03-18-02   CTH

Programmer notes for KBD300A Release 4.30

The SPDCONV routine was modified again. The reason for the change this time is that the size of the dead zone was increased for the pan and tilt directions. The SPDCONV routine now receives a byte that ranges from 0 to 110. This byte is then entered into a function that can be expressed as:

y = (134/256 * x) + (1/128 * x * shift) + 7

y is the output byte ranging from 0x07 to 0x40 (outputs above 0x40 are forced to 0x40 in the SPDCONV routine). To test this new routine I ran the MPLAB simulator over and over with different input values. The main thing to check for is whether you can reach turbo speed in the situations where you're supposed to reach turbo speed. This was verified in my testing.

08/15/01   CTH

Programmer notes for KBD300 Release 4.20

The major change in this revision is related to the Speed Conversion routine. This routine now handles speed conversion through calculation. The routine receives a byte that ranges from 0 to 115. This byte is then entered into a function that can be expressed as:

y = (127/256 * x) + (1/256 * x * shift) + 7

y is the output byte. This byte ranges from 0x07 to 0x40 (outputs above 0x40 are forced to 0x40 in the Speed Conversion routine). x is the input byte that ranges from 0 to 115. shift is the amount that the joystick is off center on the pan/tilt axis (shift is normally adjusted before it is used in this equation). This speed mapping is not a perfect one (this is shown in the speed distribution Excel file that is enclosed with the KBD300 files). The shift variable is incremented within the Speed Conversion routine to achieve the correct speed mapping. When you increment the shift variable in the equation shown the Speed Conversion routine will reach turbo sooner and the A to D speed mapping involves less ADC readings.

The Direct Mode transmission was modified such that there is no lag time between Direct Mode bytes that are transmitted. This is done by pre-loading a byte into the UART while the UART is transmitting the stop bit of the previous byte. The reason for this change comes from a customer complaint. The customer's wireless application was not receiving Direct Mode bytes properly because of the lag time between bytes.

02/15/01   CTH

Programmer notes for KBD300 Release 4.10:

Some minor changes that weren’t stated in the comments are as follows. The RAM space of the chip was rearranged to put flags and temporary variables in the RAM space between the following addresses (0x70 to 0x7F). RAM in this address range does not have to be banked to access the proper RAM value. The banking macros were changed because there are now two banking bits associated with bank switching (this change is caused by the chip change from PIC16C642 to PIC16F876). The macros handle situations where you know what bank you’re in and what bank you’re switching to. It is often the case that only one of the two banking bits needs to be set or cleared to do a proper bank switch. For example, the switch from BANK0 to BANK1 or BANK1 to BANK0 involves setting/clearing one of the two banking bits. So the macros, BNK0TO1 and BNK1TO0 will set/clear the appropriate bit. If I used the default bank switching macro, (BANKSEL) this macro always uses two bit setting/clearing instructions for banking. I tried applying this same idea to the paging macros, but the assembler did not like it. The assembler gave me error messages that suggested that this idea is not allowed with paging macros. So the default macro (PAGESEL) is being used instead.

With the update to the new chip, the RAM clearing routine had to be changed also. The previous chip had two banks to clear. This chip has four banks to clear.

Some RAM variables were deleted because it was discovered that they aren’t being used. The variables are: LEDNEW, LEDSAVE, SPEEDHI, SPEEDLO, SPEEDHIT, SPEEDLOT. The speed variables are used by the KBD200 code, but not the KBD300 code. The LED variables weren’t being used by either keyboard.

02/05/00 CTH

Programmer notes for KBD200/300 Release 4.00:

For the KBD200 some changes were made that were unique to this keyboard’s operation in direct mode. One change in particular that was necessary was for the pattern key to override use of the special debounce timer because the retransmit timer may be running when you press/hold the pattern key (to run/set a long pattern. The PATSTAT,BIT0 flag in the code shows when and where this override takes place. The other KBD200 direct mode change was to add NOP instructions following the lines of code that enabled/disabled the UART. It was found that the pan/tilt keys would lock without the NOPs following the UART enable/disable.
I don’t know why this bug occurred. There’s probably some hardware steps that are time critical, and the NOPs give enough time for them to happen.
The KBD300 didn’t exhibit the bug involving the retransmit timer because the KBD300 does not have a retransmit timer. The KBD300 also didn’t exhibit the pan/tilt locking bug of the KBD200.
Functionally both keyboard’s work as described in the source code and the history files.

****

11/17/99 CTH

Programmer notes for KBD200/300 Release 3.20:

This version of released code corrected the telephoto zoom lock problem of the KBD200/300, when the 6700 Matrix Switcher was interfacing the Spectra II in Coaxitron mode. For some yet to be found reason, the telephoto zoom command sent by the KBD200/300 to the 6700 Matrix switcher (in Coaxitron Mode) would lock the Spectra II into zooming long after a stop was sent. The software solution used to fix this problem was to send a second stop command immediately following the first one. This fixed the telephoto zoom lock problem, but the reason why this type of fix is necessary is unknown.

****

7/28/99   AVM/MMT

Programmer notes for KBD200/300 Release 3.10:

This version of released code corrected the interaction between the functionality of the KBD300 joystick and the lens keys (Iris & Focus). It allows the user to simultaneously move the joystick and push a lens key. After the joystick has been returned to the "neutral" state, the lens key continues to function. This version also added two corrections made on the LED Display. One being the corrected problem with the display of the Discrete LEDs (used both for the KBD200 and the KBD4002 keyboards) when installed in the latest PCBoard revision. The other correction was made on the 7 segment LED Display. The character 'P' is displayed when in the "Program" mode and after exiting the "Program" mode, it returns to its current state, displaying the current Monitor number.

****

3/15/99   AVM

Programmer notes for KBD200/300 Release 3.00:

This version of released code has combined '3' modes of operation for the keyboard which can be individually selected via the backpanel 8-position Dip Switch. These modes are: 1)normal CM6700 mode; 2)Direct Mode; 3) ASCII Non-polled mode; See the users manual for a complete description of each mode of operation. This release has also included an embedded Diagnostic Monitor that works with a serial PC Application to allow for quick Pass/Fail testing of keyboard keys, LEDs, and switches.

****

1/22/98   AVM

Programmer notes for KBD200/300 Release 2.00:

This version of released code corrected a Hardware contention problem between the Pic16C642 and the Maxim114 AtoD Controller. This version also added a "Direct Mode" of operation for the keyboard that allows a user to connect directly to a Receiver/Driver (up to 16) via the RS-422 port. This method of control uses the PELCO "P" protocol for data communication. See the KBD300 specification #97-109 or the enclosed KBD300.hst (history file) for details.

****

10/29/97   AVM

Programmer notes for KBD200/300 Release 1.20:

The old KBD200 source code was abandoned and the KBD300 source code was ported to the KBD200. IFDEF statements handle operations that are specific to each model. Blocks of source code are identified as follows:

```
            ifdef  KBD200

            [code segment]
                    |
                    |
            [end of code segment]

            endif ;KBD200
```

To compile the source code for KBD300, for example, one must define the symbol KBD300 in MPLAB under Menu - "Project"; Submenu - "Make Setup." Once here, enter the following: " /dKBD300 " in the field entitled: Extra Assembler Options. The released project files for the KBD200 and KBD300, kbd200mg.pjt and kbd300mg.pjt respectively, already have this Option selected.

Also added and modified in this Firmware release are the New Multiplexer Mode commands. All of these modifications are commented by - " ; MUX." Some of the KBD200 modifications do not affect the KBD300 and are commented with - " ; 200 " and other modifications are Multiplexer and KBD200 specific and are commented as - " ; 200 MUX "

Another modification made in the KBD300 code ONLY is specific to how the JOYSTICK PTZ commands are resolved and translated for transmission to the CM6700 matrix. The default communication protocol between the KBDs and the CM6700 is the Pelco ASCII Protocol. However, to improve PTZ response and accuracy with the KBD300-to-6700 communication a new Cryptic('Sly mode') protocol was adopted. This protocol is used ONLY by the KBD300 when communicating with a CM6700 whose firmware revision is v2.00 or greater. The Cryptic('Sly mode') protocol is composed of five(5) hexadecimal bytes organized in the following pattern:

```
Byte #1 --  Header/Sync Byte ==  $A0

Byte #2 --  PTZ Command Byte ==  $xx  (See Bit encoding below)


Byte #3 --  Pan Speed Byte   ==  $00 to $40   (Current Rx/Drvrs only

                                         use 64-decimal speeds)
Byte #4 --  Tilt Speed Byte  ==  $00 to $40   (Current Rx/Drvrs only

                                         use 64-decimal speeds)

Byte #5     --  ASCII Protocol Terminator == 'a' ($61)


The PTZ Command Byte is Bit encoded as follows:

      Bit7  --> ALWAYS '0'; never used

      Bit6  --> Zoom Wide direction when SET == '1'

      Bit5  --> Zoom Tele direction when SET == '1'

      Bit4  --> Tilt Down direction when SET == '1'

      Bit3  --> Tilt Up direction when SET == '1'

      Bit2  --> Pan Left direction when SET == '1'

      Bit1  --> Pan Right direction when SET == '1'

      Bit0  --> ALWAYS '0'; never used
```

This Cryptic('Sly mode') command format is transmitted ONLY for Joystick PTZ commands as shown in the above Bit Encoding. Also, note that the appropriate PTZ axis Bit can be SET and the corresponding Speed Byte can have a value of '0' which indicates a STOP of that PTZ axis (except for the Zoom Axis which does not transmit a Speed). Also, note that when a PTZ Axis is NOT moving the corresponding bit in the Command Byte is '0', regardless of the Speed value. The KBD300 code will sample the Joystick and SET the corresponding bit for the PTZ Axis/axes moving and in turn the code will Clear the corresponding bit (in the Command Byte) for a PTZ axis that has stopped moving or direction has changed (i.e. joystick is moving Right, then is changed to Left; the Command Byte bit #2 is Set then the change is made and the Command Byte bit #3 is set while bit #2 is cleared).

Two scenarios exist where the Cryptic('Sly mode') is not transmitted for PTZ movement from the Joystick. The first case is for an ALL STOP condition which indicates that all 3 axes of the Joystick are centered or in the "Dead Zone." In this case the normal ASCII All stop is transmitted: 'sa'.
What is also transmitted as a Special Case is the Zoom Wide code. This case is special because the KBD300 must allow for alternate speed settings on the Zoom Axis. This multiple speed adjustment on the Zoom axis is handled by entering a digit between 1-4 followed by the Zoom Wide command. To allow the New Zoom Speed to be accepted by the Matrix, the KBD300 transmits the Zoom

Wide command in the normal ASCII mode: 'Wa', followed by the Cryptic command.

Transmitting the Zoom Wide command in this fashion ensures that the Speed change for Zoom is Set and then activated immediately so that the KBD300 operator does not need to twist the joystick, return it to neutral and then twist the joystick one more time to move the lens at the newly set speed.