byte 6
sproto7_M	S protocol long message byte 7
sproto8_M	S protocol long message byte 8
sproto9_M	S protocol long message byte 9
sproto10_M	S protocol long message byte 10
sproto11_M	S protocol long message byte 11
sproto12_M	S protocol long message byte 12
sproto13_M	S protocol long message byte 13
temp1_0	Used to get around instruction set
temp2_0	Used to get around instruction set
thisbit_0	Used in bitbanging in Spectra data
thisioerror_0	Most recent serial IO error
tiltspeed_M	Speed command to use for tilt commands
tiltstatus_M	Type of tilt command just sent
timeout_0	Decrementing delay value

3 Hardware description

[2] $Header: d:/txb-s422/RCS/txbshard.inc,v 1.11 2002-02-06 15:57:30-08 Hamilton Exp Hamilton $



$RCSfile: txbshard.inc,v $

Figure 1. Hardware design for the TXB-S422 unit

3.1 PIC CPU IO pin usage

PIC 16C73C (or 16C63C), IO port usage			
Bit	RAx	RBx	RCx
0	2, Spare	21, `SW-1`	11, Spare
1	3, Input data enable	22, `SW-2`	12, Spare
2	4, Inversion Control	23, `SW-3`	13, Spare
3	5, Spectra Serial Out	24, `SW-4`	14, Spare
4	6, Spare	25, `SW-5`	15, Spare
5	7, Spectra Serial In	26, `SW-6`	16, Output data enable
6	N/A	27, ICSP Support/Debug Output	17, HE Output UART
7	N/A	28, ICSP Support/Spare	18, HE Input UART
Note that in this table, references are: Pin #,Use			

Table 1. CPU IO pin usage

3.2 Connector pin assignments

There is one 16 pin interface connector on the TXB-S422 translator. The connector is designed to be plugged into the accessory jack of either a Spectra or Esprit.

There is an additional 5 pin connector which is used to support In Circuit Serial Programming (ICSP) of the PIC CPU chip.

The shorting plug that is normally installed in J2 connects the following pins together: 3-4, 5-6, 7-8, 9-10, 11-12, 13-14. And leaves the following pins open: 1, 2, 15, 16.

J2 Spectra/Esprit connection	
Pin	Use
1	+5 VDC, From an Esprit, open with a Spectra
2	+12 VDC, From a Spectra, open with an Esprit
3	Ground
4	Open on a Spectra, unknown on an Esprit
5	RXD+, To Spectra
6	RX+, From host
7	RXD-, To Spectra
8	RX-, From host
9	TXD+, From Spectra
10	TX-, To host
11	TXD-, From Spectra
12	TX-, To host
13	Video in, jumpered to pin 14 on the TXB-S422
14	Video out, jumpered to pin 13 on the TXB-S422
15	Ground
16	Open on a Spectra, unknown on an Esprit

Table 2. Connector pin assignments

3.3 Pin assignments on the PIC CPU chip

`RA0`, `RA1`, `RA2`, `RA3` and `RA5` are usable as A/D converter inputs on 16C7x (x = 2, 3 or 4) type chips.

`RA4` has an open collector type of output.

3.4 Switch bit assignments for SW

SW	Use
1	LSB of RC216 speed control
2	MSB of RC216 speed control
3	SpeedDome/UltraDome line seizing control.
4	Extended address bit 3, or spare
5	Enable decoding of address 64
6	Enable debug mode
7	Terminate controller input communications
8	Terminate controller output communications

The two switch positions that control communications termination are shipped from the factory in the "on" position. To remove termination of the communications lines, change the sub-switches to the "off" position. The state of these two sub-switches may not be monitored by the CPU chip.

4 Interfacing a Spectra

4.1 To an RC58 controller

A Spectra with a TXB-S422 installed in it may be connected to an RC58 (the controller of a SensorVision™ Programmable Video Management System). This is done by connecting a cable from the IN/OUT J-BOX connectors. To do this make the following connections at point Ⓐ in Figure 2, page 19:

Spectra		Direction	RC58
P_4	CONTROL		IN/OUT
4	RX —	From Controller	Tip
3	RX +	From Controller	Ring
	—	Shield	Sleeve
2	TX —	To Controller	Tip
1	TX +	To Controller	Ring

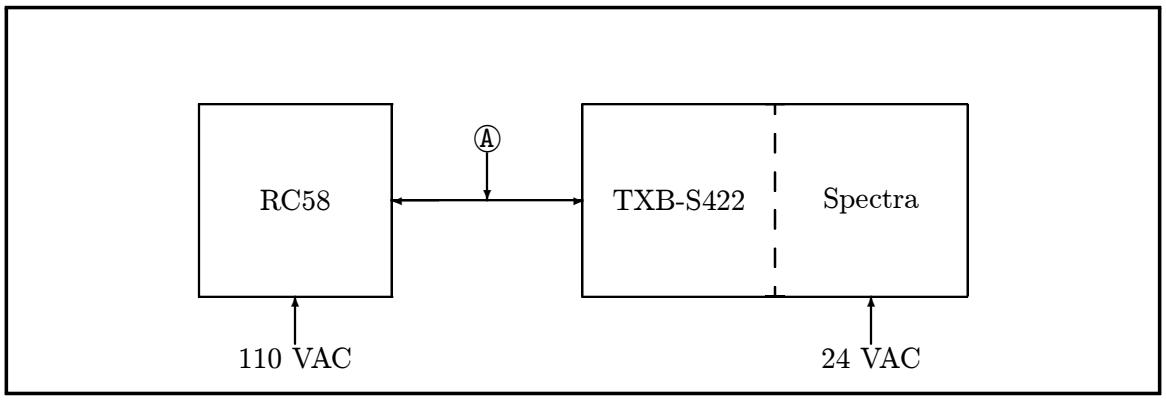
The RC58 will not recognize the presence of the Spectra until the Spectra (with a TXB-S422 installed) has been powered up (or power cycled). Even then it is important to note that the Spectra is only "partially" installed. This means that the dome will receive and process commands

[3] `$Header: d:/txb-s422/RCS/txbs422.inc,v 1.7 2001-11-30 14:25:08-08 Hamilton Exp Hamilton $`

Pin	Name	Name	Use
1	MCLR*	N/A	Master Clear and ICSP
2	RA0	spare2	Spare
3	RA1	ENABLEINPUTS	Input data enable, for head end communications
4	RA2	INVERTIO	Input inversion control
5	RA3	SPECTRAOUT	Serial data out to Spectra
6	RA4	spare6	Spare
7	RA5	SPECTRAIN	Serial data in from Spectra
8	Vss	N/A	+5 VDC
9	OSC1	N/A	Crystal oscillator, 11.0952 MHz
10	OSC2	N/A	Crystal oscillator, 11.0952 MHz
11	RC0	spare11	Spare
12	RC1	spare12	Spare
13	RC2	spare13	Spare
14	RC3	spare14	Spare
15	RC4	spare15	Spare
16	RC5	ENABLEOUTPUTS	Output data enable, for head end communications
17	RC6	UARTOUT	Serial data from UART Out
18	RC7	UARTIN	Serial data to UART In
19	Vss	N/A	+5 VDC
20	Vdd	N/A	Gnd
21	RB0	ADDRESS0	SW-1
	INT	N/A	(Interrupts are not used.)
22	RB1	ADDRESS1	SW-2
23	RB2	ADDRESS2	SW-3
24	RB3	ADDRESS3	SW-4
25	RB4	PERMIT64	SW-5, Allows the TXB-S422 to use address 64.
26	RB5	DEBUGMODEON	SW-6, Permits debug output mode to be entered.
27	RB6	DATAOUT	Spare, debug data output and used for ICSP
28	RB7	spare28	Spare and used for ICSP

Table 3. Pin assignments on the PIC CPU chip

from the controller, but that the controller does not have a completely "normal conversation" with the Spectra.



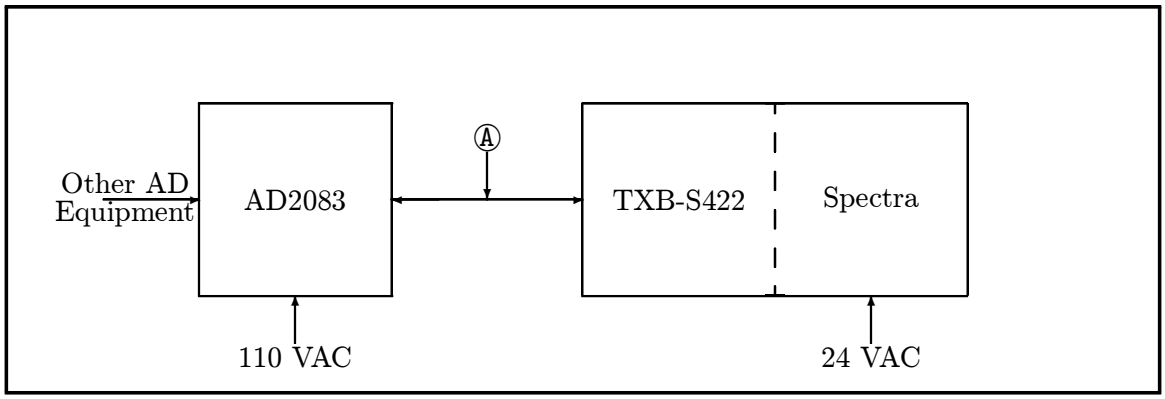
$RCSfile: txbs422.inc,v $

Figure 2. Connection diagram for a Spectra and an RC58

4.2 To an AD2083/02 translator

A Spectra with a TXB-S422 installed in it may be connected to an AD2083/02 (a code translator for an American Dynamics matrix system). This is done by connecting a cable from the T +/-, R +/- connectors. To do this make the following connections at point Ⓐ in Figure 3, page 20:

Spectra		Direction	AD2083/02
P_4	CONTROL		IN/OUT
4	RX —	From Translator	T +
3	RX +	From Translator	T —
	—	Shield	S
2	TX —	To Translator	R +
1	TX +	To Translator	R —



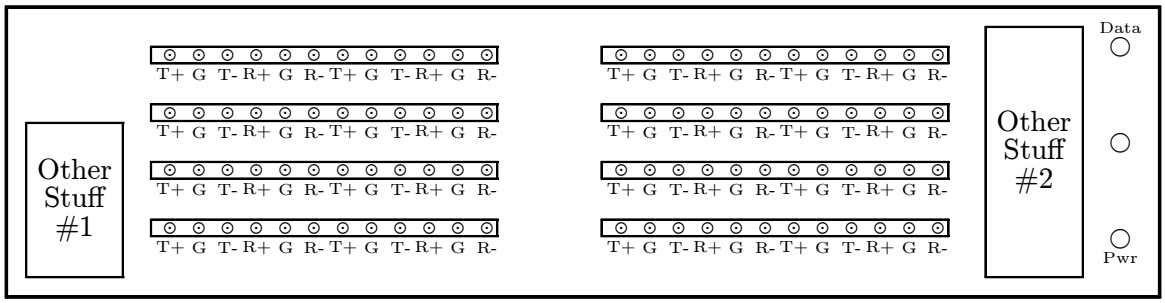
$RCSfile: txbs422.inc,v $

Figure 3. Connection diagram for a Spectra and an AD2083/02

4.3 Using an AD2083/02

The AD2083/02 Data Translator is used to convert American Dynamics "Data Line" information into RS422 messages to control domes, etc. The rear panel is shown in Figure 4, page 21.

Spectra		Direction		AD2083/02
P_4	CONTROL		DB-9	IN/OUT
4	RX +	From Translator	5	T +
3	RX —	From Translator	9	T —
	—	Shield		S
2	TX —	To Translator	1	R +
1	TX +	To Translator	6	R —



$RCSfile: txbs422.inc,v $

Figure 4. AD2083/02 Rear Panel

Other stuff			
#1		#2	
1	Power LED	2	RJ-45 comm in/out jacks
1	Alarm LED	1	12 pin IO connector
1	Code LED	2	BNC code connectors
	—	1	Power cord

5 Program operation

5.1 Initial power up

Hardware resets, master clears, etc., start running at `RESETVECTOR` and immediately go to `init_0`. The code for `init_0`, MUST be in code page 0.

5.2 Runtime execution

System starts runtime execution at `start`. When the initialization code completes, it transfers control to `start`. Any required restarts, including error restarts transfer control to `again`.

At `start` the receiver is enabled. (Selecting 8 or 9 bit mode is done in `init_0`.)

Following the `start` entrance where the receiver is enabled, we have `again` which is the error and general restart location.

Various flags and counters are configured and `getmessage_5` is called to read in a message.

At `decodeinput_M` there is a computed goto subroutine that is called from main and is used to decode commands in the range of `0x80` through `0xFF`.

On entry `w` has the command code as does `sproto2_M`.

Return is made to any of 64, or more, different places.

For several camera functions, the method of "calling" them is to depress several keyboard buttons simultaneously in a specific order. When this happens various special camera operations are activated. If the order changes, or a STOP command (which would indicate that the button was released) comes in, then the special sequence is stopped.

5.2.1 FLIP command processing

A flip command set consists of a FAST (`0x8D`) command immediately followed by a FASTEST (`0x8E`) command. Some controllers also send out a trailing pair of FAST STOP (`0x8F`) commands but they may be ignored as other controllers don't send them out at all!

5.3 Decoding commands

To shorten the decode table a trick has been used. In observing the commands sent by a Sensormatic controller, almost every op-code from `0x80` → `0xAF` is used. No command op-codes of less than `0x80` have been observed and at `0xC0` and above the usage is sparse.

Thus all values from `0x80` → `0xAF` are decoded with a "computed goto" construct. No decodes are made for op-codes of less than `0x80` and at `0xC0` and above, decoding is done on an op-code by op-code basis.

Command decoding for commands in the range of `0x80` → `0xBF`.

Command	Value	Comments
`unknown0x80`	`0x80`	Unknown three byte command
`panleft`	`0x81`	Pan left (24^o) until Pan Right or Pan Stop
`panright`	`0x82`	Pan right (24^o) until Pan Left or Pan Stop
`stoppan`	`0x83`	Stop panning
`tiltup`	`0x84`	Tilt Up until Tilt Down or Tilt Stop
`tiltdown`	`0x85`	Tilt Down until Tilt Up or Tilt Stop
`stoptilt`	`0x86`	Stop tilting
`focusnear`	`0x87`	Focus near until Focus Far or Focus Stop
`focusfar`	`0x88`	Focus far until Focus Near or Focus Stop
`stop`	`0x89`	Stop Focus
`zoomin`	`0x8A`	Zoom in until Zoom Out or Zoom Stop
`zoomout`	`0x8B`	Zoom out until Zoom In or Zoom Stop
`stop`	`0x8C`	Stop zoom
`fast`	`0x8D`	Increase pan and tilt speeds to 48^o
`fastest`	`0x8E`	Increase pan and tilt speeds to 96^o
`faststop`	`0x8F`	Stop fast/fastest speeds (back to normal 24^o)
`irisopen`	`0x90`	Opens iris
`irisclose`	`0x91`	Closes iris
`stop`	`0x92`	Stop iris offset adjustment
`allstop`	`0x93`	Stop all movement
`getdometype`	`0x94`	Request dome type (poll)
`getalarms`	`0x95`	Request status of alarm inputs
`unknown`	`0x96`	Undetected unknown command
`ack`	`0x97`	ACKnowledge sent to dome
`unknown`	`0x98`	Unknown three byte command
`unknown`	`0x99`	Unknown three byte command
`faster`	`0x9A`	Increase pan and tilt speeds to 72^o
`fasterstop`	`0x9B`	Stop faster speeds (back to normal 24^o)
`boundarystar`	`0x9C`	Start boundary definition.
`boundarymark`	`0x9D`	Marks the current position as a boundary
`onair`	`0x9E`	Set On Air status to tell the dome to send the
`onairreset`	`0x9F`	Reset On Air status
`pattern1`	`0xA0`	Start defining Pattern 1
`pattern2`	`0xA1`	Start defining Pattern 2

Continued on the next page.

Continued from the previous page.		
Command	Value	Comments
pattern3	0xA2	Start defining Pattern 3
patternaccep	0xA3	Accept the new pattern
unknown	0xA4	Unknown command
getposition	0xA5	Request Dome position Coordinates
gotoposition	0xA6	Goto absolute position (Multiple-byte format)
unknown	0xA7	Undetected unknown command
mark1	0xA8	Mark the current position as Target 1
mark2	0xA9	Mark the current position as Target 2
mark3	0xAA	Mark the current position as Target 3
mark4	0xAB	Mark the current position as Target 4
unknown	0xAC	Unknown command
unknown	0xAD	Unknown command
unknown	0xAE	Unknown command
unknown	0xAF	Unknown command
run1	0xB0	Run Pattern 1
run2	0xB1	Run Pattern 2
run3	0xB2	Run Pattern 3
runreview	0xB3	Run a newly defined pattern to review it
target1	0xB4	Go to Target 1
target2	0xB5	Go to Target 2
target3	0xB6	Go to Target 3
target4	0xB7	Go to Target 4
patternend	0xB8	Tells the dome to stop recording (defining)
mark5	0xB9	Mark the current position as Target 5
mark6	0xBA	Mark the current position as Target 6
mark7	0xBB	Mark the current position as Target 7
target5	0xBC	Go to Target 5
target6	0xBD	Go to Target 6
target7	0xBE	Go to Target 7
unknown	0xBF	Undetected unknown command

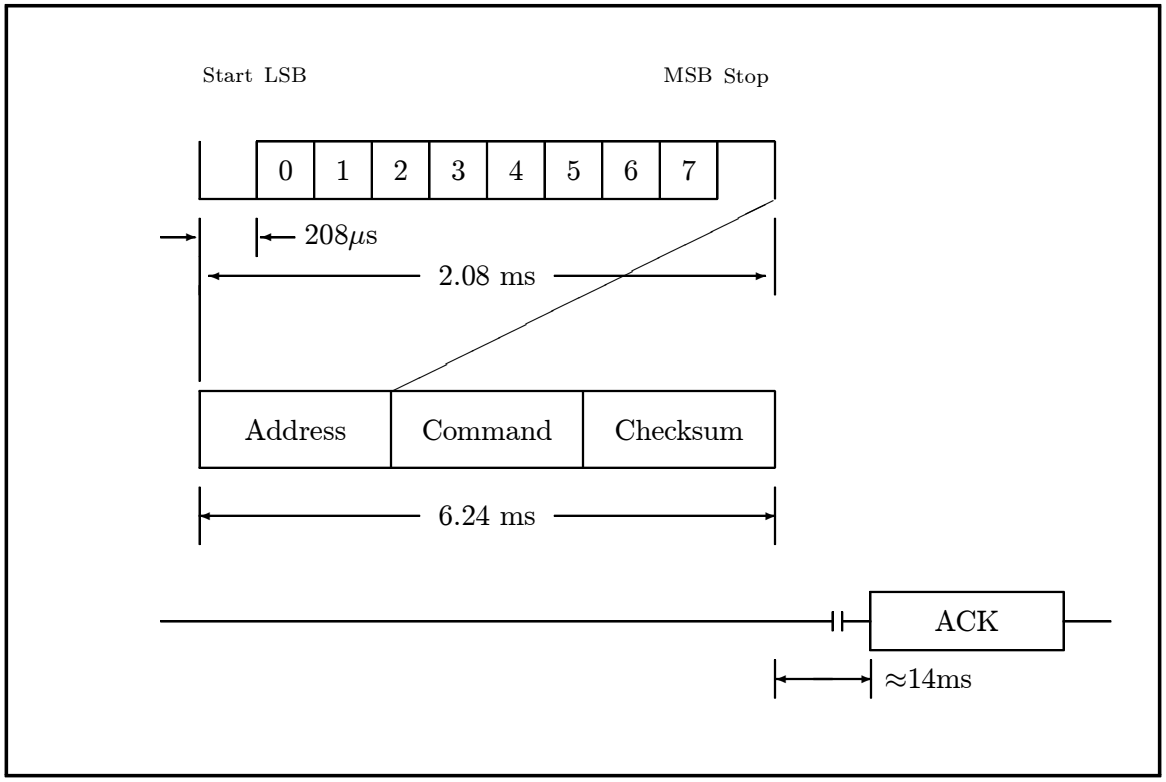
Format of the S protocol communications are:

[4] $Header: d:/txb-s422/RCS/sprotoch.inc,v 1.2 2001-11-30 15:12:06-08 Hamilton Exp Hamilton $

Table 1. Protocol characteristics

Data Rate	4.8 kbits/sec
Data Format	Start bits: 1 Data bits: 8 Parity bits: 0 Stop bits: 1
Format	3 bytes: Dome Address Command Checksum
Max devices per line	10 (Depends on device loading)
Cable type	2 shielded, twisted pair.
Wire gauge	22 AWG*
Max. length	1 km (3.28?)
Connection	Polarized
Data and Power Connectors Color-Code Conventions	Color Designation Black 24 VAC Red Ground White 24 VAC Orange RS-422 Data in High (+) Green RS-422 Data in Low (—) Yellow RS-422 Data out High (+) Brown RS-422 Data out Low (—)

∗ Sensormatic composite cable is recommended. This cable also contains wires for power and video. In another cable is substituted, cable ??? colors may be different.



$RCSfile: 3bytfmt.inc,v $

Figure 5. 3-byte format

For all commands that do not have special processing, the appropriate bits are set in a D protocol command and it is sent to the Spectra.

The Sensormatic protocol sends new commands almost immediately after it receives an ACK, or it retransmits the last command about 45 ms after starting a command (time information is from measurements made on an AD2083/02). Thus it is necessary to send an ACK, AFTER sending a command to the Spectra. Remember that when sending data to the Spectra we have to bit-bang the data out and thus can not receive any new data in from the head end while sending the Spectra data.

Having a 29.6 ms "dead window" makes it so that we miss many messages. However by delaying the ACK until AFTER sending a command to the Spectra, means that we almost always "get" all commands from the controller.

WARNING: If any controller retransmits commands faster than the AD2083/02 does there may be a problem. Currently there is about 9 ms of "slack", but different controllers may retransmit faster, etc.

Program logic used to be to send the ACK first and then do the rest.

5.4 Pan and Tilt variable speed processing

Byte	Use
1	Camera ID
2	Op Code
3	Sub-Op Code
4	Speed
5	Checksum

There are several types of keyboards that generate variable speed commands. The one from American Dynamics generates a total of eight (8) different speeds, while others generate more. Thus we have to identify the matrix/keyboard type in order to determine what to do about converting the input speeds into output speeds. If we ever get in a pan speed of `0x01`, `0x02`, `0x50` or `0x63`, the program logic will assume that it is not talking with an American Dynamics system and that it is talking to a Sensormatic system. Preliminary testing shows that the RC216 sends out the earlier speeds and that the AD2083/02 does not. This process of testing is not guaranteed to work forever but is the best that I can do to tell the difference between command sources. (Of course the RC58 system only sends fixed speed commands.)

5.4.1 AD2083/02 speed decoding

With the AD2083/02 code translator, there are only eight variable speeds available. These have to be translated into something that is reasonable for the Spectra to use. Unfortunately the data comes out in degrees/sec (?) while Pelco uses a different system to get up to 64 different speeds. So here we have to determine which of the eight speed values are being sent and then translate these values into Spectra equivalents. If a careful examination is made of the values sent from the AD translator, it will be observed that the values are highly non-linear. However Pelco's D protocol uses nice linear steps.

Pan Speeds		
Observed speeds	Decision point	Generated speeds
0x04	0x05	7/0x07
0x06	0x08	15/0x0F
0x0A	0x0C	23/0x17
0x0F	0x13	31/0x1F
0x18	0x1C	39/0x27
0x21	0x27	47/0x2F
0x2D	0x43	55/0x37
0x5A and other	0x43+	64/0x40 Or turbo speed

Tile Speeds		
Observed speeds	Decision point	Generated speeds
0x03	0x04	3/0x03
0x05	0x07	11/0x0B
0x09	0x0B	19/0x13
0x0D	0x10	27/0x1B
0x14	0x16	35/0x23
0x18	0x1C	43/0x2B
0x21	0x27	51/0x33
0x2D and higher	0x27+	63/0x3F Or max tilt speed

In this above table the column marked "Decision point" indicates at this value, all lower speeds are translated into the values shown in the "Generated speeds" column. Remember that the lower values are checked first.

It should also be noted that Pelco pan speeds start changing at 10. (I.e. all values less than 10 give the same slow pan rate.) This was done for compatibility with older keyboards. There are also several speeds that get "jumped" over to avoid mechanical resonances noise in the individual Spectra its self. (And the skipped numbers change with different rev levels of the Spectra software. I don't really want to mention what happens with the Esprit.) The net result of all this is that we have less than 64 speeds available and it is impossible to predict what will happen in any given situation.

5.4.2 RC216 speed decoding

The RC216, and possibly others, generates many different pan speed commands, at least 30. So the problem here is to translate the non-linear Sensormatic commands to the non-linear Pelco commands.

In preliminary testing the following variable pan speeds were observed: 1 → 20, 22, 24, 32, 48, 64, 80 and 99.

These speeds represent dome speeds in different degrees per second. There is a potential for having 99 different speeds while Pelco only has about 53 different speeds. (It must be remembered

that some speeds are repeated inside the Spectra and others are blocked to avoid mechanical resonance induced noise.)

Taking advantage of the fact that the Spectra repeats the first nine pan speed values we can simplify the conversion process quite a lot by always dividing the input value by 2 (the input range is now $0 \rightarrow 48$) and then adding eight to it giving a low Spectra speed of 0.5^o/sec and a high speed of of 41.9^o/deg per second. As this last speed is a little slow (it's NOT turbo speed) we do a special check to see if the input speed is `0x63`, or more, which we translate into a turbo pan speed of `0x40`.

For tilt processing, the differences are that we check to see if the tilt value is so big that we will go past `0x3F` as a maximum tilt speed which is illegal. We then divide the input by 2 and add 6 to get over the initial repeated tilt speeds in a Spectra.

5.5 Presets

5.5.1 Presets from an AD2083/02

Presets are sent three times with the following multibyte format. In this format only bytes 1, 2, 4 and 13 are used by the TXB-S422.

Byte	Use	Contents
1	Camera ID	Camera ID
2	Op Code	`0xA6`
3,4	Pan Position	`0x33`, Preset ID
5,6	Tilt Position	`0x06`, Preset ID
7,8	Zoom Position	`0x26`, Preset ID
9,10	Digital Zoom	`0x28`, Preset ID
11	Iris Offset	`0x00`
12	Zoom Limit	`0x00`
13	Checksum	Checksum

5.6 Multibyte responses

There are two "long" responses to deal with. They are:

software version 10 byte response `0xA5`

preset position 12 byte response `0xC9`

5.6.1 Preset Position Message

First an ACK is sent then the response is:
All hex fields are common to all position requests.

Byte	Use	Contents
1	Camera ID	Camera ID
2	Iris Data	0x00
3	Zoom Limit	0x00
4,5	Tilt Position	0x06, Preset ID
6,7	Zoom Position	0x26, Preset ID
8,9	Electronic Zoom	0x28, Preset ID
10,11	Pan Position	0x33, Preset ID
12	Checksum	Checksum

Values that get stuck in the reply message consist of the preset ID and various constants from a real breakout dump from dome #5. If there are any "strange" value range checks made, this should confuse them and get the controller to pass the values as OK.

5.6.2 Version Message

Response is:

Byte	Use	Contents
1	Camera ID	Camera ID
2	Op Code	0xC9
3	Unknown byte	0x06
4 –¿ 9	Software rev	0x07, 0x01, 0x00, 0x01, 0x03, 0x16
10	Checksum	Checksum

This is the reply made by dome #5 which should match the supplied type code of 0xF5. The type code comes from define PELCODOME.

5.7 Aux command processing

Format of an output command is that the four LSBs of the opcode indicate what to to with the AUXes. Thus we have:

Where f = off, N = on.

AUXes	3	2	1	0	AUXes	3	2	1	0
0xE0	f	f	f	f	0xE8	N	f	f	f
0xE1	f	f	f	N	0xE9	N	f	f	N
0xE2	f	f	N	f	0xEA	N	f	N	f
0xE3	f	f	N	N	0xEB	N	f	N	N
0xE4	f	f	f	f	0xEC	N	N	f	f
0xE5	f	f	f	N	0xED	N	N	f	N
0xE6	f	N	N	f	0xEE	N	N	N	f
0xE7	f	N	N	N	0xEF	N	N	N	N

To properly process this command we have to send a total of four commands. One each for each bit which gets turned on or off. In protocol D we can only specify one bit to turn on or off with each command. The following table illustrates which command matches each bit:

`0x09`, DSETAUXILIARY

`0x0A`, DCLEARAUXILIARY

`0x01`	Aux 1	`0xE1`
`0x02`	Aux 2	`0xE2`
`0x03`	Aux 3	`0xE4`
`0x04`	Aux 4	`0xE8`

5.8 Commands that are not processed or are ignored

RESET	`0xC6`	Run default "Apple Peel" pattern.
ACK	`0x97`	ACKnowledge sent to dome.
BOUNDARYSTART	`0x9C`	Start boundary definition.
BOUNDARYMARK	`0x9D`	Marks the current position as a boundary.
RUNREVIEW	`0xB3`	Run a newly defined pattern to review it.
unknown	—	Processing for all unknown commands. So far the list contains: `0x96`, `0x98`, `0x99`, `0xA4`, `0xA7`, `0xAC`, `0xAD`, `0xAE`, `0xAF`, `0xBF`, `0xC2`, `0xC4`, `0xC5`, `0xC7`, `0xC8`, and all commands `0xCA` → `0xDF`, `0xF0` → `0xFF`

5.9 Commands that get an acknowledge only

ONAIR	`0x9E`	Set On Air status
ONAIRRESET	`0x9F`	Reset On Air status
PATTERNACCEPT	`0xA3`	Accept the new pattern
UNKNOWN	`0x80`	

6 System subroutines

There are several "level's" of subroutines. The level # indicates its level. A subroutine may only call lower level routines, NEVER those at its level or above. As a convention all subroutine names end with their calling level. Subroutines that end with `_M` are main line extensions.

6.1 `arrowout_5`

Is used to stick an arrow out in debug mode.

6.2 `bin2hex_0`

Is used to take a binary byte in the `w` register and to convert it into two ASCII bytes. The upper half will be in `hexupper_0` and the lower half will be in `hexlower_0`. (What surprising places to stuck um.) No additional core locations are used.

6.3 `bitdelay_0`

Is used to sit in a tight loop for one bit time. How long to wait is contained in `w` on entry.

6.4 `blank_2`

Is used to output a blank character in `debug` mode.

6.5 `blankit_3`

Is used to clear the Spectra screen.

6.6 `byteread_1`

Is used to immediately read a byte and return with it in `w`. This routine is intended to sandwich reads into the middle of writing to the Spectra. Before calling `BYTE1` and `BYTE2` must be cleared. The received message is limited to three bytes and is stored in `sproto1_M` thru `sproto3_M`. Messages longer than 3 bytes are messed up and not saved correctly. (Had to make this a level 1 subroutine, instead of putting it into `sendspectra_0` because I'm worried about not having enough levels of return stack.) A check is made of what might be the first byte to be sure that it matches "our" Spectra's address. If the address check fails, we don't advance past the first byte.

6.7 `ckioerrs_0`

Is used to check the head end serial connection for errors and to get the data.
On return `w` holds the data byte. Also `temp1_0` holds the most recent byte.

6.8 `cleareememory_1`

Is used to see if EE memory has been cleared. It is assumed that it is not cleared if the 32 bit constant `0xDEADBEEF` is not in locations `EESETMEM0` through `EESETMEM3`. Anything else will result in three locations being set as follows:

Location	Initial value
`EEPRESET`	`0`
`EEPANSPEED`	`DEFAULTPANSPEED`
`EETILTSPEED`	`DEFAULTTILTSPEED`

6.9 `crlf_2`

Is used in debug mode to stick out a new-line sequence.

6.10 `delaynohang_0`

Is used to eliminate hangups while waiting for an IO event to finish. It is expected that this routine will be entered with RAM bank 1 selected, RAM bank 1 will be selected on return. Note that no effort is made to make this fast, after all we want some kind of delay to be done here and the longer the better.

Timing is as follows: about 5.79 us per call (16 * .361 us)

Uses `temp2_0` which should be set to zero before the first call. Is counted up to zero for the delay.

On return : `c` = 0 not timed out

`c` = 1 has timed out

6.11 `delayspectra_0`

Is used in receiving data from the Spectra. It delays for about one half of a bit time, at 2400 baud, so that sampling may occur in the middle of each bit period.

6.12 `delayv_0`

Is used to delay 250 μs * value in `w`. Each instruction step takes about .361898148 μs to complete.

6.13 `doaux_3`

Is used to send an AUX command to the Spectra.

`w` = which AUX to configure

`dproto4_M` = A SET or CLEAR AUX command

6.14 docheckSum_0

Is used to sum up the contents of a transmit buffer and stick the results in the command.

6.15 eeread_0

Is used to read a byte from EEPROM memory.
When called w has the address to read from.
On return w has the addresses contents.
(This routine has been copied from the Microchip data sheet for the PIC16F87x, DS30292C, page 43.)
This routine uses other than RAM bank 0.

6.16 eewrite_0

Is used to write a byte into EEPROM memory.
On entry w has the byte to be written.
eewritehere_0. has the address in it to write into.
(This routine has been copied from the Microchip data sheet for the PIC16F87x, DS30292C, page 43.)
eewrite_0 saves the currently written value in temp1_0 and does not change it.
This routine uses other than RAM bank 0.

6.17 findspectraedge_0

Is used to detect when a transition occurs in the data from the Spectra at 2400 baud.
Returns with w set to the value of the transition.
0 = low going transition
1 = high going transition
Note that this is an unusual routine in that it has two different exit points. For each bit, one or zero, there are different exits. Once checking for a transition, the check occurs about once every 1 us or so.

6.18 flipinput_0

Is used to alternate the input flipping logic.

6.19 fromspectra_1

Is used to read three bytes from the Spectra. Since there is no UART available to do this, we have to bit-bang to get the message in. Normally this is used to get just the Spectra's address which requires reading only the second byte. It has been modified to now read in three bytes so that the alarm status may be read. In all cases the checksum is ignored.
Received data format is:

Non return to zero (NRZ)
Least significant bit comes in first
High is a one
Low is a zero
Quiescent is high
RS-422 voltage levels
One start bit which goes low
Eight data bits
One stop bit which goes high or is already high depending on the data
No parity

First byte is a sync byte of all ones.
Second byte is the Spectra's address.
Most of the rest is the software PGM number.
Last byte is the checksum of everything else.

6.20 `getmessage_5`

Is used to read in a controller message. Is only to be called when the receive UART has something in it.

6.21 `getsensorbyte_1`

Is used to get in a byte of keyboard data.

This is where the program normally sits while waiting for input from the head end. When an input comes in, then it is processed and we return here.

6.22 `idlewait_0`

Is used to identify idle places in the data. Data coming from the Sensormatic controller has no "first byte" identifier. So we have to wait until there is an idle space between each message. Monitoring `RAWDATAIN` we get the actual input voltage is complicated by having the actual data go both high and low. So we must wait for at least one byte of all "high's" with mostly high values. At 4800 baud, each 10 bit (8 data, 1 start, 1 stop) byte takes 20.83 ms (10 bits * (1/4800 baud) = 20.83 ms), to come in. However we must account for some level of noise. So we count the number of highs and lows, to be considered "always high" we must get at least 90% highs in the test period.

6.23 `inlabel_3`

Is used to help in overwriting the Spectra's "`CONFIGURATION DONE`" message with our "`TXB-S422 Rev X.XX AA`" message all on line 2

Call with `w` holding the character to stick into the message. Returns with the message updated for location, a new character stuck in the Spectra buffer and the buffer sent to the Spectra.

6.24 presetincrement_1

Is used to get the next preset value from EEPROM and to update the highest preset number saved in EEPROM.

On return w has the new preset value.

Range of stored presets is $0 \rightarrow 63$, but range of Spectra presets is $1 \rightarrow 64$. So on getting the preset it is always incremented by 1.

Note that it takes $4 \rightarrow 8$ ms to complete an EEPROM write.

6.25 printsensorin_4

Is used to printout the received command, in debug mode.

6.26 printspectraout_4

Is used to printout the D protocol command, in debug mode.

6.27 send2hex_3

Is used to print out a pair of converted hex digits.

Enter with the value to printed out in w.

6.28 sendack_6

Is used to print out an arrow to mark what are ACKs.

6.29 senddebugbyte_1

Is used to transmit one byte of information from the w register to the serial output port a single bit at a time. The assumed byte characteristics are: DEBUGPERIOD baud, one start bit, two stop bits, eight data bits and no parity. (RS-232 is negative true logic with the least significant bit being shifted out first.) It is important to note that this "RS-232" does not go thru a level shifter and thus is exactly what the receiving unit receives. I know that RS-232 is not specified as having voltage levels of 0 and +5, but it does work for short distances and the only time that this is used, is in debugging. Thus it's OK to do it.

6.30 sendspectrabyte_0

Is used to transmit one byte of information from the w register to the serial output port a single bit at a time. The assumed byte characteristics are: 2400 baud, one start bit, two stop bits, eight data bits and no parity. (RS-422 is positive true logic with the least significant bit being shifted out first.)

6.31 `startout_1`

Is used to delay and then enable the controller output driver.

6.32 `threebytemessage_6`

Is used to send a three byte message to the controller
On entry `w` has the value to send

6.33 `tosensormatic_4`

Is used to send bytes to the Sensormatic controller.
`w` = what to send

6.34 `tospectra_2`

Is used to send a D protocol command to a Spectra.

A special problem is fixed with this routine. That problem is that it takes so long to send a Spectra command in D protocol at 2400 baud (29.6 ms) that we miss commands from the Sensormatic controller. (Remember that there is only one UART on this baby and that we have to bit-bang data out to the Spectra and can't do anything else or the timing gets messed up.) The solution to this is to check the receive register after each transmitted byte and to store the data if any data comes in. If `BYTE3` is set then all three bytes have been received and we can exit. Conveniently enough the UART logic on the PIC chip is somewhat more than double buffered, i.e. there are two holding registers (in a FIFO) arrangement in addition to the input shift register. Since the data comes in at 4800 baud and we send data out to the Spectra at 2400 baud, if we check for received data after each sent byte, we should never "loose" a three byte message.

7 Initialize

7.1 IO ports first

In this program the standard RAM bank is bank 0. There will be some changes to RAM bank 1, but these will be rare and when this happens the is marked in the right hand column of the source listing with the current RAM bank #.

Do much of RAM bank 1 setup first, then select RAM bank 0 for the rest of setting up.

The A/D converter inputs are disabled by sending a command of 00000110_2 to `adcon1`.

Chip registers that get cleared include all of the following:

`ccp1con`	Clear out the capture/compare control register.
`ccp2con`	Do both capture/compare control registers.
`intcon`	Disable all interrupts, `intcon` is in both RAM banks.
`pcon`	Aren't using the brown out logic, so clear bits.
`pie1`	Interrupts to disable.
`pie2`	Last interrupt to disable in RAM bank 1.
`pir2`	Last interrupt to disable in RAM bank 0.
`porta`	IO port A.
`portb`	IO port B.
`portc`	IO port C.
`sspcon`	Clear synchronous serial control register.
`t2con`	Clear out the timer #2's controls.

Setups for `porta`, a six bit port:

Bit settings 7 6 5 4 3 2 1 0	Bit #	Comments
0	7	x (IO pin does not exist)
0	6	x (IO pin does not exist)
1	5	Serial data from Spectra
0	4	Spare
0	3	Serial data to Spectra
0	2	Used to invert input/output data (not used)
0	1	RS-422/485 input enable
0	0	spare

Setups for `portb`, an eight bit port:

Bit settings 7 6 5 4 3 2 1 0	Bit #	Comments
0	7	ICSP and spare otherwise
0	6	ICSP and debug output
1	5	Debug enable
1	4	Permit using address 64
1	3	Address bit 3
1	2	Address bit 2
1	1	Address bit 1
1	0	Address bit 0

Setups for `portc`, an eight bit port:

Bit settings 7 6 5 4 3 2 1 0	Bit #	Comments
1	7	Input data from the head end, via the USART
0	6	Output data to the head end, via the USART
0	5	RS-422/485 output enable
0	4	spare
0	3	spare
0	2	spare
0	1	spare
0	0	spare

Setups for `tmr0`, the timer port:
The timer is set to the instruction counter with no prescale.

Bit settings 7 6 5 4 3 2 1 0	Bit #	Comments
1	7	Disable portb pull-ups
0	6	Interrupt on falling edge
0	5	`tmr0` clock source is instruction cycle clock
0	4	Pos transition
0	3	Prescaler to Timer0
0 0 0	012	Prescaler is set to 1:2

Setups for the Peripheral Interrupt Register 1, `pir1`:

Name	Bit settings 7 6 5 4 3 2 1 0	Bit #	Comments
—	0	7	Reserved
—	0	6	Reserved
rcif	0	5	Receive Interrupt flag
txif	0	4	Transmit Interrupt flag
sspif	0	3	Synchronous Serial Port Interrupt Flag
ccp1if	0	2	Capture flag
tmr2if	0	1	TMR2 flag
tmr1if	0	0	TMR1 flag

Setups for timer 1, `t1con`:

Name	Bit settings 7 6 5 4 3 2 1 0	Bit #	Comments
unused	0	7	Unimplemented
unused	0	6	Unimplemented
t1ckps1	1	5	Timer 1 input clock prescale select bit
t1ckps0	1	4	. . prescale by 8
t1oscen	0	3	Oscillator on/off with off selected
t1sync	0	2	Ignored if `tmr1cs` = 0
tmr1cs	0	1	Use internal/external clock, internal used
tmr1on	0	0	Start/stop timer, 1 = start, 0 = stop

Setups for USART, transmit first, `txsta`:

Name	Bit settings 7 6 5 4 3 2 1 0	Bit #	Comments
csrc	0	7	Synchronous mode clock select, async = nop
tx9	0	6	0 = 8 bit transmit mode, 1 = 9
txen	1	5	Transmit enable = 1, set to 1 to start transmit
sync	0	4	Asynchronous mode = 0, sync = 1
—	0	3	Spare, unimplemented
brgh	0	2	Low speed mode for the baud rate generator
trmt	0	1	Status of transmit register
tx9d	0	0	9th bit transmitted, spare in 8 bit mode

The baud rate generator is set up to operate at 4800 baud (`spbrg`).

Setups for the USART, now the receive stuff, `rcsta`

Name	Bit settings 7 6 5 4 3 2 1 0	Bit #	Comments
`spen`	1	7	Serial port enable
`rx9`	0	6	8 bit receive mode = 0, 1 = 9
`sren`	0	5	Sync mode receive bit, async = nop
`cren`	1	4	Asynchronous enable continuous receive
—	0	3	Spare
`ferr`	0	2	Framing status error bit
`oerr`	0	1	Overrun status error bit
`rx9d`	0	0	Parity bit, not used in 8 bit mode

7.2 Get the Spectra's address

Get the Spectra's address. This is done by sending a query command out to the Spectra. Then waiting until the first byte comes in. The first byte is thrown away and the second byte is used as the Spectra's address.

First wait until the Spectra is working.

Now send a query command so as to get the Spectra's address.

APPENDIX A

A Sensormatic protocol definations sdefs.inc

```
; "$Header: d:/txb-s422/RCS/sdefs.inc,v 1.17 2002-02-27 15:58:31-08 Hamilton
Exp Hamilton $"
; Definitions to use with Sensormatic's RS422 protocol
; Copyright by Pelco, 2001, 2002
        nolist
        ifdef listincludes
         list
        endif ; listincludes

; Commands to the dome
;
#define S_UNKNOWN80        0x80 ; Unknown three byte command
#define S_PAN_LEFT         0x81 ; Pan left (24/sec) until Pan Right or Pan
                                ; Stop
#define S_PAN_RIGHT        0x82 ; Pan right (24/sec) until Pan Left or Pan
                                ; Stop
#define S_PAN_STOP         0x83 ; Stop panning
#define S_TILT_UP          0x84 ; Tilt up until Tilt Down or Tilt Stop
#define S_TILT_DOWN        0x85 ; Tilt Down until Tilt Up or Tilt Stop
#define S_TILT_STOP        0x86 ; Stop tilting
#define S_FOCUS_NEAR       0x87 ; Focus near until Focus Far or Focus Stop
#define S_FOCUS_FAR        0x88 ; Focus far until Focus Near or Focus Stop
#define S_FOCUS_STOP       0x89 ; Stop Focus
#define S_ZOOM_IN          0x8A ; Zoom in until Zoom Out or Zoom Stop
#define S_ZOOM_OUT         0x8B ; Zoom out until Zoom In or Zoom Stop
#define S_ZOOM_STOP        0x8C ; Stop zoom
#define S_FAST             0x8D ; Increase pan and tilt speeds to 48/sec
                                ; until Fast Stop
#define S_FASTEST          0x8E ; Increase pan and tilt speeds to 96/sec
                                ; until Fast Stop
#define S_FAST_STOP        0x8F ; Stop fast/fastest speeds (back to normal
                                ; 24/sec)
#define S_IRIS_OPEN        0x90 ; Opens iris (manual iris mode/lightens
                                ; Iris Preference offset (auto iris mode)
                                ; until Iris Close or Iris Stop
#define S_IRIS_CLOSE       0x91 ; Closes iris (manual iris mode/darkens
                                ; Iris Preference offset (auto iris mode)
                                ; until Iris Open or Iris Stop
```

```
38 #define S_IRIS_STOP        0x92 ; Stop iris offset adjustment (also stops
39                                 ; V-Phase Adjust)
40 #define S_ALL_STOP         0x93 ; Stop all movement
41 #define S_GET_DOME_TYPE    0x94 ; Request dome type or poll
42 #define S_GET_ALARMS       0x95 ; Request status of alarm inputs
43 #define S_ACK              0x97 ; ACKnowledge sent to dome responds to
44                                 ; asynchronous commands.
45 #define S_UNKNOWN98        0x98 ; Start temp no transmit
46 #define S_UNKNOWN99        0x99 ; End temp no transmit
47 #define S_FASTER           0x9A ; Increase pan and tilt speeds to 72/sec
48                                 ; until Fast Stop
49 #define S_FASTER_STOP      0x9B ; Stop faster speeds (back to normal 24/sec)
50 #define S_DEFINE_BOUNDARY  0x9C ; Start boundary definition. This command
51                                 ; is followed by dome movement commands and
52                                 ; tour Mark Boundary commands.
53 #define S_MARK_BOUNDARY    0x9D ; Marks the current position as a boundary
54 #define S_ON_AIR           0x9E ; Set On Air status to tell the dome to
55                                 ; send the asynchronous boundary crossing
56                                 ; command
57 #define S_ON_AIR_RESET     0x9F ; Reset On Air status
58 #define S_DEFINE1          0xA0 ; Start defining Pattern 1
59 #define S_DEFINE2          0xA1 ; Start defining Pattern 2
60 #define S_DEFINE3          0xA2 ; Start defining Pattern 3
61 #define S_NEW_PATTERN      0xA3 ; Accept the new pattern as the current
62                                 ; pattern and delete the old pattern.
63 #define S_DUMP_DOME_MEMORY 0xA4 ; Dump dome memory
64 #define S_GET_POSITION     0xA5 ; Request Dome position Coordinates
65 #define S_GOTO_POSITION    0xA6 ; Goto absolute position (Multiple-byte
format)
66 #define S_MARK1            0xA8 ; Mark the current position as Target 1
67 #define S_MARK2            0xA9 ; Mark the current position as Target 2
68 #define S_MARK3            0xAA ; Mark the current position as Target 3
69 #define S_MARK4            0xAB ; Mark the current position as Target 4
70 #define S_GOTO_PAT1        0xAC ; Go to the start of pattern 1
71 #define S_GOTO_PAT2        0xAD ; Go to the start of pattern 2
72 #define S_GOTO_PAT3        0xAE ; Go to the start of pattern 3
73 #define S_GOTO_PAT4        0xAF ; Go to the start of pattern 4
74 #define S_RUN1             0xB0 ; Run Pattern 1
75 #define S_RUN2             0xB1 ; Run Pattern 2
76 #define S_RUN3             0xB2 ; Run Pattern 3
77 #define S_RUN_NEW          0xB3 ; Run a newly defined pattern to review it
78                                 ; before accepting it to replace the
```

```
79                                  ; previous pattern.
80 #define S_GOTO1            0xB4 ; Go to preset position called Target 1
81 #define S_GOTO2            0xB5 ; Go to preset position called Target 2
82 #define S_GOTO3            0xB6 ; Go to preset position called Target 3
83 #define S_GOTO4            0xB7 ; Go to preset position called Target 4
84 #define S_PATTERN_END      0xB8 ; Tells the dome to stop recording
85                                  ; (defining) a pattern
86 #define S_MARK5            0xB9 ; Mark the current position as Target 5
87 #define S_MARK6            0xBA ; Mark the current position as Target 6
88 #define S_MARK7            0xBB ; Mark the current position as Target 7
89 #define S_GOTO5            0xBC ; Go to preset position called Target 5
90 #define S_GOTO6            0xBD ; Go to preset position called Target 6
91 #define S_GOTO7            0xBE ; Go to preset position called Target 7
92 #define S_VARIABLE_SPEED   0xC0 ; Variable speed control (New command.
93                                  ; Multiple-byte format)
94 #define S_UNKNOWN_C1       0xC1 ; Unknown
95 #define S_PROP_SPEED       0xC3 ; Proportional speed pan or tilt movement
96                                  ; commands (Multiple-byte format)
97 #define S_PROP_LEFT        0x81 ; Subcommand of 0xC3, pan left
98 #define S_PROP_RIGHT       0x82 ; Subcommand of 0xC3, pan right
99 #define S_PROP_UP          0x84 ; Subcommand of 0xC3, tilt up
100 #define S_PROP_DOWN        0x85 ; Subcommand of 0xC3, tilt down
101 #define S_UNKNOWN_C4       0xC4 ; Unknown three byte command
102 #define S_RESET            0xC6 ; Run default "Apple Peel" pattern for a
103                                 ; spiral view of everything (only supported
104                                 ; by SpeedDome Ultra IV and DeltaDome, or
105                                 ; later products)
106 #define S_SOFTWARE_VERSION 0xC9 ; Get software version number from dome
107 #define S_OUTPUT0          0xE0 ; Clears all active drivers
108 #define S_OUTPUT1          0xE1 ; Set output Driver #1
109 #define S_OUTPUT2          0xE2 ; Set output Driver #2
110 #define S_OUTPUT3          0xE4 ; Set output Driver #3
111 #define S_OUTPUT4          0xE8 ; Set output Driver #4
112 #define S_TERM_PAT         0xF0 ; Stop/terminate current pattern
113
114         space 2
115 ; Messages from the dome
116
117 #define S_DOME_TYPE_IS     0x94 ; Response to Request Dome Type
118 #define S_BOUNDARY0CROSSED  0xB0 ; Boundary crossing #1
119 #define S_BOUNDARY1CROSSED  0xB1 ; Boundary crossing #2
120 #define S_BOUNDARY2CROSSED  0xB2 ; Boundary crossing #3
```

```
121 #define S_BOUNDARY3CROSSED  0xB3 ; Boundary crossing #4
122 #define S_BOUNDARY_CONFUSION 0xB4; Boundary Confusion (sent by dome if
123                                   ; problem defining boundaries)
124 #define S_PATTERN_DONE     0xB5 ; Pattern Done (sent by dome when it
125                                   ; completes a pattern)
126 #define S_POWERED_UP       0xC1 ; Dome Powered Up (sent by dome to indicate
127                                   ; it has powered up and is on line)
128 #define S_DOME_ALARM0      0xD0 ; Bit 0, Switch 1 (0 = on, 1 = off)
129 #define S_DOME_ALARM1      0xD1 ; Bit 1, Switch 2 (0 = on, 1 = off)
130 #define S_DOME_ALARM2      0xD2 ; Bit 2, Switch 3 (0 = on, 1 = off)
131 #define S_DOME_ALARM3      0xD3 ; Bit 3, Switch 4 (0 = on, 1 = off)
132
133         list
134 ; End of SDEFS.INC
135
```

APPENDIX A

A D protocol definations dproto.inc

```
 1 ; "$Header: d:/txb-s422/RCS/dproto.inc,v 1.23 2002-02-28 12:57:21-08 Hamilton
Exp Hamilton $"
 2 ; Copyright by Pelco, 2000, 2001, 2002
 3         nolist
 4         ifdef LIST_INCLUDES
 5          list
 6         endif ; LIST_INCLUDES
 7 ;
 8 ; To help with this long boring process, I have put together a bunch of
 9 ; macros which specify various bits in various words.  For example:
10 ;
11 ;        D_FOCUS_FAR is defined to be d_command2_M,bit7
12 ;
13 ; So all that has to be done is to see if a bit is set and then work with
14 ; it.
15 ;
16 ; All names involved in this start with a D for D protocol.
17 ; Defined names are uppercase, buffer names and values are in
18 ; lower case (usually).
19 ;
20 ; Defines to control which bits get set for coding commands in D protocol
21 #define D_ALTUSE          d_command1_M,bit7 ; Controls meaning of bits 3 and 4
22 #define D_SPARE6          d_command1_M,bit6 ; spare
23 #define D_SPARE5          d_command1_M,bit5 ; spare
24 #define D_AUTO_SCAN_ALT   d_command1_M,bit4 ; Auto scan with DALTUSE = 1
25 #define D_MANUAL_SCAN     d_command1_M,bit4 ; Manual scan with DALTUSE = 0
26 #define D_CAMERA_ON_ALT   d_command1_M,bit3 ; Camera on with DALTUSE = 1
27 #define D_CAMERA_OFF      d_command1_M,bit3 ; Camera off with DALTUSE = 0
28 #define D_IRIS_CLOSE      d_command1_M,bit2 ; Close Iris
29 #define D_IRIS_OPEN       d_command1_M,bit1 ; Open Iris
30 #define D_FOCUS_NEAR      d_command1_M,bit0 ; Focus near
31
32 #define D_FOCUS_FAR       d_command2_M,bit7 ; Focus far
33 #define D_ZOOM_OUT        d_command2_M,bit6 ; Zoom wide
34 #define D_ZOOM_IN         d_command2_M,bit5 ; Zoom tele
35 #define D_TILT_DOWN       d_command2_M,bit4 ; Down
36 #define D_TILT_UP         d_command2_M,bit3 ; Up
37 #define D_PAN_LEFT        d_command2_M,bit2 ; Left
```

```
38 #define D_PAN_RIGHT       d_command2_M,bit1 ; Right
39 #define D_EXTENDED        d_command2_M,bit0 ; 1 = extended command
40
41 #define D_TILT_MOTION     0x18              ; Tilt motion mask
42 #define D_PAN_MOTION      0x06              ; Pan motion mask
43 #define D_MOTION          0x1E              ; Any motion mask
44
45 #define D_STOP            0x00              ; All zeros stops motion
46
47 #define D_PAN_SPEED       dproto5_M         ; Pan speed
48 #define D_TILT_SPEED      dproto6_M         ; Tilt speed
49
50 #define D_SYNC            0xFF              ; Sync, 1st, byte
51
52 ; Command codes for use with extended commands in D protocol
53 #define D_CLEAR_AUXILIARY          0x0B
54 #define D_CLEAR_PERSET             0x05
55 #define D_CLEAR_SCREEN             0x17
56 #define D_END_PATTERN              0x21
57 #define D_FLIP                     0x21 ; call 33
58 #define D_GOTO_PRESET              0x07
59 #define D_HOME                     0x22 ; call 34
60 #define D_MENU                     0x5F ; set 95
61 #define D_QUERY                    0x45
62 #define D_RESET                    0x0F ; was 0x29
63 #define D_RUN_PATTERN              0x23
64 #define D_SET_AUXILIARY            0x09
65 #define D_SET_PRESET               0x03
66 #define D_START_PATTERN            0x1F
67 #define D_WRITE_CHAR               0x15
68         list
69 ; End of PROTOCOL.INC
70 ;
```

APPENDIX B

B D Protocol

This section covers the "D" Protocol. This protocol is used between matrix switching systems and receiver/drivers.

B.1 Physical Layer

Receivers use serial RS422/RS485 as the basic communications mechanism.

Connection mechanisms vary and are dependent on the equipment. Please refer to the manual for the specific equipment concerned.

B.2 Byte Format

The data format for a character is 1 start bit, 8 data bits, and 1 stop bit, no parity.

B.3 Message Format

Each message consists of 7 bytes as follows:

Byte	Function
1	Synchronization character
2	Receiver address
3	Command byte 1
4	Command byte 2
5	Data byte 1
6	Data byte 2
7	Checksum

B.4 Messages

B.4.1 Byte 1, Sync byte

The synchronization character is always `0xFF` (8 data bits, all ones).

B.4.2 Byte 2, Address byte

The receiver address is any 8 bit binary value from `0x01` through `0xFF`. Note that address zero (`0x00`) is invalid.

[5] `$Header: d:/txb-s422/RCS/dprotoc.inc,v 1.2 2001-11-30 15:12:01-08 Hamilton Exp Hamilton $`

B.4.3 Bytes 3, 4, 5, and 6 as PTZ commands

The commands are divided up into two groups. The first group are commands that control the pan, tilt, and zoom functions (are called PTZ functions).

B.4.3.1 Byte 3, Command 1

Bit	Function
7	Sense
6	Reserved, 0
5	Reserved, 0
4	Auto scan, manual scan
3	Camera on, camera off
2	Iris close
1	Iris Open
0	Focus Near

The sense bit (command 1, bit 7) indicates the meaning of bits 3 and 4. If the sense bit is on, and bits 3 and 4 are on, the command will enable auto-scan and turn the camera on. If the sense bit is off and bits 3 and 4 are on the command will enable manual scan and turn the camera off. Of course, it either bit 3 or 4 are off then no action will be taken for those features. And both bits do not need to be turned on simultaneously.

The reserved bits, 5 and 6, should be set to 0.

B.4.3.2 Byte 4, Command 2

Bit	Function
7	Focus Far
6	Zoom wide
5	Zoom telephoto
4	Tilt down
3	Tilt up
2	Pan left
1	Pan right
0	Always 0 in the PTZ commands

B.4.3.3 Byte 5, Data 1

Contains the pan speed. Legal values are `0x00` (slow) to `0x3F` (fast) and `0x40` turbo. Turbo speed is the maximum speed the unit can run at and is considered separately because the step from high speed to turbo speed is not smooth. That is, going from one speed to the next usually looks smooth and will provide for smooth motion with the exception of going into and out of turbo speed.

B.4.3.4 Byte 6, Data 2

Contains the tilt speed. Legal values are `0x00` (slow) to `0x3F` (fast).

B.4.3.5 Usage summary of bytes 3, 4, 5 and 6

	7	6	5	4	3	2	1	0
Command 1 Alt cmnd 1	Sense	0	0	Auto/ Manual	On/ Off	Close	Open	Near
Command 2	Far	Wide	Telephoto	Down	Up	Left	Right	Always 0
Data 1	Pan speed, 0 → `0x3F`, with `0x40` = Turbo speed							
Data 2	Tilt speed, 0 → `0x3F`							

B.4.4 Bytes 3, 4, 5, and 6 as Extended commands

The extended commands (non-PTZ) always have bit 0 of byte 4 set to 1. The values in the quick reference table, below, are all in hexadecimal.

Command	Byte 3	Byte 4	Byte 5	Byte 6	Spectra?
Set preset	`0x00`	`0x03`	`0x00`	`0x01 to 0x20`	`Y`
Clear preset	`0x00`	`0x05`	`0x00`	`0x01 to 0x20`	`Y`
Go to preset	`0x00`	`0x07`	`0x00`	`0x01 to 0x20`	`Y`
Flip	`0x00`	`0x07`	`0x00`	`0x21`	`Y`
Zero pan position	`0x00`	`0x07`	`0x00`	`0x22`	`Y`
Set preset, part 2	`0x00`	`0x03`	`0x00`	`0x23 to 0x42`	`Y`
Clear preset, part 2	`0x00`	`0x05`	`0x00`	`0x23 to 0x42`	`Y`
Go to preset, part 2	`0x00`	`0x07`	`0x00`	`0x23 to 0x42`	`Y`
Set aux	`0x00`	`0x09`	`0x00`	`0x01-0x08`	`Y`
Clear aux	`0x00`	`0x0B`	`0x00`	`0x01-0x08`	`Y`
Remote reset	`0x00`	`0x0F`	`0x00`	`0x00`	`Y`
Set zone start	`0x00`	`0x11`	`0x00`	`0x01-0x08`	`Y`
Set zone end	`0x00`	`0x13`	`0x00`	`0x01-0x08`	`Y`
Write character to screen	`0x00`	`0x15`	`0x00-0x27`	ASCII char	`Y`

Continued on the next page.

Continued from the previous page.

Command	Byte 3	Byte 4	Byte 5	Byte 6	Spectra?
Clear screen	0x00	0x17	0x00	0x00	Y
Alarm acknowledge	0x00	0x19	0x00	Alarm #, 1 based	Y
Zone scan on	0x00	0x1B	0x00	0x00	Y
Zone scan off	0x00	0x1D	0x00	0x00	Y
Pattern start	0x00	0x1F	0x00	0x00-0x02	Y
Pattern stop	0x00	0x21	0x00	0x00-0x02	Y
Run pattern	0x00	0x23	0x00	0x00-0x02	Y
Zoom lens speed	0x00	0x25	0x00	0x00-0x03 (slow - fast)	Y
Focus lens speed	0x00	0x27	0x00	0x00-0x03 (slow - fast)	Y
Reset camera to factory defaults	0x00	0x29	0x00	0x00	N
Focus auto/off/on	0x00	0x2B	0x00	0x00-0x02	N
Iris auto/off/on	0x00	0x2D	0x00	0x00-0x02	N
AGC auto/off/on	0x00	0x2F	0x00	0x00-0x02	N
Backlight compensation off/on	0x00	0x31	0x00	0x01-0x02	N
White balance auto/off	0x00	0x33	0x00	0x01-0x02	N
Enable device phase delay mode	0x00	0x35	0x00	0x00	N
Set shutter speed	0x00	0x37	any	any	N
Adjust line lock phase delay	0x00-0x01	0x39	any	any	N
Adjust white balance (R-B)	0x00-0x01	0x3B	any	any	N
Adjust white balance (M-G)	0x00-0x01	0x3D	any	any	N
Adjust gain	0x00-0x01	0x3F	any	any	N
Adjust auto iris level	0x00-0x01	0x41	any	any	N
Adjust auto-iris peak level	0x00-0x01	0x43	any	any	N
Query	0x00	0x45	any	any	Y

Continued on the next page.

Continued from the previous page.					
Command	Byte 3	Byte 4	Byte 5	Byte 6	Spectra?

Note that Query may only be used in a point to point application. A device being queried will respond to any address. Therefore, if more than one device hears this command, they will all simultaneously respond and the replies will be messed up.

The response to this style of command consists of four bytes. Byte 1 is the sync byte (`0xFF`), byte 2 is the receiver address, byte 3 is alarm information, and byte 4 is the check sum.

The alarm information consists of the values `0xB0` through `0xB7` representing alarms 1 through 8 respectively.

B.4.5 Byte 7, Checksum

Contains the 8 bit sum of bytes 2 through 6.

B.5 Creating Labels

Many devices have the ability to display labels on the video. Labels that identify the preset or zone being scanned are common. There is a special technique to establish a label that is associated with either a preset or a zone. First, send the label to the receiver/driver using the "Write char. to Screen" command. After the label is up on the screen set the preset/zone. That will establish the label and associate it with the preset/zone.

B.6 Examples

Command to send	Command						
	Sync	Add	Cmnd1	Cmnd2	Data1 Pan	Data2 Tilt	Cksm
Receiver 1, Camera on	`0xFF`	`0x01`	`0x88`	`0x00`	`0x00`	`0x00`	`0x89`
Receiver 1, Camera off	`0xFF`	`0x01`	`0x08`	`0x00`	`0x00`	`0x00`	`0x09`
Receiver 2, Left $\frac{1}{2}$ speed	`0xFF`	`0x02`	`0x00`	`0x04`	`0x00`	`0x20`	`0x26`
Receiver 2, Stop	`0xFF`	`0x02`	`0x00`	`0x00`	`0x00`	`0x00`	`0x02`
Receiver 10, Camera on, Focus Far, Left, Turbo Speed	`0xFF`	`0x0A`	`0x88`	`0x82`	`0x40`	`0x00`	`0x54`

B.6.1 Calculating a checksum

Using the last of the examples, we get:

	0x0A	0000 1010	
	0x88	1000 1000	
1	Subtotal	1001 0010	0x92
	0x82	1000 0020	
2	Subtotal	0001 0100	0x14
	0x40	0400 0000	
3	Subtotal	0101 0100	0x54
	0x00	0000 0000	
4	Total	0101 0100	0x54

Note in step 2 that when adding modulo 256 results in having the high order bit "disappearing".

B.7 Responses

Devices that utilize the D protocol may generate a response.

B.7.1 General Responses

The general response to a received command consist of information about the current alarms present at the remote unit. The format of the response message is as follows:

Sync	Address	Alarm information	Checksum

The alarm information is a bit encoded byte which indicates which bytes are active. The format of the alarm information byte is:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Alarm 8	Alarm 7	Alarm 6	Alarm 5	Alarm 4	Alarm 3	Alarm 2	Alarm 1

The address is the address of the unit responding.

Active alarms are indicated by having the appropriate bit set (`1`). Inactive alarms have their bits reset (`0`).

The checksum is the sum, modulo 256, of the address and alarm information bytes.

B.7.2 Query response

The response to a query command is:

Sync	Address	Part Number (15 bytes)	Checksum

The address is the address of the unit responding.

The part number is the ASCII text string containing the program number (`PRG...`) of the device being queried.

The checksum is the sum, modulo 256, of the original query command's checksum, the address and the 15-byte part number.

B.8 Command Descriptions

Most of the commands are straight forward. The following explains some of the commands.

1. Adjust auto-iris level `0x46`

   If byte 1 is 0, byte 3 and byte 4 are the high and low bytes respectively of an unsigned 16-bit number which the new level. If byte 1 is 1, byte 3 and byte 4 are the high and low bytes respectively of a signed 16-bit number which is the change to the current level. The level is limited internally to the range from 0 to 255 (FF hex). If an attempt is made to set or change the level to a value outside this range, the gain will be set to the appropriate end of the range.

2. Adjust auto-iris peak value `0x43`

   If byte 1 is 0, byte 3 and byte 4 are the high and low bytes respectively of an unsigned 16-bit number which the new peak value. If byte 1 is 1, byte 3 and byte 4 are the high and low bytes respectively of a signed 16- bit number which is the change to the current peak. The peak is limited internally to the range from 0 to 255 (FF hex). If an attempt is made to set or change the peak to a value outside this range, the peak will be set to the appropriate end of the range.

3. Adjust gain `0x3F`

   If byte 1 is 0, byte 3 and byte 4 are the high and low bytes respectively of an unsigned 16-bit number which is the new gain. If byte 1 is 1, byte 3 and byte 4 are the high and low bytes respectively of a signed 16-bit number which is the change to the current gain. The gain is limited internally to the range from 0 to 448 (1C0 hex). If an attempt is made to set or change the gain to a value outside this range, the gain will be set to the appropriate end of the range.

4. Adjust line lock phase delay `0x39`

   If byte 1 is 0, byte 3 and byte 4 are the high and low bytes respectively of an unsigned 16-bit number which is the new phase delay. If byte 1 is 1, byte 3 and byte 4 are the high and low bytes respectively of a signed 16-bit number which is the change to the current phase delay. The phase delay is the delay between the zero crossing of the AC power waveform and the line lock signal sent to the camera. It is in units of 1.085 microseconds. The phase delay is limited internally to the range from 0 to 32767. If an attempt is made to set or change the delay to a value outside this range, the delay will be set to the appropriate end of the range. This command disables device phase delay mode.

5. Adjust white balance (M-G) `0x3D`

   If byte 1 is 0, byte 3 and byte 4 are the high and low bytes respectively of an unsigned 16-bit number which is the new magenta-green white balance value. If byte 1 is 1, byte 3 and byte 4 are the high and low bytes respectively of a signed 16-bit number which is the change to the current magenta-green white balance value. The balance value is limited internally to the range from 192 (C0 hex) to 768 (300 hex). If an attempt is made to set or change the

balance to a value outside this range, the balance will be set to the appropriate end of the range. This command turns off auto white balance.

6. Adjust white balance (R-B) `0x3B`

   If byte 1 is 0, byte 3 and byte 4 are the high and low bytes respectively of an unsigned 16-bit number which is the new red-blue white balance value. If byte 1 is 1, byte 3 and byte 4 are the high and low bytes respectively of a signed 16-bit number which is the change to the current red-blue white balance value. The balance value is limited internally to the range from 192 (C0 hex) to 768 (300 hex). If an attempt is made to set or change the balance to a value outside this range, the balance will be set to the appropriate end of the range. This command turns off auto white balance.

7. AGC auto/on/off `0x2F`

   If byte 4 is 0 the device automatically controls whether AGC (automatic gain control) is on or off (default), If byte 4 is 1, AGC is turned off (manual gain). If byte 4 is 2, AGC is turned on. Other values are ignored. Sending an adjust gain command turns AGC off.

8. Auto focus auto/on/off `0x2B`

   If byte 4 is 0 the device automatically controls whether auto focus is on or off (default). If byte 4 is 1, auto focus is turned off. If byte 4 is 2, auto focus is turned on. Other values are ignored.

9. Auto iris auto/on/off `0x2D`

   If byte 4 is 0 the device automatically controls whether auto iris is on or off (default). If byte 4 is 1, auto iris is turned off. If byte 4 is 2, auto iris is turned on. Other values are ignored.

10. Auto white balance on/off `0x33`

    If byte 4 is 1, auto white balance is turned on (default). If byte 4 is 2, auto white balance is turned off. Other values are ignored. Sending an adjust white balance command turns auto white balance off.

11. Backlight compensation on/off `0x31`

    If byte 4 is 1, backlight compensation is turned off (default). If byte 4 is 2, backlight compensation is turned on. Other values are ignored.

12. Enable device phase delay mode `0x36`

    When device phase delay is set, the phase delay is set by the device (there may be a manual adjustment). Sending an adjust line lock phase delay command will disable device phase delay mode.

13. Reset camera to defaults `0x29`

    Resets the camera to its default condition, except that the current phase delay is not changed.

14. Set shutter speed `0x37`

    Byte 3 and byte 4 are the high and low bytes respectively of 1 divided by the shutter speed. The shutter speed is limited internally to the range from 1/60 second (NTSC) or 1/50 second (PAL) to 1/30000 second, corresponding to a sent number range from 60 (or 50) to 30000. If the sent number is, 0 the shutter speed is reset to its default value (1/60 or 1/50 second). If the sent number is 1, the shutter speed is moved to the next faster speed in the shutter speed table (below). If the sent number is 2, the shutter speed is set to the next slower speed in the table.

    Valid values are in the range 60-30000.

    Sending the value 1 will increment the shutter speed to the next higher value in the shutter speed table. Sending a 2 will decrement to the next lower value in the shutter speed table.

Shutter Speeds	
$\frac{1}{60}$ NTSC	$\frac{1}{1000}$
$\frac{1}{50}$ PAL	$\frac{1}{2000}$
$\frac{1}{100}$	$\frac{1}{4000}$
$\frac{1}{120}$	$\frac{1}{10000}$
$\frac{1}{250}$	$\frac{1}{30000}$
$\frac{1}{500}$	—

APPENDIX C

Index