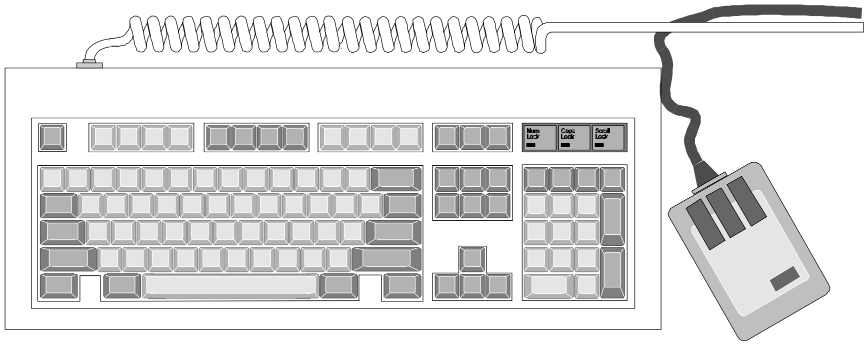


MultiKey/42i Developer's Technical Reference

Keyboard Controller Firmware
for a Super I/O Chipset
With An Embedded 8042 Microcontroller

Copyright

Disclaimers

Trademarks

Document Inquiries

No document number is displayed on the cover of this document. When discussing this document please refer to the title (MultiKey/42i Developer's Technical Reference), publication date (August 23, 1996) and the following internal tracking number: 1.01.

Preface

This document presents the specifications and functional data for the Phoenix MultiKey/42i product. MultiKey/42i is a keyboard and auxiliary device software for personal computers designed and licensed for use in the 8042 microcontroller family. This document is organized into the following chapters.

Chapter 1 MultiKey/42i Overview **-** An introduction to the MultiKey/42i features and functionality.

Chapter 2 MultiKey/42i Hardware Perspectives **-** A discussion of the MultiKey interface from a hardware point of view.

Chapter 3 MultiKey/42i Software Interface **-** A complete listing of the MultiKey/42i command set.

Chapter 4 MultiKey/42i Configuration Utility **-** A complete description of the MultiKey/42i Configuration Utility.

Chapter 5 MultiKey Keyboard Controller Routines **-** Coding examples for communicating with the Keyboard Controller.

Related Documentation

Consult the following documents for additional information.

- System BIOS for IBM PCs, Compatibles, and EISA Computers, Second Edition, Phoenix Technologies Ltd., Addison Wesley, 1991

This page left blank.

Contents

Figures

This page left blank.

Tables

This page left blank.

Chapter 1
MultiKey/42i Overview

The following describes the design considerations of the MultiKey/42i Keyboard Controller code. The design of the MultiKey/42i code follows the other MultiKey products, and is most closely related to the MultiKey/42E (Enhanced 2K 8042 Keyboard Controller). The 'i' in the name MultiKey/42i stands for 'improved'; (an improved 'green' keyboard controller). MultiKey/42i is also a 2KByte 8042 product with extended security and power conservation features added.

1.1 Standard/Extended Features

The MultiKey/42i code has been designed to fit in a standard 2K 8042, however the feature set has been setup to best fit with a Super I/O chipset containing an 8042 processor. The following is a list of the MultiKey/42i standard and extended features:

- Standard 2K 8042 Keyboard Controller Code Size
- Standard AT, PS/2, AX, OADG, Microsoft Natural Keyboard Support
- Standard and Extended PS/2 Mouse Support
- Transparent Software GateA20 Support
- Keyboard and Mouse Port Swapping Support
- RAM/ROM Scan Code Conversion Table
- Password and KeyLock Security Support
- Dual Password (User & Supervisor) Support
- Enable/Disable Security Pin Control task
- Secure USB Password Validation Support
- Secure Password (cannot be read or overwritten)
- Secure Configuration (cannot be changed once locked)
- Programmable HotKey and Task Support
- QuickLock with Rotating LED Support
- Inactivity Timer for Powering Down External Devices
- Inactivity Invoked Security Support
- Inactivity Indication (Flashing Scroll Lock LED)
- External Input Detection & Task Support
- External Input Enable/Disable Security Support
- Power Restored based on Mouse, Keyboard, and External Input
- OEM MultiKey/42i Configuration Utility

1.1.1 Standard 2K 8042 Code Size

The 2KByte code size will allow MultiKey/42i code to be programmed into any standard 8042 or Super I/O chipset that contains a 8042 processor. No hardware GateA20 is required for the 8042 processor because MultiKey/42i supports transparent software GateA20 switching. The 8042's schematic follows the PS/2 style architecture whether the design requires a PS/2 Mouse, or not.

1.1.2 Standard/Extended Keyboard Support

MultiKey/42i is compatible with all types of keyboards including any AT, PS/2, 84-key, 101/102-key, 105-key, 106-key, AX, OADG, or a Microsoft Natural Keyboard.

1.1.3 Standard/Extended PS/2 Mouse Support

MultiKey/42i is compatible with all types of PS/2 pointing devices including Trackballs, Touchpads, and Mice.

1.1.4 Transparent Software GateA20

GateA20 commands are used to access system memory above 1MB. They are frequently used in Windows and Novell networking applications. GateA20 commands have a higher priority than Keyboard input operations. As machine speed and software memory requirements have increased, the number of GateA20 commands have increased and the number of Keyboard and Mouse input operations have decreased.

When MultiKey/42i is used, the Keyboard and Mouse data paths are not disabled during GateA20 operations; this ensures that no Keyboard or Mouse data is missed and increases the speed of each GateA20 command.

1.1.5 Keyboard/Mouse Port Swapping

In MultiKey/42i the System BIOS can select which external device is the Keyboard and which is the Mouse. Traditionally, Port0 is used by a Keyboard and Port1 is used by a Mouse, however, Port Swapping permits these ports to be interchanged providing significant savings in Motherboard real estate. This capability offers the benefit of Ease of Configurability with Two Identical Ports: both PS/2 Mouse and Keyboard use 6-pin Mini-DIN connectors.

NOTE: If the System only has one device connector, it should be connected to Port0 and Port1 should be tied high.

1.1.6 RAM/ROM Scan Code Conversion Table

MultiKey/42i allows the System to support multiple Keyboard languages and Keyboard layouts without using an added Device Driver loaded with the Operating System. Languages can be selected, at BIOS Initialization (Boot Time), by the SETUP program.

In addition, individual keys can be remapped, for example: The position of the <Ctrl> can be swapped with <Caps Lock>, mimicking the original 84-key Keyboard; this function is transparent to the operating system as OS/2, DOS, and Windows are.

The Scan Code Table is downloaded by the 0B8h and 0BBh Commands. KCSTATE.3 determines which Scan Code Conversion table (RAM/ROM) is used. KCSTATE.3 is accessed through the MultiKey variable interface (0B8h, 0BCh, & 0BDh Commands). This feature is available only on 8042's with 256 bytes of RAM (i.e. 8742AH, 8042AH, etc.).

1.1.7 Password and KeyLock Security

The Keyboard Password feature is provided by the Keyboard Controller. This Password is in addition to the System BIOS Password support provided by some Systems. The Keyboard Controller Password is from one to sixteen characters and it is stored in the controller, making it a more secure machine than a machine with only the System BIOS Password support.

Along with the Password support, MultiKey/42i offers KeyLock supported on Port1 bit 7 (P1.7), compatible with the original AT 8042 support. Both the KeyLock and the Password have to be inactive before the user can use the Keyboard or Mouse. If KeyLock is not used, P1.7 must be high for normal operation.

1.1.8 Dual Password Support

MultiKey/42i extends the Password size from 8 to 16 characters and it has added a second Password which could be downloaded to the Keyboard Controller. This feature allows the Keyboard Controller to support separate user and supervisor passwords. When both passwords are downloaded, a match on either password disables security. The System can interrogate the Keyboard Controller to determine which password was entered and choose to limit machine access, if desired. If both passwords are identical, the first password match gets the credit for disabling security.

1.1.9 Security Pin Control Task

The Password Security feature can affect external hardware. When Security is enabled, the Pin Control Tasks (LCK1TSK & LCK2TSK) of the loaded passwords are executed. When either Password is entered, the corresponding Pin Control Task is restored. For example, this feature can be used to lower P1.3, which would prevent a cold reset from occurring until the Password was typed in, which would raise P1.3 again.

Since there are two Passwords and two different Pin Control Tasks (LCK1TSK & LCK2TSK), the machine's access can be limited by hardware pins on the 8042. In addition, since each Pin Control Task can affect multiple pins simultaneously, one task can be configured as a subset of the other Pin Control Task. The Pin Control Task can also notify the System by pulsing a Port Pin and thereby causing an SMI.

1.1.10 Secure USB Password Validation

Through a combination of hardware and the Secure Controller Configuration, MultiKey/42i can be configured to validate a password from USB emulation legacy support without producing a hole in which the System or applications would have access to the password or to the password validation path. The password is checked against the USB legacy scan codes when this feature is enabled (KCMISC.2 = 0) and Port1 bit2 (P1.2) is high. Therefore to complete this feature, P1.2 should be connected to a line that is active high during SMM mode.

1.1.11 Secure Password

The password can never be overwritten. Once the password is loaded, the Keyboard Controller must be hard reset before the password can be reloaded. The password storage area cannot be read or written in the Keyboard Controller memory. The MultiKey/42i Extended Commands which are used to read and write the Keyboard Controller memory are blocked in the password storage area.

1.1.12 Secure Controller Configuration

To prevent the Keyboard Controller's configuration from being changed, and there by compromising the Security, the System cannot change the configuration once either Password has been downloaded. Once the Password is loaded, the Keyboard Controller must be hard reset before the Keyboard Controller's configuration can be changed. The System only allows read access to the Keyboard Controller's memory once either password is loaded.

The write Controller memory Command, 0BBh, is blocked providing a Controller Configuration Lock. The MultiKey variable interface is not blocked allowing non-Security related variables to be modified (example: Port Swapping, RAM/ROM conversion table, Inactivity Timer value, LED flags, and Controller Speed variables).

1.1.13 Programmable HotKeys

MultiKey/42i supports up to five HotKey combinations. The HotKeys' Scan Codes can be defined by the System BIOS. The default activate keys are the Ctrl and Alt keys, however these Scan Codes can also be redefined by the System BIOS. The System BIOS uses the MultiKey memory Commands (0B8h, 0BAh, and 0BBh) to update the HotKey related variables. Once the Ctrl+Alt+HotKey Scan Code is detected, the corresponding Pin Control Task is performed (KEY1TSK - KEY5TSK).

HotKey 5 Pin Control Task (KEY5TSK) can be shared with the Inactivity Timer based on TMRFLGS.1. This was done to conserve RAM and to allow the Inactivity Timer to control two different tasks (for example: Lower P1.3 and activate QuickLock). A very efficient use of the configuration resources in the case where Inactivity task is a compound event (above example) and a HotKey QuickLock is desired, by setting the KEY5TSK to equal QuickLock.

This Pin Control Task allows any single or group of Port1 pins to be set, cleared, or pulsed when the a HotKey sequence is detected. In addition to manipulating the Port 1 pin(s), the Pin Control Task can invoke Security, force Inactivity, or toggle between RAM/ROM conversion table. Using the pulsing feature, a HotKey can cause an SMI or other interrupt rather than IRQ1 or IRQ12. During invocation, each HotKey sets an internal variable (FUNCTION) to indicate which HotKey was active. This allows the SMI Handler to determine which HotKey caused the SMI by reading through the MultiKey Variable interface to the FUNCTION variable. The SMI Handler must clear the FUNCTION variable after it has read the value, so shared SMIs can be distinguished.

NOTE: All HotKey sequences work when the keys are released. This prevents the System's shift state flags (example: Ctrl and Alt state flags) from being left in the wrong state. QuickLock is an example where no other keys are sent to the System after the HotKey is detected.

1.1.14 QuickLock Indication

The Keyboard Controller Password, together with the HotKeys (QuickLock), permits the user to disable the Keyboard and PS/2 Mouse without exiting an application. QuickLock is a HotKey configured with a unique scan code and its HotKey Task is set to Enable Security. The Keyboard LEDs rotate to indicate that QuickLock is enabled; the user must enter a Password before continuing. Once the QuickLock HotKey sequence is released, the Keyboard and Mouse remain locked until the Password is entered.

NOTE: Both the Keyboard Controller Password and the QuickLock options require the downloading of at least one Password to the Keyboard Controller. Depending on the specific implementation, this may be done from SETUP or the Phoenix Password Utility.

1.1.15 User Input Inactivity Timer

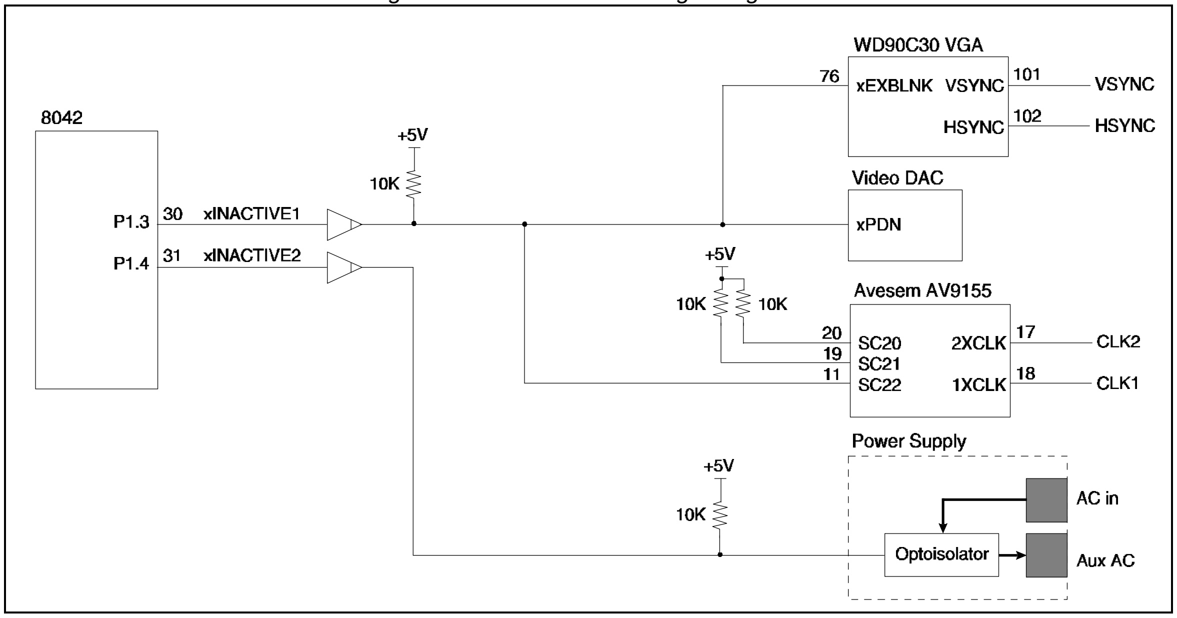
The Inactivity Timer monitors the amount of time that has elapsed since the last Keyboard, Mouse, or External Input event, with an expiration interval selected by the OEM or end-user (options include Off or a time from 30 seconds to 128 minutes). The expiration interval is set by the 0AFh Command Set Inactivity Timer, and the value is stored in the variable TMRATE5 (two’s complement value).

One or two Pin Control Tasks (TMR1TSK and KEY5TSK based on TMRFLGS.1) are performed after the Inactivity Timer expires. By setting the KEY5TSK to QuickLock , the Inactivity Timer shares HotKey 5’s Pin Control Task; this conserves RAM and allows the Inactivity Timer to control two different tasks (for example, Lower P1.3 and activate QuickLock). This allows any single or group of Port1 pins to be set, cleared, or pulsed when the Inactivity Timer expires. In addition to manipulating the Port 1 pin(s), the Pin Control Task can invoke Security or toggle between RAM/ROM conversion table. Using the pulsing feature, the Inactivity Timer can cause an SMI or other interrupt; it is not limited to IRQ1 or IRQ12. During invocation, each Inactivity Tasks sets an internal variable (FUNCTION) to indicate that this is an Inactivity event SMI. This allows the SMI Handler to determine which event caused the SMI by reading through the MultiKey Variable interface to the FUNCTION variable. After it has read the value, the SMI Handler must clear the FUNCTION variable so shared SMIs can be distinguished.

When in Standby mode (i.e. the Inactivity Timer has expired), an incoming keystroke, mouse data, or external input event causes the opposite (except if pulsed) Timer Task function to be performed. If the Timer Task lowered a pin, the new activity raises the pin again. The new activity also re-triggers the Inactivity Timer to the original TMRATE5 value.

Figure 1-1 shows one implementation example controlling Video, Monitor Power, and CPU Clock Speed. The BIOS or a SETUP option would configure the Inactivity Task to lower P1.3 only or to lower both P1.3 & P1.4 based on whether or not a VESA compatible monitor was attached. The Inactivity pin P1.3, xINACTIVE1, speed switches the CPU and powers down the Video DAC which forces the VESA Video Monitor into Standby. The Inactivity pin P1.4, xINACTIVE2, will completely shut off the Power to the monitor, whether it is a VESA Video monitor or not.

Figure 1-1. Inactive Pin Usage Diagram.



Assuming the Inactivity Timer value is not zero (disabled), the corresponding MultiKey/42i configuration for the hardware shown in Figure 1-1, is as follows: TMR1TSK should be set to lower P1.3 or P1.3-2 (lower pin: 00000000b with pin data 00001000b or 00001100b). Optionally, the KEY5TSK could be combined with TMR1TSK to make a compound Inactivity Task (TMRFLGS.1 = 0), and KEY5TSK could be setup to invoke QuickLock (set Security: 11010000b with no pin data 00000000b).

NOTE: When Activity is restored MultiKey/42i will perform the opposite function task. To prevent Security from being cleared, bit7 of Pin Control Task (Pulse Pin flag) is set.

1.1.16 Inactivity Invoked Security

The Inactivity Timer can be configured to set QuickLock when it expires. This allows the user to walk away from the System, and it will power down and secure itself (see Password and QuickLock support), requiring the user to type in a Password before continuing.

1.1.17 Inactivity Indication

MultiKey/42i flashes the Scroll Lock LED (according to TMRFLGS.0) to indicate that the System is powered down due to a Green-PC event. Since the monitor may be off, this flashing may be the user notification that the machine is in STANBY mode rather than completely powered off. A keystroke will restore full power mode.

The Inactivity Indicator has a higher priority than the QuickLock Indicator. If QuickLock is enabled and the LEDs are rotating, and then the Inactivity Timer expires, the Scroll Lock LED will flash. After Keyboard or Mouse activity, the Scroll Lock LED will stop flashing and all the LEDs will rotate until a password is typed.

If the Inactivity Indicator is turned Off (TMRFLGS.0 = 0), and QuickLock is enabled, rotating LEDs indicate that QuickLock is enabled.

1.1.18 External Input Detection

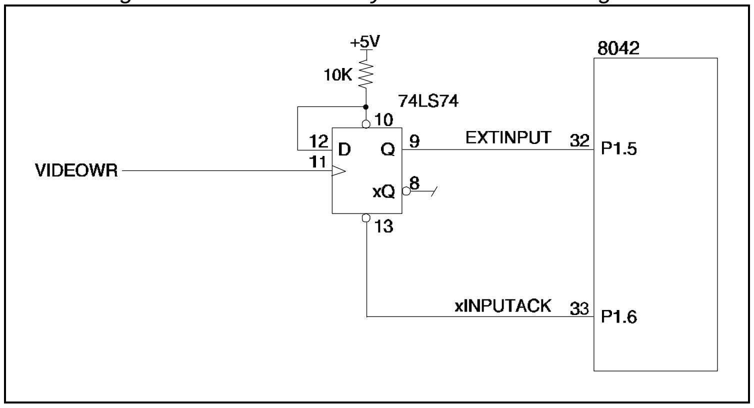
MultiKey/42i monitors two sets of Port1 pins for activity. The Keyboard Controller responds to the event if its pulse length is at least 30µs. After the event is detected, the pin is debounced for 120ms to prevent the performance of multiple Pin Control Tasks.

MultiKey/42i has two Input Pin masks TST1PIN and TST2PIN, configured by using the Extended Memory Commands 0B8h and 0BBh. The corresponding Pin Control Task (PIN1TSK or PIN2TSK) is performed when a pin in the Input Pin Mask wiggles. PIN1TSK and PIN2TSK variables are also configured using the Extended Memory Commands 0B8h and 0BBh. If both masks include the toggling pin, then both Pin Control Tasks are performed.

This Pin Control Task allows any single or group of Port1 pins to be set, cleared, or pulsed when the a External Input is detected. In addition to manipulating the Port 1 pin(s), the Pin Control Task can set or clear Security, set or clear Standby mode, or toggle between RAM/ROM conversion table. Using the pulsing feature, an External Input can cause an SMI or other interrupt rather than IRQ1 or IRQ12. In invocation, each External Input pin mask sets an internal variable (FUNCTION) to indicate which task went active. This allows the SMI Handler to determine which external event caused the SMI by reading through the MultiKey Variable interface the FUNCTION variable. The SMI Handler must clear the FUNCTION variable after it has read the value, so shared SMI's can be distinguished.

MultiKey/42i can guarantee detecting a signal pulse greater than 30us without any external hardware, however with an external flip/flop nanosecond pulses could be detected. An example of a configuration which would prevent the System from going into Standby mode if the Video Screen was being updated is shown below in Figure 1-2. The flip/flop latches the input so the MultiKey/42i code can detect the input change and clears the flip/flop only if any event was detected. With constant activity the Keyboard Controller would clear the flip/flop every 120ms, the debounce rate.

Figure 1-2. Video Memory Write Detection Diagram.

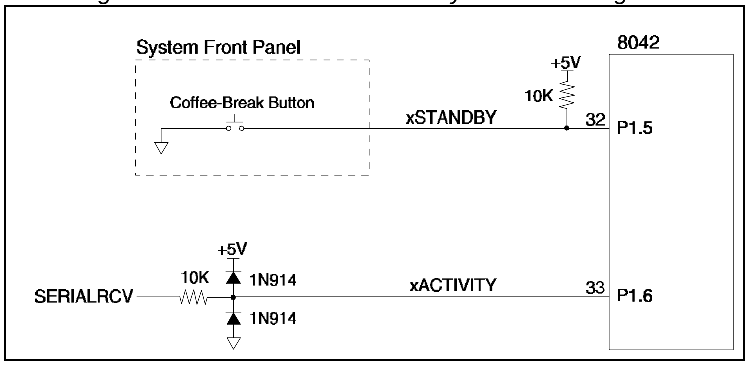


The corresponding MultiKey/42i configuration for the hardware shown in Figure 1-2, is as follows: Both TST1PIN & TST2PIN Input Pin masks should be set to P1.5 (00100000b), PIN1TSK should be set to set Activity (clear Standby: 00110000b with no pin data 00000000b), and PIN2TSK should be set to pulse low P1.6 (10000000b with pin data 01000000b) to reset the flip/flop.

NOTE: The System BIOS should reset the flip/flop manually (with 0C7h & 0C8h Commands) when configuring the rest of the MultiKey/42i features, since the power-up state of the flip/flop is indeterminate.

An alternate configuration for the External Inputs is shown below in Figure 1-3. In this case the MultiKey/42i code is setup to watch the RS-232 Serial Port (Serial Mouse, Modem, etc.) to prevent the System from entering Standby mode and to watch the Coffee-Break[1] button to force Standby mode. The diodes on the SERIALRCV line translate the plus and minus voltage supply of the RS-232 to a 0 to 5 volts level for the 8042.

Figure 1-3. Front Panel with Activity Detection Diagram.



The corresponding MultiKey/42i configuration for the hardware shown in Figure 1-3, is as follows: TST1PIN Input Pin mask should be set to P1.5 (00100000b), TST2PIN Input Pin mask should be set to P1.6 (01000000b), PIN1TSK should be set to set Inactivity (set Standby: 01110000b with no pin data 00000000b), and PIN2TSK should be set to set Activity (clear Standby: 00110000b with no pin data 00000000b. The setting Standby mode would

[1] The Coffee-Break button puts the system in a low power mode. Password protection is available for this option. The Coffee-Break button is OEM configurable. It can be a dedicated front panel key or a HotKey option.

exactly match the Inactivity Timer expiring and both of the Inactivity Timer's Pin Control Tasks would be performed.

1.1.19 External Input Invoked Security

As shown in Section 1.1.18, the External Input event can force the Keyboard Controller into a Standby mode, which could have as an associated task to invoke QuickLock. However, referring to Figure 1-3, the PIN1TSK could be setup to invoke QuickLock (set Security: 01010000b with no pin data 00000000b) directly without causing the Keyboard Controller to go into Standby mode. This allows the user to push the Front-Panel button and walk away from the System, and it will be secure (see Password and QuickLock support), requiring the user to type in a Password before continuing. This configuration directly mimics the original KeyLock on AT machines.

1.1.20 Activity Restored by Mouse, Keyboard and External Input

MultiKey/42i allows the user to resume from Standby mode in a variety of natural ways. System operation resumes in response to the movement of the PS/2 Mouse, a Keystroke, or an Inactivity Timer Command. The System can be configured to resume from external events such as: Serial Mouse movement, Modem Ring, Parallel Port usage, and Video Memory updates.

Upon new activity detection the Inactivity Timer is re-triggered and if it had already expired the Pin Task would be restored. If a the Timer Task Lowered a pin, the new activity will raise the pin again. If the Timer Task was configured to pulse low an external SMI pin when entering Standby mode, then MultiKey/42i will pulse low the same external SMI pin when activity is again detected. The Activity SMI FUNCTION value will be one greater than the Inactivity SMI FUNCTION value.

1.1.21 OEM MultiKey/42i Configuration Utility

Phoenix provides a MultiKey/42i Configuration Utility which allows the OEM to completely setup each unique platform and save the configuration to disk. After running the Configuration Utility, every feature of the platform is 100% testable. The Configuration Utility even calibrates the 8042 clock and adjusts the MultiKey/42i compensation factor so all timings will be accurate.

The output of the Configuration Utility is an ASCII file which can be directly used by the System BIOS to configure the MultiKey/42i every time the System is rebooted.

1.2 Architectural Considerations

Some features, traditionally supported by the MultiKey/42 family have been removed to meet the timing and code space requirements of the MultiKey/42i. Issues presented by these changes are minor when the target motherboard is a new design. These support issues include:

- Keyboard Controller must run at 12MHz
- PS/2 style Keyboard Controller support only
- Some IBM reserved RAM locations are used
- Only the Keyboard LED state is saved
- Extended Keyboard and Mouse Echo Commands

1.2.1 12MHz KBC Platform Support

To guarantee support for all Keyboards the greatest interval through the Main Loop is 27.5µs. Since the Keyboard and PS/2 Mouse devices are polled, the Keyboard Controller must be able to detect the device Startbit which can be as small as 30µs.

1.2.2 PS/2 Style Platform Support

An 8042 running MultiKey/42i code is not functionally equivalent to the same controller under other MultiKey/42 products. To meet code space restrictions the Environment Autodetection (AT or PS/2), Keyboard Controller Interface State Switching (AT or PS/2), and the AT Style device transmission code were removed.

Keyboard Controller Interface State Switching allows a PS/2 Style motherboard without a PS/2 Mouse to be configured as a AT Style Keyboard Controller that cannot respond to Auxiliary Device Commands.

MultiKey/42i allows you to build a PS/2 Mouse-less System with the PS/2 Mouse clock and data lines pulled up. The System recognizes a PS/2 Style 8042 but will not detect a PS/2 Mouse.

1.2.3 IBM Defined RAM Locations

This seems to be a minor issue, but it is documented here for completeness. The IBM RAM locations are defined from 20h to 3Fh, and are accessed by Commands 20h-3Fh (read RAM) and 60h-7Fh (write RAM). These commands were undocumented commands in the AT Style 8042 and then formally documented in the PS/2 Style 8042. Only one RAM location was documented: location 20h, the Keyboard Controller Command Byte (KCCB). The Phoenix clean room documented a few additional locations but most were still reserved. IBM released a second 8042 with only seven of the original commands (KCCB; Password NULL1, NULL2, SCAN1, & SCAN2; and two addition locations 3Dh & 3Fh for temporary storage). The portable designs have also limited the number of valid commands to save memory needed to support this feature.

MultiKey/42i uses a total of eight IBM Reserved RAM locations for memory and Main Loop considerations.

1.2.4 Keyboard Controller State Saving

To save code space MultiKey/42i only shadows the Keyboard LED state, it does not shadow the Typematic Rate, the Scan Code Set, and Keyboard Enable/Disable flag. The Keyboard Controller sets the following default settings: Typematic Rate = 10.9 characters/sec; Scan Code Set = 2; Keyboard is Enabled. Since these features are rarely changed and most desktop Systems do not cut power to their Keyboards, these changes are barely noticable.

1.2.5 Extended Keyboard and Mouse Echo Commands

The MultiKey/42i Keyboard and PS/2 Mouse Echo Commands (0D2h & 0D3h respectively) have been extended beyond the original IBM design to more closely follow the regular Keyboard and PS/2 Mouse data path. These commands were extended for USB legacy Keyboard Controller and device emulation. The Keyboard Scan Code, sent to the Keyboard Controller via the 0D2h Command, is checked against the HotKey list and the Inactivity Timer is restored. Special caution was used, along with hardware considerations, with this command and the password, to prevent a program from determine the password by sending an exhaustive set of Scan Codes until security was disabled. The Secure Password Validation support provides USB HID class device security for USB legacy support identical to the PS/2 Keyboard/Mouse security support.

The HotKey5 Pin Control Task has been linked with the Inactivity Timer Pin Control Task to increase functionality without using additional memory.

1.3 Product Differentiation

The number of features vary greatly between each of the MultiKey products designed to run on the Intel 8042, 8042AH, and 80C42 family of products. One of the 80C42 products, MultiKey/42L, is not a desktop solution and will be left out of the comparison. Table 1-1 lists features supported by the MultiKey 8042 line of desktop products.

Table 1-1. MultiKey 8042 Product Comparison.

MultiKey Feature	/42	/C42	/42G	/42E	/42i
Standard AT, PS/2, AX, OADG, Microsoft Natural Keyboard support	Yes	Yes	Yes	Yes	Yes
Standard and extended PS/2 Mouse support	Yes	Yes	Yes	Yes	Yes
Keyboard Controller code fits in 2KBytes 8042	Yes	-	-	Yes	Yes
Keyboard Controller code capable of running from 6MHz - 12MHz	Yes	Yes	Yes	Yes	-
Autodetection of legacy AT vs. PS/2 Platform	Yes	Yes	Yes	-	-
AT vs PS/2 Keyboard Controller Command mode	Yes	Yes	Yes	Yes	-
Transparent Software GateA20 support	Yes	-	-	Yes	Yes
Hardware GateA20 support	-	Yes	Yes	-	-
Keyboard and Mouse Port-Swapping support	-	Yes	Yes	Yes	Yes
RAM/ROM Scan Code Conversion Table	-	Yes	Yes	Yes	Yes
Password and Keylock Security support	Yes	Yes	Yes	Yes	Yes
Dual Password (User & Supervisor) support	-	-	-	Yes	Yes
Secure Password (cannot be read or overwritten)	-	-	-	-	Yes
Secure USB Password Validation support	-	-	-	-	Yes
Programmable HotKey and Task support	-	-	Yes	Yes	Yes
Quicklock with rotating LED support	-	-	Yes	Yes	Yes
Watchdog or Delayed Event Timer	-	-	Yes	-	-
Inactivity Timer for Powering Down external Devices	-	-	Yes	-	Yes
Five separate Inactivity Timers	-	-	Yes	-	-
Inactivity Invoked Security support	-	-	Yes	-	Yes
Inactivity Indication (Flashing Scroll Lock LED)	-	-	Yes	-	Yes
Temporary Kbd/Aux Lockout Timer (Eat-a-Key Timer)	-	-	Yes	-	-
Lockout Indication (Flashing all LEDs)	-	-	Yes	-	-
External Input Detection & Task support	-	-	Yes	-	Yes
External Input Enable/Disable Security support	-	-	-	-	Yes
Edge and Level External Input Detection	-	-	Yes	-	-
Suspend Power Down for complete Power Management	-	-	Yes	-	-
Power Restored based on Mouse, Keyboard, and External Input	-	-	Yes	-	Yes
Secure Configuration (cannot be changed once locked)	-	-	-	-	Yes
Port 1 “Input Port” Emulation	-	-	Yes	-	-
BIOS Configurable Interrupt Control	-	Yes	Yes	-	-
OEM MultiKey Configuration Utility	-	-	Yes	Yes	Yes
Enable/Disable Security Pin Control Task	-	-	Yes	Yes	Yes

This page left blank.

Chapter 2
MultiKey/42i Hardware Perspectives

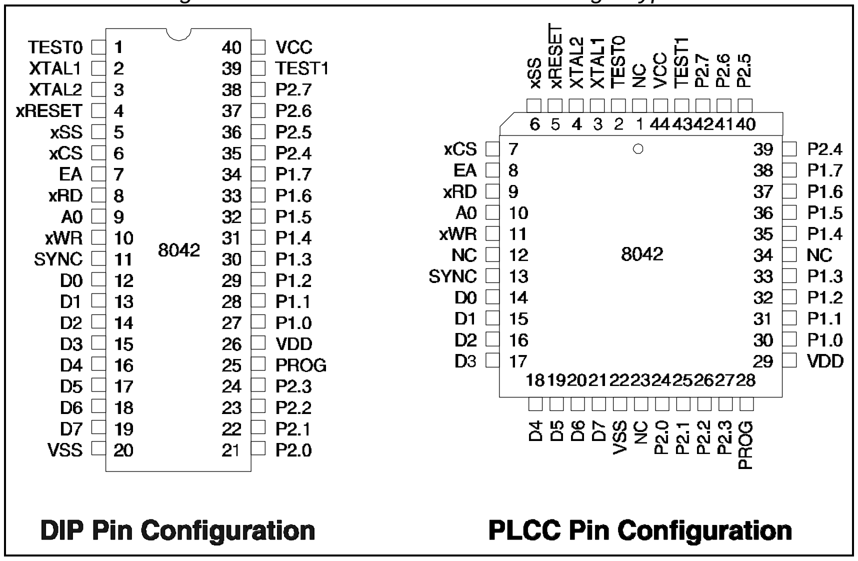
This chapter discusses the MultiKey/42i interface with the Intel 8042. The five main topics include: microprocessor features; schematics of AT and PS/2 platforms, with and without mouse interrupt hardware; pin control tasks; the Standard Memory Map; and the default Scan Code Conversion Table.

2.1 Keyboard Controller Microprocessor

The Intel 8042, 8742, and 8742AH Keyboard Controllers are members of Intel's Universal Peripheral Interface family of Microcontrollers and feature the following:

- Pin, Software and Architectural Compatibility with all Intel UPI-41 and UPI-42 Products
- 8-bit CPU
- Up to 12 MHz Operation
- 8-bit Data Bus Interface Registers
- Interval Timer/Event Counter
- Two 8-bit TTL Compatible I/O Ports
- Resident Clock Oscillator Circuits
- DMA, Interrupt, or Polled Operation Supported
- Expandable I/O
- Interchangeable ROM and EPROM Versions
- 2048 x 8 ROM Size, 256 x 8 RAM Size
- Available in 40-Lead Plastic (DIP) & 44-Lead Plastic Leaded Chip Carrier (PLCC) Packages (see Figure 2-1)

Figure 2-1. 8042 In DIP & PLCC Package Types.



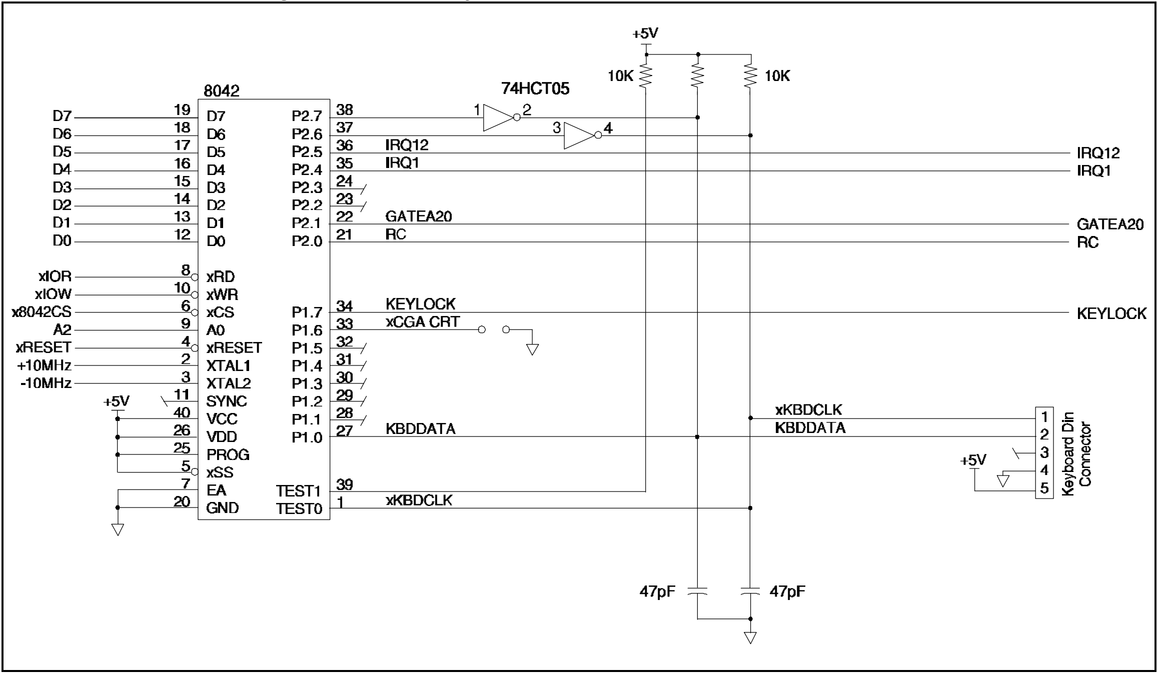
2.2 Schematics

Figures 1-2, 2-3, and 2-4 show the recommended schematics for the AT platform with no PS/2 Mouse, AT platform with PS/2 Mouse, and PS/2 platform with PS/2 Mouse respectively. The AT platform has “edge sensitive” interrupts and the PS/2 platform has “level sensitive” interrupts. Some of the considerations in the MultiKey/42i schematic design are:

- One circuit design for AT Platform without a PS/2 Mouse
- One circuit design for AT Platform with a PS/2 Mouse
- One circuit design for PS/2 Platform with a PS/2 Mouse
- Compatible Keyboard and PS/2 Mouse interface
- Original and Mini-DIN Device Connectors
- Support for KeyLock and CRT jumper
- Support for Software GateA20 on P2.1
- Support for Software CPU Reset on P2.0
- Interrupt support IRQ1 and IRQ12 pins
- Port 1 Pins for System Control features

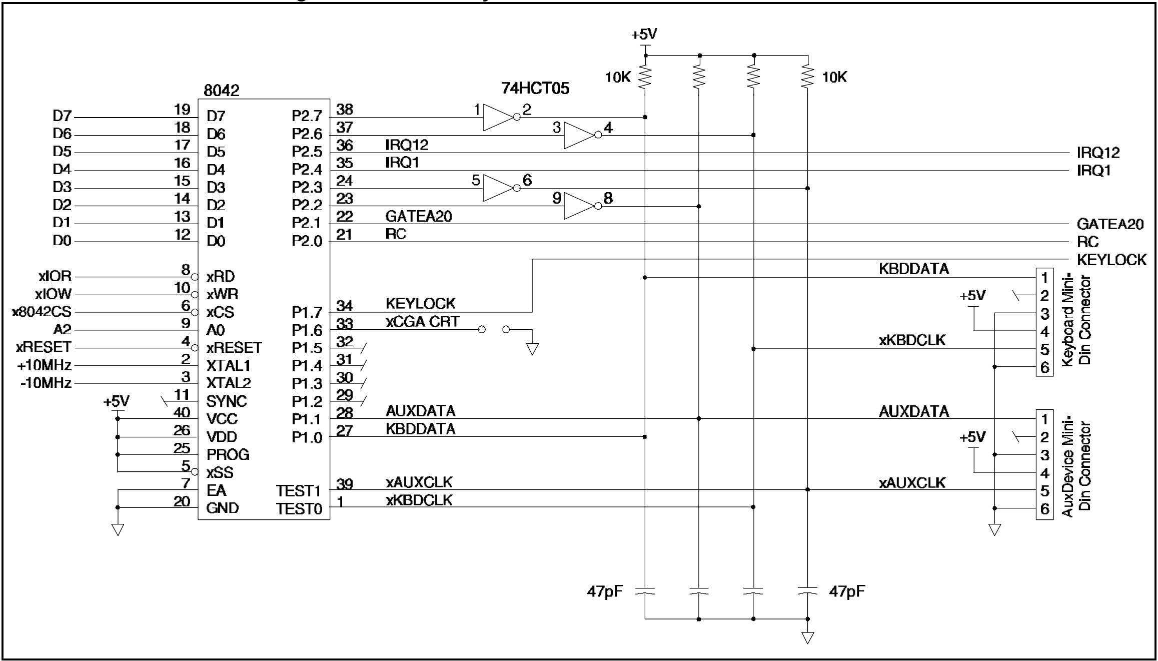
The schematic shown in Figure 1-2 closely reassembles the original PS/2 Platform design with the PS/2 Mouse lines deleted. Unlike MultiKey/42 however, MultiKey/42i does not support the original AT Platform design, however it provides the same support as illustrated in Figure 2-2. The Keyboard connector is shown as the original large DIN connector, however, the Mini-DIN could easily be substituted. The TEST1 pin must be pulled up since there is no internal pull-up, and MultiKey/42i would interpret a low as the constant stream of PS/2 Mouse data (data errors to be precise). Finally, the PS/2 Mouse interrupt request line (IRQ12) has been added to allow the Mouse interrupt service routine to clear the Keyboard Controller of data if a Mouse Command is ever issued. Without IRQ12, the Keyboard Controller would hang while waiting for the System to read the Mouse Command timeout codes.

Figure 2-2. MultiKey/42i AT Platform without a PS/2 Mouse.



In Figure 2-3, there is the addition of the AUXCLK, xAUXDATA, xAUXCLK, & AUXDATA pins along with two more open-collector inverters to provide PS/2 Mouse support. The device connectors shown are the Mini-DIN connectors compatible with the PS/2 Style Keyboards and Mice.

Figure 2-3. MultiKey/42i AT Platform with a PS/2 Mouse.



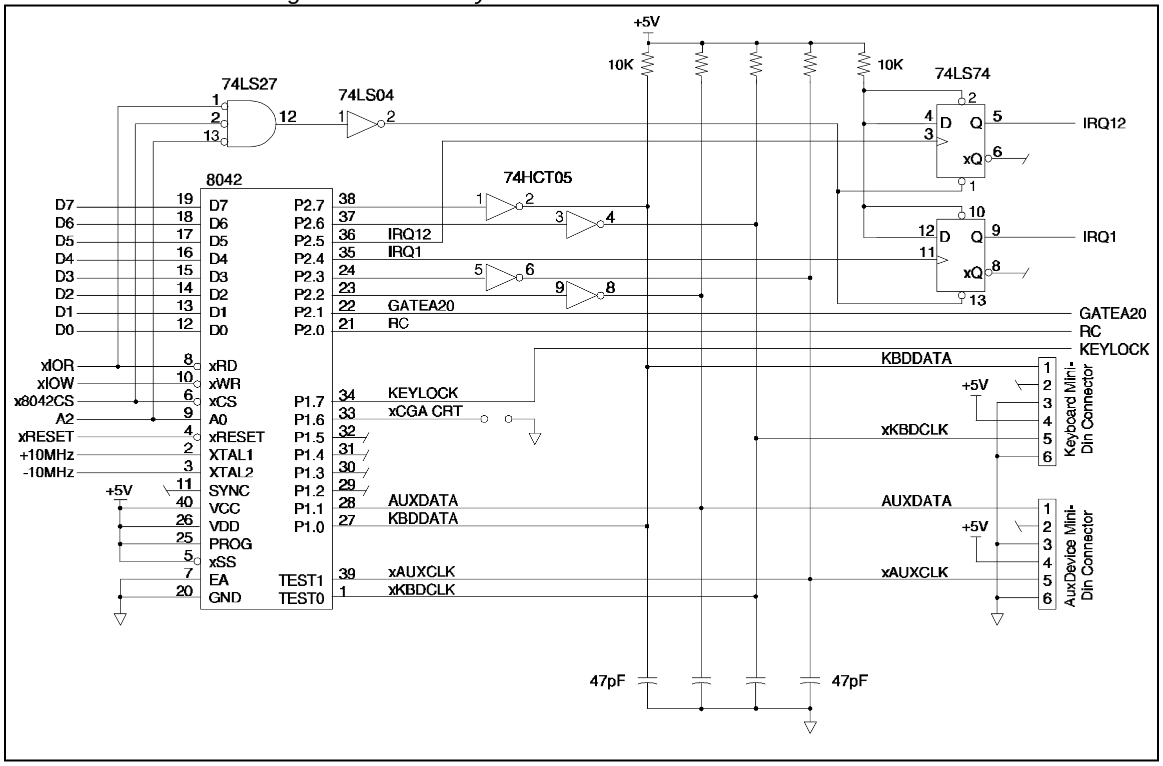
Two flip/flops and a decode of a Port60h read are added in Figure 2-4. The decode of the Port60h read clears the flip/flops to support the “level sensitive” Interrupts on the PS/2 Platform.

The flip/flops are added to guarantee correct operation on fast Systems where the MultiKey/42i software emulation of the flip/flops may not respond fast enough to prevent a second Interrupt from being generated.

This paragraph is intended to resolve some of the confusion that has resulted from the interrupt support through IRQ1 and IRQ12 pins. After the execution of the EN FLAGS instruction on the original AT Platform, the 8042 was put into a mode where the Output Buffer Full (OBF) and the Input Buffer Full (IBF) status register flags were reflected on P2.4 & P2.5. The OBF flag & P2.4 were raised when data was placed in the Output Buffer and lowered when the system read Port60h. On the original PS/2 Platform, P2.5 was needed for the Mouse Interrupt line so IBM designed flip/flops on those pins as shown in Figure 2-5, Schematic 1. The software pulsed both pins as shown in Figure 2-9, Case 1, setting the flip/flops. The flip/flops were cleared by a read of Port60h, this mimics the original AT Platform 8042.

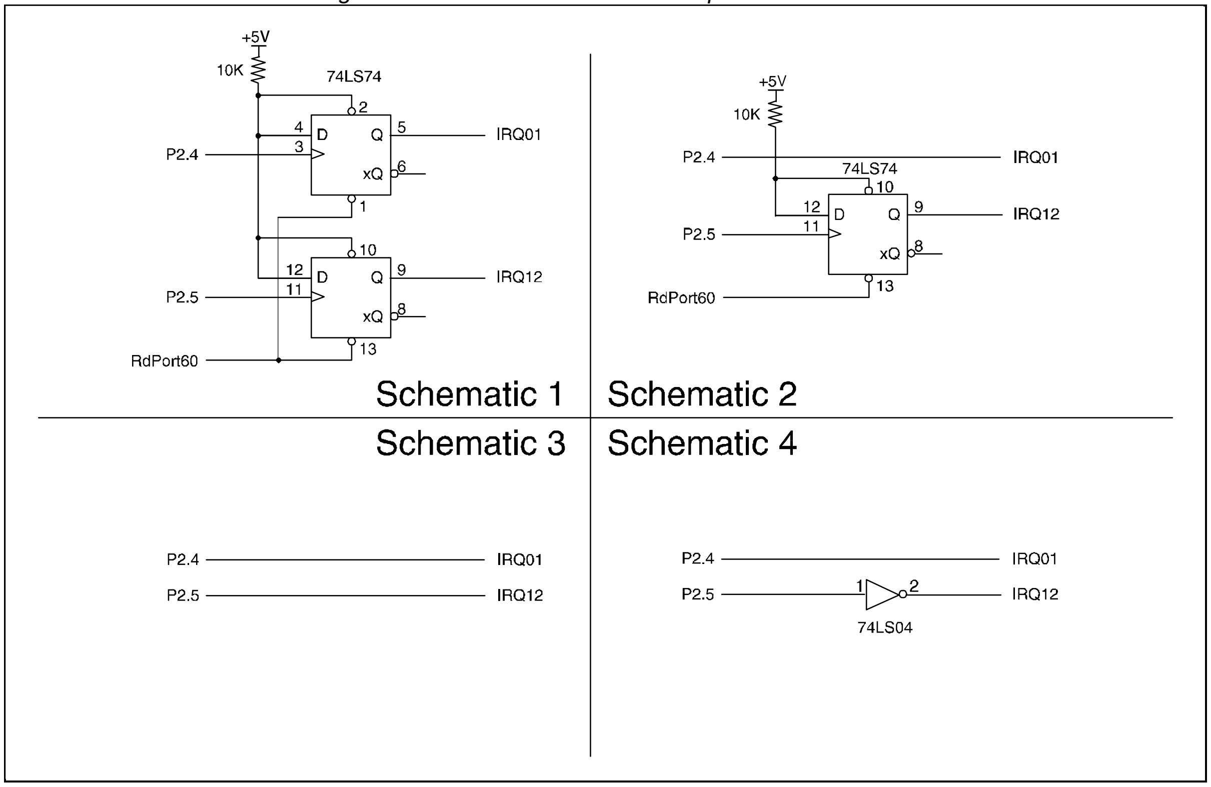
Some additional confusion resulted from the presence, on the PS/2 Platform, of a level sensitive Interrupt Controller while it lacked edge sensitive Interrupt Controller, like the AT Platform. The reason for this apparent contradiction is that an edge sensitive PIC (Programmable Interrupt Controller) is not like other edge triggered devices, it requires the Interrupt Request Line to remain high until the first INTA (Interrupt Acknowledge) cycle.

Figure 2-4. MultiKey/42i PS/2 Platform with a PS/2 Mouse.



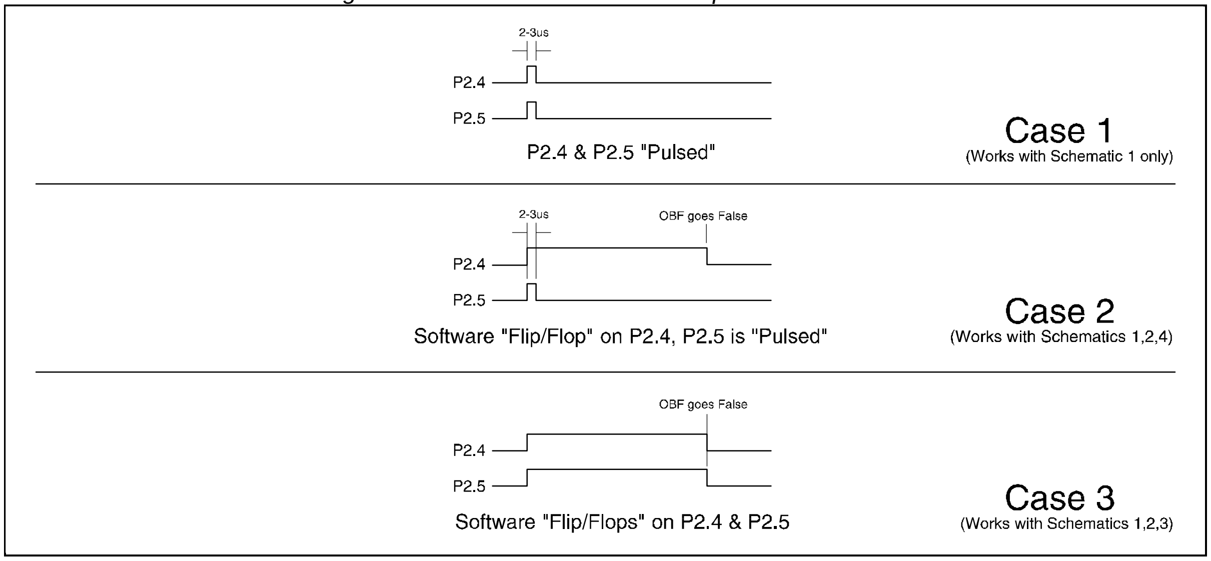
MultiKey/42i is built with Software flip/flops as its default, allowing it to work with Schematics 1, 2, and 3 in Figure 2-5. Schematic 4 is an old design of an EISA system and will not work with the Software Flip/Flop.

Figure 2-5. 8042 with Mouse Interrupt Hardware.



To work with Schematic 4, the Keyboard Controller must be configured as Case 2 of Figure 2-6.

Figure 2-6. 8042 with Mouse Interrupt Software.


2-3us
P2.4
P2.5
P2.4 & P2.5 "Pulsed"
Case 1
(Works with Schematic 1 only)
OBF goes False
Software "Flip/Flop" on P2.4, P2.5 is "Pulsed"
Case 2
(Works with Schematics 1,2,4)
Software "Flip/Flops" on P2.4 & P2.5
Case 3
(Works with Schematics 1,2,3)

The 10k Ohm resistor pull-ups on the Keyboard and PS/2 Mouse interface provide compatible drives with that of the IBM AT and the IBM PS/2 8042 designs (which keyboard manufactures expect).

KeyLock is available on all platforms and works in conjunction with the password security. If the KeyLock feature is not needed make sure P1.7 is tied high. The Jumper status is read by the System BIOS with the 0C0h Command (Read Input Port).

2.3 Pin Control Task Definition

One of the basic structures throughout the MultiKey/42i configuration is the Pin Control Task variable. The Pin Control Task variable defines what to do when an event (HotKey detection, Inactivity Timer expires, Activity is restored, Security is enabled, Security is disabled, or an External Input Event) occurs. The Pin Control Task variable is two bytes in size and its definition is shown in Table 2-1.

If the function "equal set/clear Port1 pins and BitF" is True after 2.4ms BitE of the Pin Control Task is XORed and the task re-performed. In addition the Pulse Pin Task does not changed when restoring to the original Keyboard Controller State. For example: If P1.3 is lowered when the Inactivity Timer expires, P1.3 will be raised when Activity is restored. However, if P1.3 is pulsed low when the Inactivity Timer expires, P1.3 will be again pulsed low when Activity is restored.

Therefore, if the Pin Control Task is a function other than setting or clearing the Port 1 pins, and it is desired not to reset the function when restoring the original Keyboard Controller State, then set the Pin Control Task BitF True. For example: If Security is enabled when the Inactivity Timer expires and BitF is False, then Security will be disabled when Activity is restored. However, if Security is enabled when the Inactivity Timer expires and BitF is True, then Security will not change when Activity is restored.

Table 2-1. Pin Control Task Definition.

Bit	Description
F	Pulse Port Pin Function after 2.4ms
E	Set or Clear Function
D-C	Type of Function (2 bits) 11 - Set or Clear Standby mode 10 - Toggle RAM/ROM Conversion Table 01 - Set or Clear Security 00 - Set or Clear Port 1 Pin
B-8	Function Number, range 0 - 15 (4 bits)
7-0	Port 1 Pin Data Mask (8 bits)

2.4 Standard Memory Map

MultiKey/42i allows the PhoenixBIOS or the OEM Keyboard utilities to read, except for the Password storage & Memory Index locations, the RAM with the extended commands 0B8h through 0BBh. The same RAM locations can be written with the extended commands 0B8h through 0BBh, only until one or both Passwords are loaded. The MultiKey/42i Memory Map is detailed in Table 2-2. Bit definitions for the RAM variables are included in the table. These bits described all the diagnostic as well as state saving/restoring information needed to understand the MultiKey/42i internal states.

Table 2-2. Memory Map. (sheet 1 of 3)

Symbol	RAM Location (Range)	Description
TEMP	00h-01h	Temporary Subroutine Scratch Registers (2 bytes)
KCMISC	02h	Keyboard Controller Miscellaneous Flags Bit7 - Auxiliary Expecting Response (bit1) Bit6 - Auxiliary Expecting Response (bit0) Bit5 - Keyboard Expecting Response (bit1) Bit4 - Keyboard Expecting Response (bit0) Bit3 - Auxiliary Expecting Four Responses Bit2 - No D2h Command Password checking Bit1 - Password Loaded, Memory is Read-Only Bit0 - Security is Enabled
KCSTATE	03h	Keyboard Controller State Flags Bit7 - OBF Data is not pending Bit6 - Internal Device Command flag Bit5 - Auxiliary Device Disabled Bit4 - Keyboard Device Disabled Bit3 - Use RAM Scan Code Conversion Table Bit2 - Not Waiting for Keyboard LED Data Bit1 - AT Environment (0=PS/2) Bit0 - Keyboard/Auxiliary Ports Not Swapped
TEMP	04h-05h	Temporary Scratch Register
TIMEOUT	06h	Keyboard Controller Timeout Flags Bit7 - STS7:Parity Error Bit6 - STS6:Timeout Error Bit5 - STS5:Auxiliary Device Output Buffer Full Bit4 - STS4:Security is Inactive Bit3 - Reserved Bit2 - Transmission Internal Bit1 - Transmission Type (bit 1) Bit0 - Transmission Type (bit 0)
TEMP	07h	Temporary Scratch Register
STACK	08h-017h	Processor Stack (16 bytes)
KSTATE1	018h	Keyboard Scan Code Set and LED State Bit7 - Keyboard Disabled at Device Bit6 - Reserved Bit5 - Scan Code Set Bit1 Bit4 - Scan Code Set Bit0 Bit3 - Reserved Bit2 - Caps Lock LED Bit1 - Num Lock LED Bit0 - Scroll Lock LED
* Indicates an IBM defined RAM location which is initialized, but not used.		

Table 2-2. Memory Map. (sheet 2 of 3)

Symbol	RAM Location (Range)	Description
KSTATE2	019h	Keyboard Typematic Delay and Rate Bit7 - Transparent Security Mode Bit6 - Typematic Delay Bit1 Bit5 - Typematic Delay Bit0 Bit4 - Typematic Rate Bit4 Bit3 - Typematic Rate Bit3 Bit2 - Typematic Rate Bit2 Bit1 - Typematic Rate Bit1 Bit0 - Typematic Rate Bit0
HOTKEYS	01Ah	HotKey State flags Bit7 - HotKey Work Pending Bit6 - Hold Key1 Active Bit5 - Hold Key2 Active Bit4 - HotKey5 Key Active Bit3 - HotKey4 Key Active Bit2 - HotKey3 Key Active Bit1 - HotKey2 Key Active Bit0 - HotKey1 Key Active
HOTTASK	01Bh	Detect HotKey Pending Task Offset
TST1PIN	01Ch	External Input Event Pin Mask (PIN1TSK)
TST2PIN	01Dh	External Input Event Pin Mask (PIN2TSK)
TMRFLGS	01Eh	Timer Miscellaneous State flags Bit7 - Flashing LED Counter (bit1) Bit6 - Flashing LED Counter (bit0) Bit5 - Reserved Bit4 - Reserved Bit3 - Flashing LED Task Pending Bit2 - Keyboard Controller Suspended Bit1 - KEY5TSK is only for HotKey 5 Bit0 - Flashing LED when Suspended
P1VALUE	01Fh	The Port 1 Shadow Latch Register
KCCB	020h	Keyboard Controller Command Byte Bit7 - Reserved Bit6 - Convert Scan Codes Bit5 - Auxiliary Disabled Bit4 - Keyboard Disabled Bit3 - Reserved Bit2 - System Flag Bit1 - Auxiliary Interrupt Enabled Bit0 - Keyboard Interrupt Enabled
RETRY*	021h	Number of times to Resend a Transmission
KBDRSP*	022h	If not 0, expect response from Keyboard
KSRSND*	023h	Count of RESENDS sent to Keyboard
PENDING	024h	Storage for the OBF Pending Data
INIT*	025h	IBM RESERVED
LEDDATA	026h	Storage for the Flashing/Rotating LED Pattern
TMRATE1	027h	Timer value 380µs, Device Bit Time
TMRATE2	028h	Timer value 2.4ms, Byte Receive Time
TMRATE3	029h	Timer value 11.7ms, Start Bit Time
INIT*	02Ah-02Ch	IBM RESERVED (3 bytes)
BREAK	02Dh	Break-Code (00h or 80h) from Keyboard
LOCOUNT	02Eh	Compensation Timer (0.12 seconds)
MDCOUNT	02Fh	Mid-Range Timer (30.0 seconds)
AUXRSP*	030h	If not 0, expect response from AuxDevice
ARESND*	031h	Count of RESENDs sent to AuxDevice
* Indicates an IBM defined RAM location which is initialized, but not used.		

Table 2-2. Memory Map. (sheet 3 of 3)

Symbol	RAM Location (Range)	Description
P1INPUT	032h	The lasted checked Port 1 Input value
PWNULL1	033h	Sent when Password enabled (if not 0)
PWNULL2	034h	Sent when Password disabled (if not 0)
FUNCTION	035h	Interrupt Function Request Value
PWSCAN1	036h	Ignored Scan Code when Password = enabled
PWSCAN2	037h	Ignored Scan Code when Password = enabled
TMRATE4	038h	Timer value 0.12s, Compensation Time
TMRATE5	039h	Timer value 30s-128m, Inactivity Time
HICOUNT	03Ah	Inactivity Timer (range 30s to 128m)
	03Bh-03Fh	IBM RESERVED (5 bytes)
KEY1TSK	040h-041h	HotKey1 Pin Control Task Value (2 bytes)
KEY2TSK	042h-043h	HotKey2 Pin Control Task Value (2 bytes)
KEY3TSK	044h-045h	HotKey3 Pin Control Task Value (2 bytes)
KEY4TSK	046h-047h	HotKey4 Pin Control Task Value (2 bytes)
KEY5TSK	048h-049h	HotKey5 & Inactivity Timer Pin Control Task Value (2 bytes)
LCK1TSK	04Ah-04Bh	Normal Password Pin Control Task Value (2 bytes)
LCK2TSK	04Ch-04Dh	Extended Password Pin Control Task Value (2 bytes)
TMR1TSK	04Eh-04Fh	Inactivity Timer Pin Control Task Value (2 bytes)
PIN1TSK	050h-051h	External Input Event1 Pin Control Task Value (2 bytes)
PIN2TSK	052h-053h	External Input Event2 Pin Control Task Value (2 bytes)
HOTKEY1	054h	HotKey1 Scan Code Storage
HOTKEY2	055h	HotKey2 Scan Code Storage
HOTKEY3	056h	HotKey3 Scan Code Storage
HOTKEY4	057h	HotKey4 Scan Code Storage
HOTKEY5	058h	HotKey5 Scan Code Storage
HLDKEY1	059h	1st Hold Key Scan Code Storage
HLDKEY2	05Ah	2nd Hold Key Scan Code Storage
INDEX	05Bh	MultiKey Memory Index
PW1INDX	05Ch	Normal Password Index
PW1AREA	05Dh-06Dh	Normal Password Storage Area (17 bytes)
PW2INDX	06Eh	Extended Password Index
PW2AREA	06Fh-07Fh	Extended Password Storage Area (17 bytes)
SCANTBL	080h-0FFh	RAM loaded Scan Code Conversion Table (128 bytes)
* Indicates an IBM defined RAM location which is initialized, but not used.		

2.5 Default Scan Code Conversion Table

Table 2-3 lists the content of the default Scan Code Conversion table. This table is stored in memory locations 080h through 0FFh, see Table 2-2 (Symbol SCANTBL).

Table 2-3. Default Scan Code Conversion Table. (sheet 1 of 3)

Index	Value	Description
000h	0FFh	Error (Overrun)
001h	043h	F9
002h	041h	F7
003h	03Fh	F5
004h	03Dh	F3
005h	03Bh	F1
006h	03Ch	F2
007h	058h	F12
008h	064h	Reserved
009h	044h	F10
00Ah	042h	F8
00Bh	040h	F6
00Ch	03Eh	F4
00Dh	00Fh	Tab
00Eh	029h	~ '
00Fh	059h	Reserved
010h	065h	Reserved
011h	038h	Left Alt
012h	02Ah	Left Shift
013h	070h	Reserved
014h	01Dh	Left Ctrl
015h	010h	Q
016h	002h	! 1
017h	05Ah	Reserved
018h	066h	Reserved
019h	071h	Reserved
01Ah	02Ch	Z
01Bh	01Fh	S
01Ch	01Eh	A
01Dh	011h	W
01Eh	003h	@ 2
01Fh	05Bh	Reserved
020h	067h	Reserved
021h	02Eh	C
022h	02Dh	X
023h	020h	D
024h	012h	E
025h	005h	$ 4
026h	004h	# 3
027h	05Ch	Reserved
028h	068h	Reserved
029h	039h	Space
02Ah	02Fh	V
02Bh	021h	F
02Ch	014h	T
02Dh	013h	R
02Eh	006h	% 5
02Fh	05Dh	Reserved

Table 2-3. Default Scan Code Conversion Table. (sheet 2 of 3)

Index	Value	Description
030h	069h	Reserved
031h	031h	N
032h	030h	B
033h	023h	H
034h	022h	G
035h	015h	Y
036h	007h	^ 6
037h	05Eh	Reserved
038h	06Ah	Reserved
039h	072h	Reserved
03Ah	032h	M
03Bh	024h	J
03Ch	016h	U
03Dh	008h	& 7
03Eh	009h	* 8
03Fh	05Fh	Reserved
040h	06Bh	Reserved
041h	033h	< ,
042h	025h	K
043h	017h	I
044h	018h	O (upper case letter o)
045h	00Bh	) 0 (number zero)
046h	00Ah	(9
047h	060h	Reserved
048h	06Ch	Reserved
049h	034h	> .
04Ah	035h	? /
04Bh	026h	L
04Ch	027h	: ;
04Dh	019h	P
04Eh	00Ch	_ -
04Fh	061h	Reserved
050h	06Dh	Reserved
051h	073h	Reserved
052h	028h	" '
053h	074h	Reserved
054h	01Ah	{ [
055h	00Dh	+ =
056h	062h	Reserved
057h	06Eh	Reserved
058h	03Ah	Caps Lock
059h	036h	Right Shift
05Ah	01Ch	Return
05Bh	01Bh	}]
05Ch	075h	Reserved
05Dh	02Bh	| \ (US) ~ # (102-key)
05Eh	063h	Reserved
05Fh	076h	Reserved

Table 2-3. Default Scan Code Conversion Table. (sheet 3 of 3)

Index	Value	Description
060h	055h	Fn (Phx Special)
061h	056h	| \ (102-key)
062h	077h	Reserved
063h	078h	Reserved
064h	079h	Reserved
065h	07Ah	Reserved
066h	00Eh	Backspace
067h	07Bh	Reserved
068h	07Ch	Reserved
069h	04Fh	1 End
06Ah	07Dh	Reserved
06Bh	04Bh	4 Left Arrow
06Ch	047h	7 Home
06Dh	07Eh	Reserved
06Eh	07Fh	Reserved
06Fh	06Fh	Reserved
070h	052h	0 Ins
071h	053h	. Del
072h	050h	2 Down Arrow
073h	04Ch	5
074h	04Dh	6 Right Arrow
075h	048h	8 Up Arrow
076h	001h	Esc
077h	045h	NumLock
078h	057h	F11
079h	04Eh	+
07Ah	051h	3 PgDn
07Bh	04Ah	-
07Ch	037h	*
07Dh	049h	9 PgUp
07Eh	046h	Scroll Lock
07Fh	054h	Sys Req (84-key only)

Chapter 3
MultiKey/42i Software Interface

The command set supported by the MultiKey/42i code is a superset of the IBM-compatible standard command set. All standard IBM commands are supported.

3.1 Command Invocation

The System writes commands to Port64h; the data associated with the command is written to Port60h. The System reads all auxiliary device (PS/2 mouse) and keyboard data at Port60h. The System reads the 8042 status at Port64h. Keyboard commands and data are written to Port60h. Auxiliary Device commands are written to Port60h after the MultiKey/42i Write Auxiliary Device (0D4h) Command; Auxiliary Device Data is sent with the same procedure.

The 8042 Status Register (read of Port 64h) indicates whether the 8042 is ready to accept another command or if data is ready from the last command. The System can only send data or commands to the 8042 if the IBF flag (Input Buffer Full, Bit1 of the Status Register) is false. The data from the 8042 is valid only if the OBF flag (Output Buffer Full, Bit0 of the Status Register) is true. Before issuing a command to return data, the OBF and IBF should both be false. After waiting for the OBF flag to go true, the data is read from Port60h.

3.2 Status Register

The Status Register is an eight bit read only register accessed via Port64h. An IN on Port64h provides the status shown in Table 3-1.

Table 3-1. Status Register.

Bit	Default	Description
7	0	Parity Error 1 = last byte received had incorrect parity
6	0	General Timeout 1 = Last transmission timed out before completion
5	0	Auxiliary Device Output Buffer Full 1 = Auxiliary output buffer contains data from the Auxiliary Device
4	0	Inhibited Switch 1 = The devices are uninhibited 0 = Password or Keylock is enabled
3	1	Command/Data (F1) 1 = System wrote to Port64h 0 = System wrote to Port60h
2	1	System Flag (F0) Value = Value of the System bit in the Keyboard Controller Command Byte
1	0	Input Buffer Full (IBF) 1 = Input buffer contains data for the Keyboard Controller
0	0	Output Buffer Full (OBF) 1 = Output buffer contains data for the System

3.3 Standard Commands

Phoenix Technologies MultiKey/42i supports the Standard Command Set described in Table 3-2.

Table 3-2. Standard Command Set.

Command	Description
00h-1Fh	Read the contents of the designated RAM locations (20h-3Fh) and send it to System
20h-3Fh	Read the contents of the designated RAM locations (20h-3Fh) and send it to System
40h-5Fh	Get a byte of data from System and write into one of locations (20h-3Fh)
60h-7Fh	Get a byte of data from System and write into one of locations (20h-3Fh)
A4h	Test Normal Password Returns 0FAh if Normal Password is loaded Returns 0F1h if Normal Password is loaded
A5h	Load Normal Password Loads Password until a '0' is received from the System (max. size = 16 characters)
A6h	Enable Password Security Enables the checking of keystrokes for a match with the passwords
A7h	Disable Auxiliary Device's Interface (PS/2 Mouse)
A8h	Enable Auxiliary Device's Interface (PS/2 Mouse)
A9h	Test Auxiliary Device Clock and Data
AAh	8042 Self Test Returns 055h if successful self test
ABh	Test Keyboard Clock and Data lines
ACh	Reserved (diagnostic dump)
ADh	Disable Keyboard Device's Interface
AEh	Enable Keyboard Device's Interface
C0h	Read the Input Port(P1) and send data to the System
C1h	Continuously puts the lower four bits of Port1 into the STATUS Register
C2h	Continuously puts the upper four bits of Port1 into the STATUS Register
D0h	Send Port2 value to the System
D1h	Only set/reset GateA20 line based on the System data Bit1
D2h	Send data back to the System as if it came from the Keyboard
D3h	Send data back to the System as if it came from the Auxiliary Device (PS/2 Mouse)
D4h	Output next received byte of data from System to Auxiliary Device (PS/2 Mouse)
E0h	Reports the state of the test outputs
FXh	Pulse only RC (the reset line) low for 6µs if the Command Byte is even

3.4 Extended Commands

Phoenix Technologies MultiKey/42i supports the Extended Command Set described in Table 3-3.

Table 3-3. Extended Command Set.

Command	Description
A2h	Test Extended Password Returns 0FAh if Extended Password is loaded Returns 0F1h if Extended Password is not loaded
A3h	Load Extended Password Loads Password until '0' is received from the System (max. size = 16 characters)
AFh	Set Inactivity Timer value from 0.5 to 128 minutes (zero disables timer)
B8h	Setup Phoenix Extended Memory Access INDEX
B9h	Get current Phoenix Extended Memory Access INDEX
BAh	Get current Phoenix Extended Memory referenced by INDEX Cannot read the Password Storage Area
BBh	If neither Password is loaded, write Phoenix Extended Memory referenced by INDEX. Cannot write the Password Storage Area. Once the Password is loaded, memory is locked
BCh - BDh	Read/Write the following MultiKey variables referenced by INDEX: LENGTH (0) Number of MultiKey variables KCSTATE (1) Keyboard Controller State flags TMRFLGS (2) Timer Miscellaneous State flags TMRATE1 (3) Timer value 380ms, Device Bit Time TMRATE2 (4) Timer value 2.4ms, Byte Receive Time TMRATE3 (5) Timer value 11.7ms, Start Bit Time TMRATE4 (6) Timer value 0.12seconds, Compensation Time TMRATE5 (7) Timer value 30 seconds to 128 minutes, Inactivity Time KSTATE1 (8) Keyboard Scan Code Set & LED state KSTATE2 (9) Keyboard Typematic Delay & Rate FUNCTION (A) Interrupt Function Request value
C7h	Sets Port1 bits corresponding to System data bits that are set
C8h	Clears Port1 bits corresponding to System data bits that are set
C9h	Sets Port2 bits corresponding to System data bits that are set
CAh	Clears Port2 bits corresponding to System data bits that are set
D5h	Read MultiKey code revision level (2 bytes). The digits automatically filled by PVCS Source Control System. MultiKey/42i revision level starts at 4.10 to distinguish it from MultiKey/42 (1.20+), MultiKey/C42 (2.10+) and MultiKey/42E (3.10+).
D6h	Read Version Information (2 bytes). MultiKey/42i returns Byte1 = 81h and Byte2 = ACh. Byte1 / Byte2 B7 - Processor Type (bit2) / B7 - IRQ12 software flip/flop B6 - Extended MultiKey Interface / B6 - cause IRQ before OBF B5 - KBD Scanning support / B5 - IRQ1 software flip/flop B4 - Power Down support / B4 - Reserved B3 - Processor type (bit1) / B3 - Clock speed (bit3) B2 - PS/2 mouse emulation / B2 - Clock speed (bit2) B1 - AT platform / B1 - Clock speed (bit1) B0 - Processor type (bit0) / B0 - Clock speed (bit0)
D7h	Read MultiKey model numbers (3 bytes). The CONVERT filled digits are in Hex format (for example: AAh, 55h, 00h)

3.5 Keyboard Controller Command Byte

The internal status is defined by the Keyboard Controller Command Byte (KCCB). The KCCB resides in RAM at location 20h. The KCCB can be read and written with the special commands listed in Table 3-4. Note that the KCCB is read using a 20h Command and written to using a 60h Command.

Table 3-4. Keyboard Controller Command Byte.

Bit	Default	Description
7	0	Reserved = 0
6	1	IBM PC Compatibility Mode 1 = Translate Scan Codes to IBM PC standard before passing it to the System 0 = Pass untranslated Scan Codes to the System
5	1	Disable Auxiliary Device 1 = Auxiliary Device's Interface disabled (PS/2 mouse)
4	0	Disable Keyboard 1 = Keyboard Device's Interface disabled
3	0	Reserved = 0
2	1	System Flag 1 = the System is executing POST as the result of a shutdown or warm boot 0 = the System is executing POST as the result of a cold boot NOTE: The value of this bit is written to the System Flag Bit of the Status Register (Bit2 of a read of Port64h)
1	0	Enable Auxiliary Output Buffer Full Interrupt 1 = An interrupt to System is generated when a byte is placed into the Auxiliary Output Buffer (IRQ12)
0	0	Enable Keyboard Output Buffer Full Interrupt 1 = An interrupt to System is generated when a byte is placed into the Output Buffer (IRQ1)

3.6 Keyboards and Auxiliary Device Commands

Any Command/Data written to Port60h is automatically transmitted to the Keyboard by the Keyboard Controller if MultiKey/42i is not in a waiting for data mode. See Table 3-5 for all Keyboard Commands. In the case of a two-byte Keyboard Command, for example, Set LEDs (0EDh), both the Command and Data are written to Port60h.

Table 3-5. Keyboard Commands.

Command	Description
EDh	Set LEDs
EEh	Echo
EFh	Invalid command
E0h	Select alternate scan code set
F1h	Invalid command
F2h	Read ID bytes
F3h	Set typematic delay and rate
F4h	Enable Keyboard
F5h	Disable Keyboard and set defaults
F6h	set defaults
F7h*	Set all keys typematic
F8h*	Set all keys make/break
F9h*	Set all keys make only
FAh*	Set all keys typematic/make/break
FBh*	Set key type typematic
FCh*	Set all keys type make/break
FDh*	Set key type make only
FEh	Resend the last transmission
FFh	BAT, Reset the defaults and buffers
* Commands F7h through FDh are normally used for Character Set 3	

The Auxiliary Device Command sequence is executed in two steps:

1. Write an 8042 Command D4h (Write Auxiliary Device) to Port64h.

2. Write Command/Data to Port60h.

The above sequence is executed twice for two-byte auxiliary device commands, such as the Set Scaling (0E7h) Command (see Table 3-6).

Table 3-6. Auxiliary Commands.

Command	Description	Command	Description
E6h	Reset scaling	F0h	Set remote mode
E7h	Set scaling	F1h	Invalid command
E8h	Set resolution	F2h	Read device type
E9h	Status Request	F3h	Set sampling rate
EAh	Set stream mode	F4h	Enable auxiliary device
EBh	Read data	F5h	Disable auxiliary device
ECh	Reset wrap mode	F6h	Set default values
EDh	Invalid command	F7h - FDh	Reserved
EEh	Set wrap mode	FEh	Resent
EFh	Invalid command	FFh	Reset

This page left blank.

Chapter 4
MultiKey/42i Configuration Utility

The Configuration Utility is designed to work with the Phoenix MultiKey/42i product. The MultiKey/42i Configuration Utility allows an OEM to configure and test the new Keyboard Controller configuration immediately. The configuration can also be saved as an assembler ASCII file, so it can be combined with the System BIOS routines (as shown in Chapter 5, MultiKey Keyboard Controller Routines).

4.1 Configuration Utility Overview

The MultiKey/42i Configuration Utility, CFG42i.EXE , supports:

- Automatically detects and gets the current MultiKey/42i Configuration
- Definition of the HotKey Scan Codes, Tasks and SMI Numbers
- Definition of the Input Pin Events, Tasks and SMI Numbers
- Definition of the Inactivity Timer, Configuration, Tasks and SMI Numbers
- Definition of the Dual Passwords, Configuration, Tasks and SMI Numbers
- Definition of the Port Usage, Clock Rate and the ROM/RAM Conversion Table

Figure 4-1. CFG42i.EXE Main Screen.

```
(c)Copyright°Phoenix°Technologies°1996:°MultiKey/42i°Configuration°(Ver°1.4)°°°°
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Key Ä¿
³ Active Key °°Ctrl°°³
³ Active Key º  Alt  ÃÄ HotKey Task ÄÄÄÄÄÄÄÄÄÄÂÄ SMI Ä¿
³   HotKey 1 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 2 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 3 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 4 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 5 º   Q   ³ Enable Security      ³  03h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Port Mask ÄÄÄÂÄ External Pin Task ÄÄÄÂÄ SMI Ä¿
³ InputPin 1 º      úúú      ³          úúú          ³  00h  ³
³ InputPin 2 º      úúú      ³          úúú          ³  00h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Timer Value ÄÂÄ Second Task ÄÂÄ Inactivity Indicator ÄÄÄÄÄÄÄÄÄ¿
³ Inactivity º      úúú      ³ Share HotKey5 ³ Flashing the Scroll Lock LED    ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Standby TaskÄÁÄÄÄÄÄÄÄÄÄÂÄ SMI ÄÅÄ Resume Task ÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Inactive 1 º Lower Port1 00001000b ³  01h  ³ Raise Port1 00001000b ³   02h   ³
³ Inactive 2 º Enable Security       ³  03h  ³ Enable Security       ³   04h   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Normal/Password 1 ÄÄÄÂÄ Extended/Password 2 ÄÂÄ Security Mode Ä¿
³   Password º   úúúúúúúúúúúúúúúú    ³   úúúúúúúúúúúúúúúú    ³ Block Commands  ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Enable Task ÄÄÄÄÄÄÄÄÄÅÄ SMI ÄÂÄ Disable TaskÄÁÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Security 1 º          úúú          ³  00h  ³          úúú          ³   01h   ³
³ Security 2 º          úúú          ³  00h  ³          úúú          ³   01h   ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Key ÄÂÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÅÄÄÄÄÄÄÄÄÄÁÄÄÂÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÅÄ Value Ä´
³ Ignore Key º  úúú  ³ D2h:Pswd test ³ Disabled ³  Send when Enabled ³   úúú   ³
³ Ignore Key º  úúú  ³ when P1.2=1   ÀÄÄÄÄÄÄÄÄÄÄÙ Send when Disabled ³   úúú   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Miscellaneous ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ Port Usage º Ports are not Swapped (Keyboard on Port0, Mouse on Port1)      ³
³ Clock Rate º Timer variables are based on a 12.0 MHz clock rate             ³
³ Conversion º Use ROM ScanCode Conversion Table                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ



         Controller identified as:  MultiKey/42i for the 8042 (v4.12)

ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
  1°°Help°°° 2°°Info°°° 3°°Color°° 4°°°°°°°°° 5°°Load°°° 6°°Save°°° Esc°Exit°°
```

4.1.1 Starting the Configuration Utility

The main Configuration Utility screen is shown in Figure 4-1; it is displayed by typing CFG42i /40 at the DOS prompt. Since the Configuration Utility issues extended MultiKey Keyboard Controller commands, the program should be run from the regular DOS prompt and not from a Windows' DOS Box. The Configuration Utility performs several functions when loading the program. The first function is to check the Command Line for switches (examples: /40 sets 40 line mode, /F fakes MultiKey/42i Hardware). The next function is to verify that this platform has a MultiKey/42i product (unless the /F switch was invoked). If the Configuration Utility does not find MultiKey/42i Hardware, the program immediately exits back to the DOS prompt, and displays the following message:

Figure 4-2. MultiKey/42i Configuration Error Message.

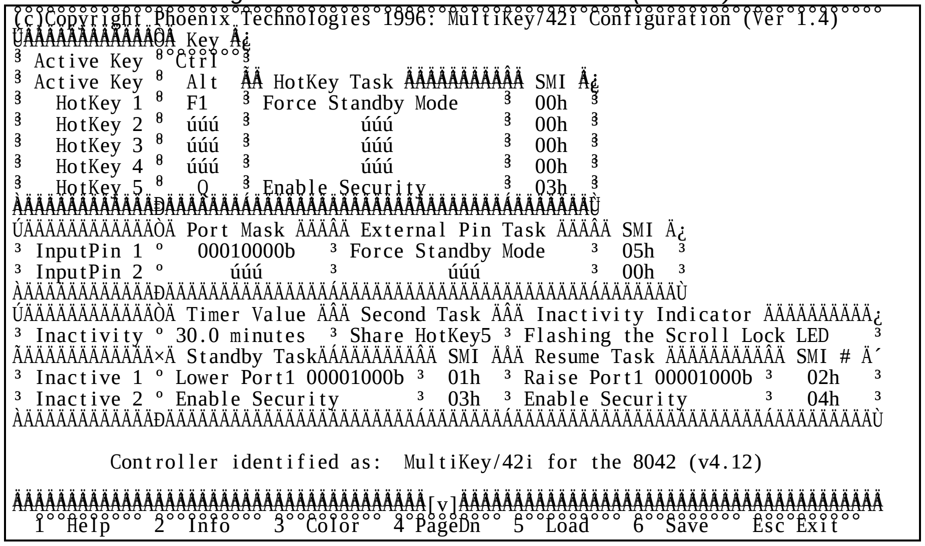
```
MultiKey/42i Configuration Utility (Ver 1.4).
Type CFG42i /? for Command Line Usage & Help.

ERROR: No MultiKey/42i processor found.
```

The Configuration Utility then evaluates the Command Line for a Configuration Filename. If a Configuration Filename is found it is loaded and the information is downloaded to the MultiKey/42i Keyboard Controller and the program immediate exits back to the DOS prompt. If no Configuration file is found, the current MultiKey/42i Configuration is read from the Keyboard Controller and displayed as the program data (see Figure 4-1).

4.1.2 The Main Screen Layout

The Configuration Utility screen (see Figure 4-1) is shown in 40 line mode, so all of the windows are visible at once. If the Configuration Utility is run in 25 line mode (see Figure 4-3), only the first three windows would be visible and Function key F4 would swap between the two pages.

Figure 4-3. CFG42i.EXE Main Screen. (25 lines)



```
(c)Copyright Phoenix Technologies 1996: MultiKey/42i Configuration (Ver 1.4)
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Key Ä¿
³ Active Key º  Ctrl  ³
³ Active Key º  Alt  ÃÄ HotKey Task ÄÄÄÄÄÄÄÄÄÄÂÄ SMI Ä¿
³   HotKey 1 º  F1   ³ Force Standby Mode    ³  00h  ³
³   HotKey 2 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 3 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 4 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 5 º   Q   ³ Enable Security       ³  03h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Port Mask ÄÄÄÂÄ External Pin Task ÄÄÄÂÄ SMI Ä¿
³ InputPin 1 º   00010000b   ³ Force Standby Mode    ³  05h  ³
³ InputPin 2 º      úúú      ³          úúú          ³  00h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Timer Value ÄÂÄ Second Task ÄÂÄ Inactivity Indicator ÄÄÄÄÄÄÄÄÄÄ¿
³ Inactivity º 30.0 minutes  ³ Share HotKey5 ³ Flashing the Scroll Lock LED    ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Standby TaskÄÁÄÄÄÄÄÄÄÄÄÂÄ SMI ÄÅÄ Resume Task ÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Inactive 1 º Lower Port1 00001000b ³  01h  ³ Raise Port1 00001000b ³   02h   ³
³ Inactive 2 º Enable Security       ³  03h  ³ Enable Security       ³   04h   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ

         Controller identified as:  MultiKey/42i for the 8042 (v4.12)

ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ[v]ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
  1 Help    2 Info    3 Color   4 PageDn   5 Load    6 Save    Esc Exit
```

There are five separate windows, the HotKey Configuration Window is the first window, followed by the External Input Event Window, the Inactivity Window, the Password Window, and finally the Miscellaneous Feature Window. The active window is indicated by the bright border color and the fact that the cursor is visible in that window. From the program initialization, the HotKey Configuration Window is the first window made active (or selected). Each window can be selected in turn with the TAB key. Even if the Configuration Utility was run in 25 line mode, the TAB key will swap to page two after the third window and back to page one after the fifth window.

4.1.3 Program Control Overview

The TAB key will select the next window. SHIFT key + TAB key will select the previous window. Once a window is selected (active), the separate ARROW keys and the number pad keys will move the cursor to the value or feature which requires changing. Each selected value or feature can be modified by pressing the ENTER key and selecting the desired options from the Dialog Box. The ESC key will abort the Dialog Box without changing the value or feature. Any numeric value can be immediately typed in and accepted by pressing the ENTER key. If either the ARROW keys or ESC key are pressed before the ENTER key, the value is restored to the original value.

Numeric values entered will be post-processed and may be modified if they are not in the valid range for that variable. Some values and features may have dependencies on other features and may alter the other values or features.

4.1.4 Program On-Line Help

Pressing the F1 key will provide Program Information for the currently selected window. Pressing the SHIFT key + the F1 key provide General Program Information (i.e. non-window specific), as shown in Figure 4-4. The General Information Help Screen provides additional background and user instructions. The General Help Screen has two pages, a program functional overview and a list of all active keys. The General Help also has a complete list of all the valid Command Line switches.

All of the specific window Help Screens are only one page long and at the top of the Help Screen remind the user how to get to the General Program Information Screen.

Figure 4-4. CFG42i.EXE General Information Help Screen.

```
°(c)Copyright°Phoenix°Technologies°1996:°MultiKey/42i°Configuration°(Ver°1.4)°°°
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Key Ä¿
³ Active Key º Ctrl  ³
³ Active Key º  Alt  ÃÄ HotKey Task ÄÄÄÄÄÄÄÄÄÂÄ SMI Ä¿
³   HotKey 1 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 2 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 3 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 4 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 5 º   Q   ³ Enable Security      ³  03h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Port Mask ÄÄÄÂÄ External Pin Task ÄÄÄÂÄ SMI Ä¿
³ InputPin 1 º     úúú     ³          úúú          ³  00h  ³
³ InputPin 2 º     úúú     ³          úúú          ³  00h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Timer Value ÄÂÄ Second Task ÄÂÄ Inactivity Indicator ÄÄÄÄÄÄÄÄÄ¿
³ Inactivity º     úúú     ³ Share HotKey5 ³ Flashing the Scroll Lock LED    ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Standby TaskÄÁÄÄÄÄÄÄÄÄÄÂÄ SMI ÄÅÄ Resume Task ÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Inactive 1 º Lower Port1 00001000b ³  01h  ³ Raise Port1 00001000b ³   02h   ³
³ Inactive 2 º Enable Security       ³  03h  ³ Enable Security       ³   04h   ³
ÀÉÍÍÍÍÍÍÍÍÍÍÍ»ÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
É¼   Help   ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º                          General Program Information          (Page 1 of 2) º
º    Esc -- Exits to DOS or puts away the current active Filecard or Dialog.   º
º   Help -- Provides specific information for the currently Selected Window.   º
º   Info -- Displays the actual values of the Keyboard Controller variables.   º
º  Color -- Allows the Color and Monochrome attributes to be modified.         º
º   Load -- Loads pre-existing MultiKey/42i Configuration file from the Disk.  º
º   Save -- Saves the MultiKey/42i Configuration to an Ascii file on the Disk. º
º PrtScn -- Captures the Screen Image and appends it to the SCREEN.TXT file.   º
º    Tab -- Selects the Next (Shft Tab = Previous) set of Features (Window).   º
º Arrows -- Along with Home, End, PgUp, & PgDn keys, provide Cursor movement.  º
º  Enter -- Brings up a Dialog prompting the user for Data input allowing      º
º           the MultiKey/42i features to be configured.                        º
º                                                                              º
º > The Display Attributes will be saved to disk (.CFG file) and reloaded the  º
º   next time the program is run.  Deleting the .CFG file will reset Defaults. º
º                                                                              º
º Phoenix Technologies      <Esc> or <F1> Next Page                            º
ÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÐ
  1  Help    2°°Info°°° 3°°Color°° 4°°°°°°°°° 5°°Load°°° 6°°Save°°° Esc°Exit°°
```

4.1.5 Keyboard Controller Information

The Configuration Utility displays the MultiKey/42i Keyboard Controller information when Function key F2 info, is pressed. The information Dialog Box displays the Keyboard Controller variables that are affected by the various feature and value settings. These Keyboard Controller variables are primarily intended for the Keyboard Controller engineers, since the variable names corresponds to Keyboard Controller source code names. The two second beep is a “Basic program timing integrity” check.

4.1.6 Configuration Utility Screen Attributes

The Configuration Utility allows all of the screen text attributes to be modified by pressing Function key F3 Color. The Color Attribute Dialog Box displays two sets of numbers for each text type (TitleBar, Headers, and so on), the first column lists the Color Video mode values and the second lists the Mono Video mode values. These attributes will be saved to disk in the CFG42i.CFG file, so the next time the program is run these same attributes can be used.

To modify an attribute, first select the attribute with the ARROW keys, type in the new attribute number and press the ENTER key. Moving the cursor or pressing the ESC key before pressing the ENTER key, will restore the entry's original attribute. The upper nibble of the attribute number is the background color and the lower nibble of the attribute number is the foreground color. There are a total of 16 foreground colors. There is one special case background color (value = 8) which indicates a transparent background, where the foreground text is put over the existing background color. The most significant bit of the attribute traditionally indicates blinking, however, the VGA Video has been reprogram to allow 16 background color minus the transparent color giving a total of 15 background colors.

4.1.7 Saving the Configuration to Disk

The MultiKey Configuration can be saved to disk at anytime. Pressing Function key F6 Save, brings up the directory filecard as shown in Figure 4-5. For the first second, the top line of the filecard displays the directory sort parameters (example -> D:\MULTIKEY\UTILS\CFG42I\????????.42I). After one second the directory is made active, any file or directory can be selected by pressing the ENTER key. Selecting a directory will change directories and resort the files. Selecting a file will allow a file to be overwritten.

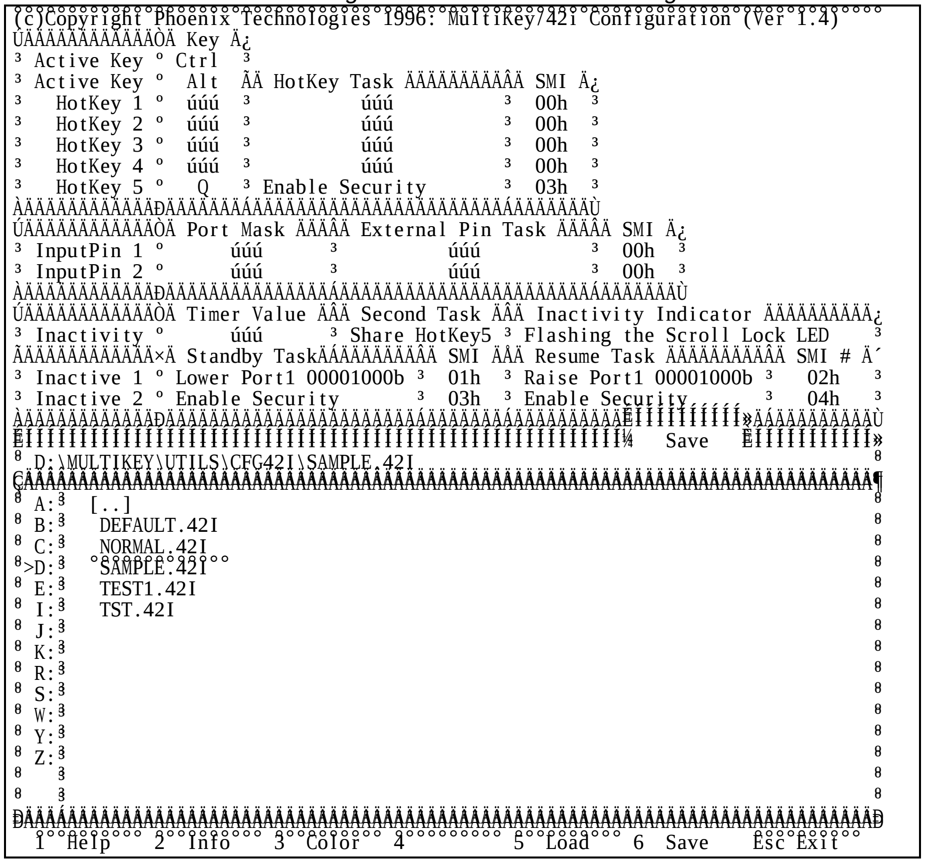


Figure 4-5.   CFG42i.EXE Saving the File to Disk.

```
(c)Copyright Phoenix Technologies 1996: MultiKey/42i Configuration (Ver 1.4)
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Key Ä¿
³ Active Key º Ctrl  ³
³ Active Key º  Alt  ÃÄ HotKey Task ÄÄÄÄÄÄÄÄÄÂÄ SMI Ä¿
³   HotKey 1 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 2 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 3 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 4 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 5 º   Q   ³ Enable Security       ³  03h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Port Mask ÄÄÄÂÄ External Pin Task ÄÄÄÂÄ SMI Ä¿
³ InputPin 1 º      úúú      ³          úúú          ³  00h  ³
³ InputPin 2 º      úúú      ³          úúú          ³  00h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Timer Value ÄÂÄ Second Task ÄÂÄ Inactivity Indicator ÄÄÄÄÄÄÄÄÄ¿
³ Inactivity º      úúú      ³ Share HotKey5 ³ Flashing the Scroll Lock LED    ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Standby TaskÄÁÄÄÄÄÄÄÄÄÄÂÄ SMI ÄÅÄ Resume Task ÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Inactive 1 º Lower Port1 00001000b ³  01h  ³ Raise Port1 00001000b ³   02h   ³
³ Inactive 2 º Enable Security       ³  03h  ³ Enable Security       ³   04h   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÉÍÍÍÍÍÍÍÍÍÍÍ»ÁÄÄÄÄÄÄÄÄÄÙ
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼   Save   ÈÍÍÍÍÍÍÍÍÍÍÍ»
º D:\MULTIKEY\UTILS\CFG42I\SAMPLE.42I                                          º
ÇÄÄÂÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¶
º A:³  [..]                                                                    º
º B:³   DEFAULT.42I                                                            º
º C:³   NORMAL.42I                                                             º
º>D:³   SAMPLE.42I                                                             º
º E:³   TEST1.42I                                                              º
º I:³   TST.42I                                                                º
º J:³                                                                          º
º K:³                                                                          º
º R:³                                                                          º
º S:³                                                                          º
º W:³                                                                          º
º Y:³                                                                          º
º Z:³                                                                          º
º   ³                                                                          º
º   ³                                                                          º
ÐÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÐ
  1  Help    2  Info    3  Color    4           5  Load    6  Save    Esc Exit
```

Pressing the TAB key will move the cursor to the File Specification area, which allows the user to specify the path and filename. If wildcards are included in the filename, the directory will be resorted to the new sort parameters. Pressing the TAB key again will move the cursor to the Drive Select area. If the cursor is moved and a new drive is selected the directory will be resorted again.

Pressing the Function key F1 help, will provide help for the directory filecard. Pressing the ESC key or Function key F6 Save, will abort the Saving process. The saved MultiKey/42i Configuration file can be recalled at anytime, using Function key F5 Load. The MultiKey/42i Configuration file (.42i) is saved as an ASCII file, so it can be modified by a normal ASCII editor, if desired. This allows the data to be incorporated into the System BIOS very easily. In Chapter 5, the kbdCfgController routine is a sample routine which configures the MultiKey/42i Keyboard Controller based on the .42i file data. The format of the . 42i file is shown in Figure 4-6.

Figure 4-6.   CFG42i.EXE .42i File Format.

```
;===========================================================================
;            M U L T I K E Y / 4 2 I   C O N F I G U R A T I O N
;===========================================================================

kcState         DB      001h    ; (1) Keyboard Controller State flags
kcTmrFlgs       DB      000h    ; (2) Timer Miscellaneous State flags
kcTmRate1       DB      0F7h    ; (3) Timer value 380us, Device Bit time
kcTmRate2       DB      0C4h    ; (4) Timer value 2.4ms, Byte Receive time
kcTmRate3       DB      000h    ; (5) Timer value 11.7ms, Start Bit time
kcTmRate4       DB      0CFh    ; (6) Timer value 0.5s, Flashing LED time
kcTmRate5       DB      000h    ; (7) Timer value 30s-128m, Inactivity time
kcKState1       DB      000h    ; (8) Keyboard ScanCode Set & LED State
kcKState2       DB      000h    ; (9) Keyboard Typematic Delay & Rate

kcMisc          DB      004h    ; Keyboard Controller Miscellaneous flags
kcTst1Pin       DB      000h    ; External Input Event Pin mask (PIN1TSK)
kcTst2Pin       DB      000h    ; External Input Event Pin mask (PIN2TSK)
kcPswNull1      DB      000h    ; Sent when Password enabled (if not 0)
kcPswNull2      DB      000h    ; Sent when Password disabled (if not 0)
kcPswScan1      DB      000h    ; Ignored ScanCode when Password = enabled
kcPswScan2      DB      000h    ; Ignored ScanCode when Password = enabled

kcHotTasks      DW      000F0h, 00000h, 00000h, 00000h, 000D3h
kcLckTasks      DW      00000h, 00000h
kcTmrTask       DW      00801h
kcPinTasks      DW      00000h, 00000h

kcHotKeys       DB      03Bh, 000h, 000h, 000h, 010h, 01Dh, 038h

;---------------------------------------------------------------------------
```

4.2 MultiKey/42i Feature Support

MultiKey/42i is an 8042 product using only 2KBytes of ROM and 256Bytes of RAM. To accommodate these limitations, some of the features found in other MultiKey/42 products had to be removed. Due to the small amount of RAM, and the desire to provide a second Inactivity Timer Task, the second task is shared with HotKey5. If this feature is enabled, the HotKey5 Task will be invoked when the Inactivity Timer expires. Many designs require Quicklock (HotKey invoked Password Security) and also require Security to be invoked when the Inactivity Timer expires, this is the best shared HotKey 5 Task and Inactivity Task example. If all 5 HotKeys are required and nothing can be shared with the Inactivity Timer, then the Inactivity Timer must be limited to one task. If two Inactivity Timer Tasks are required, then only the first 4 HotKeys can be used and the fifth Scan Code value must be zero.

Another trade-off is the SMI function number generation. Most designs do not worry about generating SMI's for each task; but if they do, careful consideration must be given to the SMI function numbers. If any particular task invokes another task the SMI function number will be a result of the second task. If HotKey 5 Task is set to Enable Security, the HotKey 5 SMI value will be overwritten by the second Security Enabled SMI value (if Password Two is loaded), which will be overwritten by the first Security Enabled SMI value (if Password One is loaded); therefore the SMI value read after the HotKey 5 is invoked would most likely be the result of the first Security Enabled Task. Like Password Security, the second Inactivity Task is actually performed before the first Inactivity Task is performed. The SMI caused by the Inactivity Timer expiring would read an SMI function number value based on the first Inactivity Timer Standby Task. If both external Pin Event Tasks were enabled and set to the same pin(s), the first external Pin Event Task is performed before the second external Pin Event Task. One of the more complicated arrangements of SMI function numbers overwriting each other is shown in Table 4-1.

Table 4-1. SMI Function Number Values.

Function	Task	SMI
Input Pin 1 (monitoring P1.3)	Pulse Low Port 1 Pin 4	001h
Input Pin 2 (monitoring P1.3)	Force Standby Mode	003h
Inactivity Timer 1	Enable Security	005h
Inactivity Timer 2	Lower Port 1 Pin 5	007h
Password Security 1	Lower Port 1 Pin 6	009h
Password Security 2	Lower Port 1 Pin 7	00Bh

The Input Pin 1 Task would set the SMI value to 001h, which would be overwritten by Input Pin 2 Task so the SMI value would be 003h, since both functions are monitoring the same pin. However, since the Input Pin 2 Task causes the Inactivity Timer to immediately expire (Force Standby Mode), the second Inactivity Timer Task overwrites the SMI value to 007h which in turn is overwritten to 005h by the first Inactivity Timer Task. And since the Inactivity Timer Task invokes Security, the second Security Task will overwrite the SMI value to 00Bh which is finally overwritten by the first Security Task to 009h. So the Pulse Low of Port 1 Pin 4 which caused the SMI would generate a value of 009h. The functional equivalent shown in Table 4-2 is much more clear with the Tasks rearranged.

Table 4-2. SMI Function Number Values (tasks rearranged).

Function	Task	SMI
Input Pin 1 (monitoring P1.3)	Force Standby Mode	001h
Input Pin 2 (monitoring P1.3)	Pulse Low Port 1 Pin 4	003h
Inactivity Timer 1	Lower Port 1 Pin 5	005h
Inactivity Timer 2	Enable Security	007h
Password Security 1	Lower Port 1 Pin 6	009h
Password Security 2	Lower Port 1 Pin 7	00Bh

The same Pulse Low of Port 1 Pin 4 which caused the SMI would generate a value of 003h and the other SMI overwrites are secondary. In addition, the Inactivity Timer Tasks will generate a SMI value directly from the first Inactivity Timer Task and the Password Security Task SMI values will not confuse the issue.

Along with picking the correct Function and Task pairings, careful consideration must be given to values of SMIs chosen. Note that some functions have two separate Tasks, an Enable and a Disable Task, or Standby and a Resume Task. These functions' SMI values are automatically produced from the first Task by incrementing the SMI value. One way to avoid overlapping numbers is to assign odd numbers to all Tasks requiring an SMI see Table 4-1 and Table 4-2.

4.2.1 Configuring HotKeys and Tasks

Once the HotKey & Task window is active (i.e. the window is highlighted and cursor is enabled), select the HotKey or the Activate key to be modified with the ARROW keys and press the ENTER key. The Activate keys are the keys held down in addition to the HotKey. The default values for the activate keys are left CTRL key and left ALT key. Once selected a dialog will prompt the user to select any key on the external Desktop Keyboard that is not an extended key. The extended keys are the separate arrows and the separate cursor control keys, basically any key added between the original IBM AT 84-Key Keyboard and the 101-Key Keyboard. Extended keys produce more than one Scan Code per Make/Break and cannot be used as a HotKey. The HotKey Scan Code dialog uses the right CTRL key to clear the HotKey entry and the right ALT key to abort the process. These keys are used since they are extended keys and cannot be used as the HotKey.

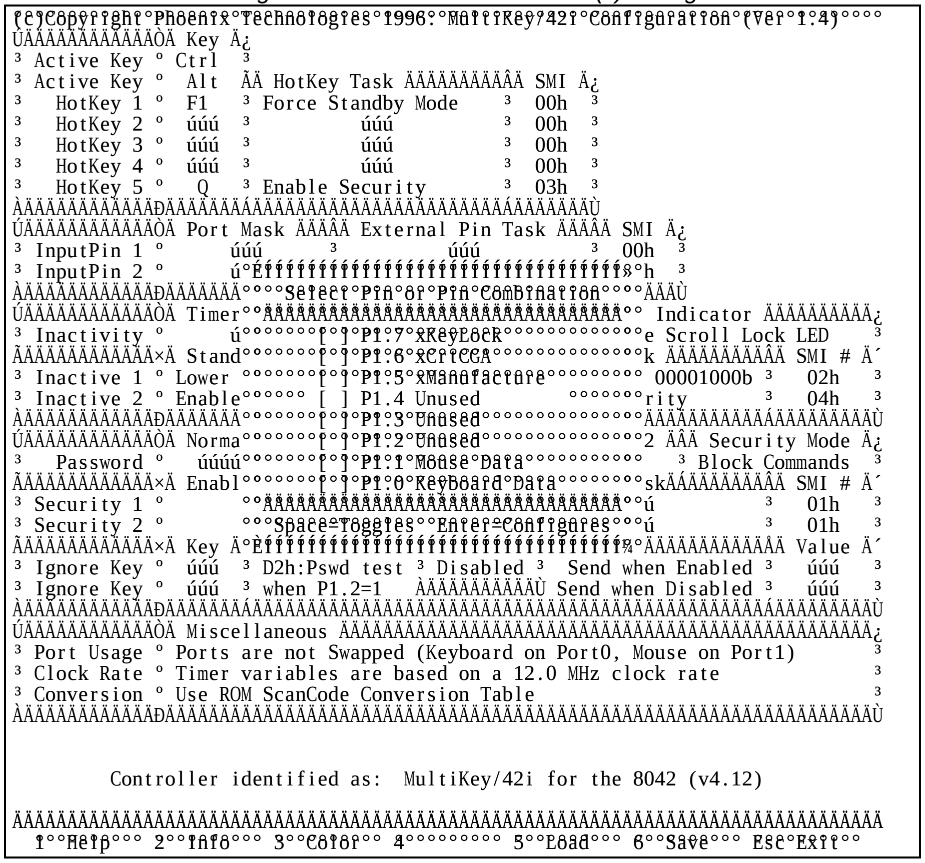
To configure the Task select the Task to be modified with the ARROW keys and press the ENTER key. A dialog will prompt the user to select the type of Task to be performed, as shown in Figure 4-7.

Figure 4-7. CFG42i.EXE HotKey Task Dialog.

```
°(c)Copyright°Phoenix°Technologies°1996:°MultiKey/42i°Configuration°(Ver°1.4)°°°
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Key Ä¿
³ Active Key º Ctrl  ³
³ Active Key º  Alt  ÃÄ HotKey Task ÄÄÄÄÄÄÄÄÄÄÂÄ SMI Ä¿
³   HotKey 1 º  F1   ³          úúú          ³  00h  ³
³   HotKey 2 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 3 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 4 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 5 º   Q   ³ Enable Security       ³  03h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Port Mask ÄÄÄÂÄ External Pin Task ÄÄÄÂÄ SMI Ä¿
³ InputPin 1 º      úúú      ³          úúú          ³  00h  ³
³ InputPin 2 º      úú°ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»°h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄ°°°°°°°°Select°HotKey°Task°°°°°°°°°ÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Timer °°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ°° Indicator ÄÄÄÄÄÄÄÄÄÄ¿
³ Inactivity º      úú°°°þ°Clear°Task°°°°°°°°°°°°°°°°°°°°°e Scroll Lock LED    ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Standb°°°þ°Lower°Port°1°pin...°°°°°°°°°°°°°k ÄÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Inactive 1 º Lower P°°°þ°Raise°Port°1°pin...°°°°°°°°°°°°° 00001000b ³   02h   ³
³ Inactive 2 º Enable °°°þ°Pulse°Low°Port°1°pin...°°°°°°°°°°°rity      ³   04h   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄ°°°þ°Pulse°High°Port°1°pin...°°°°°°°°°ÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Normal°° þ Force Standby Mode          °°2 ÄÂÄ Security Mode Ä¿
³   Password º  úúúúú°°°þ°Set°Password°Security°°°°°°°°°°°   ³ Block Commands  ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Enable°°°þ°Toggle°RAM/ROM°Convert°Table°°°skÁÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Security 1 º        °°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ°°ú          ³   01h   ³
³ Security 2 º        °°°°°°Esc=Aborts°°Enter=Selects°°°°°ú          ³   01h   ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Key ÄÂ°ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼°ÄÄÄÄÄÄÄÄÄÄÄÄÅÄ Value Ä´
³ Ignore Key º  úúú  ³ D2h:Pswd test ³ Disabled ³  Send when Enabled ³   úúú   ³
³ Ignore Key º  úúú  ³ when P1.2=1   ÀÄÄÄÄÄÄÄÄÄÄÙ Send when Disabled ³   úúú   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Miscellaneous ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ Port Usage º Ports are not Swapped (Keyboard on Port0, Mouse on Port1)       ³
³ Clock Rate º Timer variables are based on a 12.0 MHz clock rate             ³
³ Conversion º Use ROM ScanCode Conversion Table                              ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ


         Controller identified as:  MultiKey/42i for the 8042 (v4.12)

ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
  1°°Help°°° 2°°Info°°° 3°°Color°° 4°°°°°°°°° 5°°Load°°° 6°°Save°°° Esc°Exit°°
```

Once the Task has been specified and the Task chosen requires Port Pins to be defined, the program prompts the user for the Port Pin number(s) with the Port Pin Dialog Box shown in Figure 4-8.

Figure 4-8. CFG42i.EXE Port Pin(s) Dialog.



```
(c)Copyright°Phoenix°Technologies°1996:°MultiKey/42i°Configuration°(Ver°1.4)°°°°
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Key Ä¿
³ Active Key º Ctrl  ³
³ Active Key º  Alt  ÃÄ HotKey Task ÄÄÄÄÄÄÄÄÄÂÄ SMI Ä¿
³   HotKey 1 º  F1   ³ Force Standby Mode    ³  00h  ³
³   HotKey 2 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 3 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 4 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 5 º   Q   ³ Enable Security       ³  03h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Port Mask ÄÄÄÂÄ External Pin Task ÄÄÄÂÄ SMI Ä¿
³ InputPin 1 º     úúú      ³           úúú          ³  00h  ³
³ InputPin 2 º     ú°ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»°h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄ°°°Select°Pin°or°Pin°Combination°°°ÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Timer°°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ°° Indicator ÄÄÄÄÄÄÄÄÄ¿
³ Inactivity º     ú°°°°°°°[°]°P1.7°xKeyLock°°°°°°°°°°°°°e Scroll Lock LED    ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Stand°°°°°°°[°]°P1.6°xCrtCGA°°°°°°°°°°°°°°k ÄÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Inactive 1 º Lower °°°°°°°[°]°P1.5°xManufacture°°°°°°°°° 00001000b ³   02h   ³
³ Inactive 2 º Enable°°°°°° [ ] P1.4 Unused        °°°°°°°rity       ³   04h   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄ°°°°°°°[°]°P1.3°Unused°°°°°°°°°°°°°°°ÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Norma°°°°°°°[°]°P1.2°Unused°°°°°°°°°°°°°°°2 ÄÂÄ Security Mode Ä¿
³   Password º  úúúú°°°°°°°[°]°P1.1°Mouse°Data°°°°°°°°°°°   ³ Block Commands  ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Enabl°°°°°°°[°]°P1.0°Keyboard°Data°°°°°°°°skÁÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Security 1 º      °°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ°°ú          ³   01h   ³
³ Security 2 º      °°°Space=Toggles°°Enter=Configures°°°ú          ³   01h   ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Key Ä°ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼°ÄÄÄÄÄÄÄÄÄÄÄÄÅÄ Value Ä´
³ Ignore Key º  úúú  ³ D2h:Pswd test ³ Disabled ³  Send when Enabled ³   úúú   ³
³ Ignore Key º  úúú  ³ when P1.2=1   ÀÄÄÄÄÄÄÄÄÄÄÙ Send when Disabled ³   úúú   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Miscellaneous ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ Port Usage º Ports are not Swapped (Keyboard on Port0, Mouse on Port1)        ³
³ Clock Rate º Timer variables are based on a 12.0 MHz clock rate               ³
³ Conversion º Use ROM ScanCode Conversion Table                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ


         Controller identified as:  MultiKey/42i for the 8042 (v4.12)

ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
  1°°Help°°° 2°°Info°°° 3°°Color°° 4°°°°°°°°° 5°°Load°°° 6°°Save°°° Esc°Exit°°
```

4.2.2 Configuring Input Pin Events and Tasks

When the External Input Pin Event & Task window is active (i.e. the window is highlighted and cursor is enabled), select between Input Pin 1 & 2 Port Mask with the ARROW keys and press the ENTER key to enable and modify Input Pin monitoring. Once the Port Mask has been selected the Port Pin Dialog Box of Figure 4-8 prompts the user to define the Pins used to monitor activity. The Pin Mask can be set to watch one or more Port1 Pins. If it is desired to perform two Tasks on one external Input Event, both Port Masks can be set to the same value.

Once the Keyboard Controller detects activity on a selected pin, it performs the corresponding Task. The Input Pin Event Task can be modified by selecting the Task which requires changing, with the ARROW keys and pressing the ENTER key. The program will prompt the user to select a Task from the Input Pin Event Task Dialog Box as shown from Figure 4-9. The list of Tasks available for Input Pin Events is very similar to the HotKey Task list shown in Figure 4-7 with a few additions (Restore Active State & Disable Security).

Once the Task has be specified and the Task chosen requires Port Pins to be defined, the program prompts the user for the Port Pin number(s) with the Port Pin Dialog Box shown in Figure 4-8.

Figure 4-9. CFG42i.EXE Input Pin Event Task Dialog.

```
°(c)Copyright°Phoenix°Technologies°1996:°MultiKey/42i°Configuration°(Ver°1.4)°°°
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Key Ä¿
³ Active Key º Ctrl  ³
³ Active Key º  Alt  ÃÄ HotKey Task ÄÄÄÄÄÄÄÄÄÂÄ SMI Ä¿
³   HotKey 1 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 2 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 3 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 4 º  úúú  ³          úúú         ³  00h  ³
³   HotKey 5 º   Q   ³ Enable Security      ³  03h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Port Mask ÄÄÄÂÄ External Pin Task ÄÄÄÂÄ SMI Ä¿
³ InputPin 1 º   00010°ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»°h  ³
³ InputPin 2 º      úú°°°°°Select°External°Input°Task°°°°°h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄ°°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ°°ÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Timer °°°þ°Clear°Task°°°°°°°°°°°°°°°°°°°°°° Indicator ÄÄÄÄÄÄÄÄÄ¿
³ Inactivity º      úú°°°þ°Lower°Port°1°pin...°°°°°°°°°°°°°e Scroll Lock LED    ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Standb°°°þ°Raise°Port°1°pin...°°°°°°°°°°°°°k ÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Inactive 1 º Lower P°°°þ°Pulse°Low°Port°1°pin...°°°°°°°°° 00001000b ³   02h   ³
³ Inactive 2 º Enable °°°þ°Pulse°High°Port°1°pin...°°°°°°°°rity      ³   04h   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄ°° þ Force Standby Mode          °°ÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Normal°°°þ°Restore°Active°State°°°°°°°°°°°°2 ÄÂÄ Security Mode Ä¿
³   Password º   úúúúú°°°þ°Set°Password°Security°°°°°°°°°°   ³ Block Commands  ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Enable°°°þ°Clear°Password°Security°°°°°°°°skÁÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Security 1 º        °°°þ°Toggle°RAM/ROM°Convert°Table°°°ú          ³   01h   ³
³ Security 2 º        °°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ°°ú          ³   01h   ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Key ÄÂ°°°°°°Esc=Aborts°°Enter=Selects°°°°°ÄÄÄÄÄÄÄÄÄÄÅÄ Value Ä´
³ Ignore Key º  úúú  ³°ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼°is Enabled ³   úúú   ³
³ Ignore Key º  úúú  ³ when P1.2=1   ÀÄÄÄÄÄÄÄÄÄÄÙ Send when Disabled ³   úúú   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Miscellaneous ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ Port Usage º Ports are not Swapped (Keyboard on Port0, Mouse on Port1)        ³
³ Clock Rate º Timer variables are based on a 12.0 MHz clock rate               ³
³ Conversion º Use ROM ScanCode Conversion Table                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ



         Controller identified as:  MultiKey/42i for the 8042 (v4.12)

ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
  1°°Help°°° 2°°Info°°° 3°°Color°° 4°°°°°°°°° 5°°Load°°° 6°°Save°°° Esc°Exit°°
```

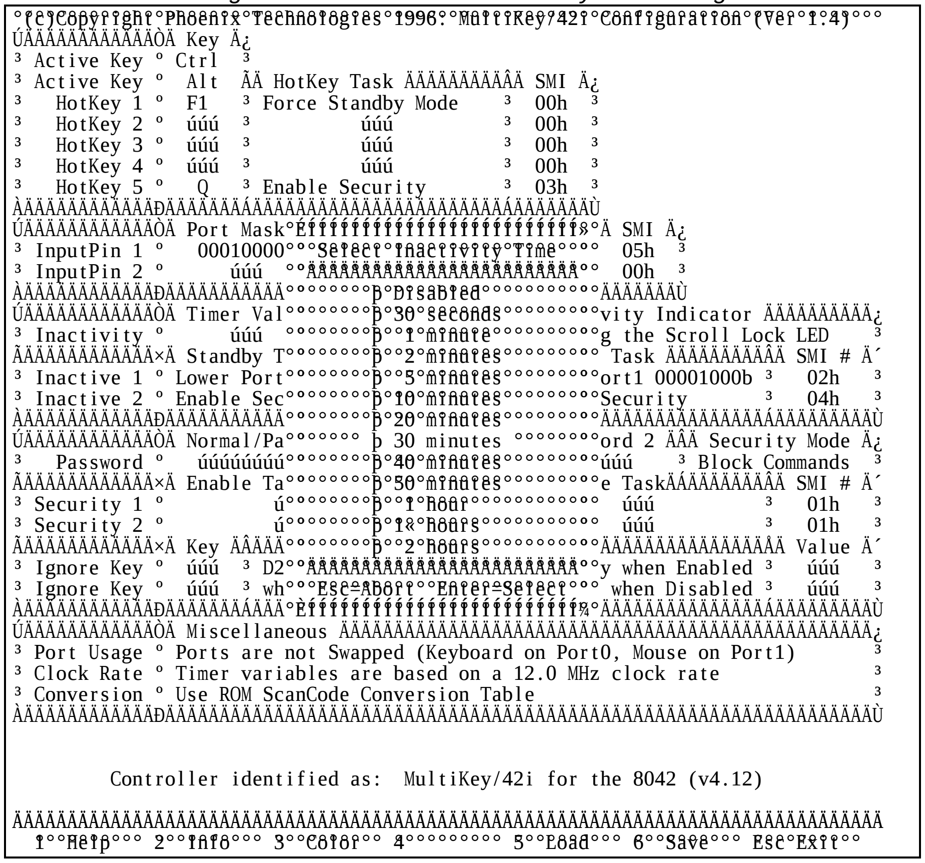
4.2.3 Configuring Inactivity Timer and Tasks

When the Inactivity Timer Configuration window is active (i.e. the window is highlighted and cursor is enabled), the user can choose to modify the Inactivity Time; Enable or Disable the Flashing Scroll Lock LED as an Inactivity Indicator; and set the Standby Tasks and the Standby SMI values. The Resume Tasks and Resume SMI values are automatically generated from the Standby values.

If the Inactivity Time is selected with the ENTER key, the program prompts the user for a Time value as shown in the Inactivity Time Dialog Box in Figure 4-10. Setting the Inactivity Time has been included in the MultiKey/42i Configuration Utility for completeness since the BIOS would probably set this value from user defined values stored in CMOS. This allows testing of the Keyboard Controller and the Inactivity Timer without any BIOS modifications.

As indicated in Section 4.2, the Inactivity Task was extended by sharing the HotKey 5 Task, the Second Task item enables/disables this feature. If selected the user is prompted by a simple Dialog Box whether to enable or disable this feature.

Figure 4-10. CFG42i.EXE Inactivity Time Dialog.



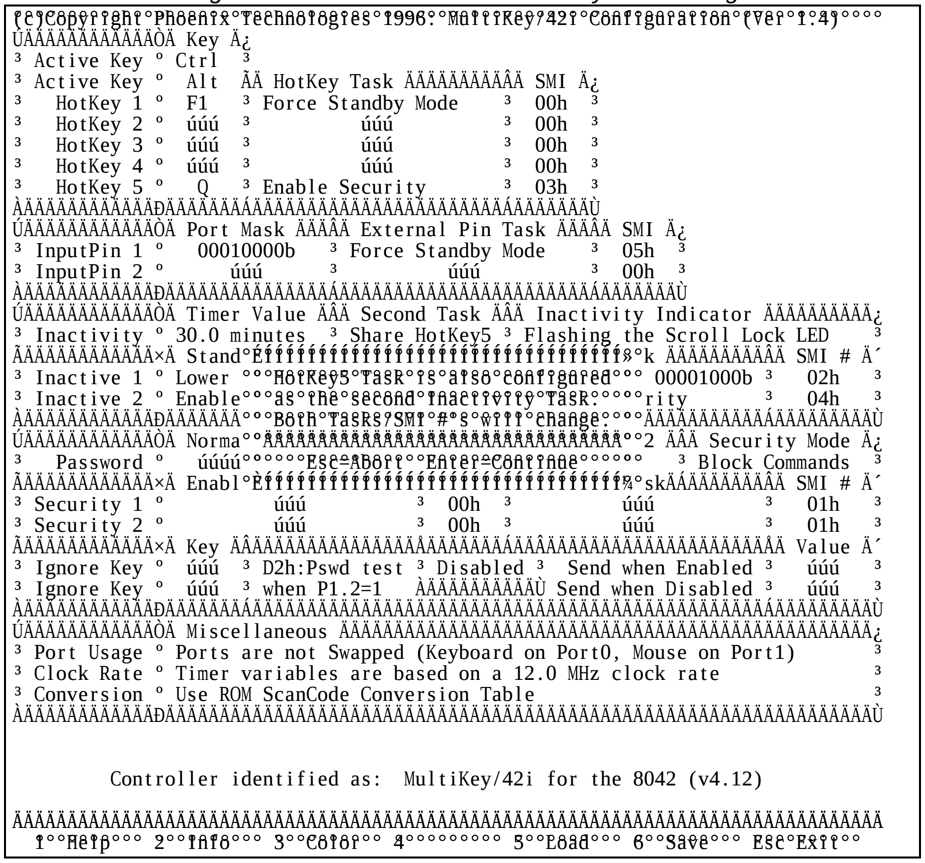
```
°(c)Copyright°Phoenix°Technologies°1996:°MultiKey/42i°Configuration°(Ver°1.4)°°°
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Key Ä¿
³ Active Key º Ctrl  ³
³ Active Key º  Alt  ÃÄ HotKey Task ÄÄÄÄÄÄÄÄÄÂÄ SMI Ä¿
³   HotKey 1 º  F1   ³ Force Standby Mode    ³  00h  ³
³   HotKey 2 º  úúú  ³         úúú           ³  00h  ³
³   HotKey 3 º  úúú  ³         úúú           ³  00h  ³
³   HotKey 4 º  úúú  ³         úúú           ³  00h  ³
³   HotKey 5 º   Q   ³ Enable Security       ³  03h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Port Mask°ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»°Ä SMI Ä¿
³ InputPin 1 º   00010000°°°Select°Inactivity°Time°°°°  05h  ³
³ InputPin 2 º      úúú  °°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ°°  00h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄ°°°°°°°°þ°Disabled°°°°°°°°°°°ÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Timer Val°°°°°°°°þ°30°seconds°°°°°°°°°vity Indicator ÄÄÄÄÄÄÄÄÄ¿
³ Inactivity º      úúú  °°°°°°°°þ°°1°minute°°°°°°°°°°g the Scroll Lock LED    ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Standby T°°°°°°°°þ°°2°minutes°°°°°°°°° Task ÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Inactive 1 º Lower Port°°°°°°°°þ°°5°minutes°°°°°°°°°ort1 00001000b ³   02h   ³
³ Inactive 2 º Enable Sec°°°°°°°°þ°10°minutes°°°°°°°°°Security       ³   04h   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄ°°°°°°°°þ°20°minutes°°°°°°°°°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Normal/Pa°°°°°°° þ 30 minutes °°°°°°°°ord 2 ÄÂÄ Security Mode Ä¿
³   Password º  úúúúúúúú°°°°°°°°þ°40°minutes°°°°°°°°°úúú    ³ Block Commands  ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Enable Ta°°°°°°°°þ°50°minutes°°°°°°°°°e TaskÄÁÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Security 1 º          ú°°°°°°°°þ°°1°hour°°°°°°°°°°°°°  úúú         ³   01h   ³
³ Security 2 º          ú°°°°°°°°þ°1«°hours°°°°°°°°°°°°  úúú         ³   01h   ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Key ÄÂÄÄÄ°°°°°°°°þ°°2°hours°°°°°°°°°°°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÅÄ Value Ä´
³ Ignore Key º  úúú  ³ D2°°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ°°y when Enabled ³   úúú   ³
³ Ignore Key º  úúú  ³ wh°°°Esc=Abort°°Enter=Select°°° when Disabled ³   úúú   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÁÄÄÄÄ°ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Miscellaneous ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ Port Usage º Ports are not Swapped (Keyboard on Port0, Mouse on Port1)       ³
³ Clock Rate º Timer variables are based on a 12.0 MHz clock rate              ³
³ Conversion º Use ROM ScanCode Conversion Table                               ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ


         Controller identified as:  MultiKey/42i for the 8042 (v4.12)

ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
  1°°Help°°° 2°°Info°°° 3°°Color°° 4°°°°°°°°° 5°°Load°°° 6°°Save°°° Esc°Exit°°
```

The Inactivity Indicator item is also an enable/disable feature. If selected, the user is prompted to enable/disable the Scroll Lock LED as a flashing Inactivity mode (Standby mode) indicator.

There are a maximum of two Standby Tasks to be performed. The second Task if defined is always performed before the first Task, this information is important when calculating unique SMI function numbers for all events. The Resume Tasks are automatically generated from the Standby Task. If the Task is a Pulsed function, the Task is exactly repeated for the Resume Task, otherwise the function is inverted for the Resume Task. If Port 1 Pin 3 was lowered by the Standby Task, it would be raised by the Resume Task. The Resume SMI number values are simply incremented from the Standby SMI numbers. Therefore the maximum Standby SMI number value is 00Eh, making the maximum Resume SMI number value equal to 00Fh.

The Standby Tasks are modified just as the HotKey Tasks with one addition; if the second Inactivity Timer Task is being modified, a Dialog Box reminding the user that this Task is connected with the HotKey 5 Task is displayed. If there is no second Standby Task, the Dialog Box will remind the user that this Task can only be used if the Second Task is set to Share HotKey5. If the HotKey 5 Task is being shared, the a Dialog Box will remind the user that modifying this Task will also modify the HotKey 5 Task as shown in Figure 4-11.

Figure 4-11. CFG42i.EXE 2nd Standby Task Dialog.



```
(c)Copyright°Phoenix°Technologies°1996:°MultiKey/42i°Configuration°(Ver°1.4)°°°°
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Key Ä¿
³ Active Key º Ctrl  ³
³ Active Key º  Alt  ÃÄ HotKey Task ÄÄÄÄÄÄÄÄÄÂÄ SMI Ä¿
³   HotKey 1 º  F1   ³ Force Standby Mode    ³  00h  ³
³   HotKey 2 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 3 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 4 º  úúú  ³          úúú          ³  00h  ³
³   HotKey 5 º   Q   ³ Enable Security       ³  03h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Port Mask ÄÄÄÂÄ External Pin Task ÄÄÄÂÄ SMI Ä¿
³ InputPin 1 º   00010000b   ³ Force Standby Mode    ³  05h  ³
³ InputPin 2 º      úúú      ³          úúú          ³  00h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Timer Value ÄÂÄ Second Task ÄÂÄ Inactivity Indicator ÄÄÄÄÄÄÄÄÄÄ¿
³ Inactivity º 30.0 minutes  ³ Share HotKey5 ³ Flashing the Scroll Lock LED    ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Stand°ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»°k ÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Inactive 1 º Lower °°°HotKey5°Task°is°also°configured°°° 00001000b ³   02h   ³
³ Inactive 2 º Enable°°°as°the°second°inactivity°Task.°°°°rity      ³   04h   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄ°°°Both°Tasks/SMI°#°s°will°change.°°°ÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Norma°°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ°°2 ÄÂÄ Security Mode Ä¿
³   Password º   úúúú°°°°°Esc=Abort°°Enter=Continue°°°°°°   ³ Block Commands  ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Enabl°ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼°skÁÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Security 1 º          úúú          ³  00h  ³          úúú          ³   01h   ³
³ Security 2 º          úúú          ³  00h  ³          úúú          ³   01h   ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Key ÄÂÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÅÄÄÄÄÄÄÄÁÄÄÂÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÅÄ Value Ä´
³ Ignore Key º  úúú  ³ D2h:Pswd test ³ Disabled ³  Send when Enabled ³   úúú   ³
³ Ignore Key º  úúú  ³ when P1.2=1   ÀÄÄÄÄÄÄÄÄÄÄÙ Send when Disabled ³   úúú   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Miscellaneous ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ Port Usage º Ports are not Swapped (Keyboard on Port0, Mouse on Port1)       ³
³ Clock Rate º Timer variables are based on a 12.0 MHz clock rate              ³
³ Conversion º Use ROM ScanCode Conversion Table                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ



        Controller identified as:  MultiKey/42i for the 8042 (v4.12)

ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
  1°°Help°°° 2°°Info°°° 3°°Color°° 4°°°°°°°°° 5°°Load°°° 6°°Save°°° Esc°Exit°°
```

Press the ENTER key to Continue defining the Task using the same procedures used to set the HotKey and External Input Pin Event Tasks (Sections 4.2.1 and 4.2.2).

4.2.4 Configuring Password Security and Tasks

Once the Password Configuration window is active (i.e. the window is highlighted and cursor is enabled), the user can choose to modify with the ARROW keys either of the Passwords, the Security Mode, the Security Enabled Tasks, the Security Enabled SMI values, the keys to not process as Password matches, the USB Password Validation mode, and the values to send to the System when Security is enabled & disabled. The Security Disabled Tasks and Security Disabled SMI values are automatically generated from the Security Enabled values.

Both 16-byte Passwords can be set from the MultiKey/42i Configuration Utility. This is an alternate method to having the BIOS set one or more user defined values that were stored in CMOS. It allows the testing of the Keyboard Controller and both Passwords without any BIOS modifications. Either Password can only be downloaded once and neither Password can ever be overwritten. After installation, either Password can disable Security. Passwords can employ separate Enable/Disable Tasks. These tasks can configure a 'customized system' accessing separate hardware and software components. Each Password has a separate Enable/Disable SMI function number which can be read (as soon as the Security is enabled) to determine which Password was entered through software, without causing an SMI. kbdWait4Security is an example of that type of routine; it waits for either Password and then reads the MultiKey/42i Keyboard Controller SMI value to determine which Password was entered.

The Security Mode item is an Enable/Disable feature. If selected the user is prompted by a simple Dialog Box whether to enable or disable the Blocking of Device Commands when the Security feature is enabled.

The Security Enabled Tasks are modified just as the HotKey, External Input Pin Event, and Inactivity Tasks. Defining the Task will follow the same procedure as setting the HotKey or External Input Pin Event Task (Sections 4.2.1 and 4.2.2). If both Passwords are installed, then both Security Enabled Tasks are performed when Security is enabled. The second Task, if defined, is always performed before the first Task; this information is important when calculating unique SMI function numbers for all events. The Security Disabled Tasks are automatically generated from the Security Enabled Task. If the Task is a Pulsed function, the Task is exactly repeated for the Security Disabled Task; otherwise the function is inverted for the Security Disabled Task. If Port 1 Pin 2 was lowered by the Security Enabled Task, it would be raised by the Security Disabled Task. The Security Disabled SMI number values are simply incremented from the Security Enabled SMI numbers. Therefore the maximum Security Enabled SMI number value is 00Eh, making the maximum Security Disabled SMI number value equal to 00Fh.

The USB Password Validation (D2h: Pswd test when P1.2 is High) item is an enable/disable feature. If selected the user is prompted by a simple Dialog Box whether to enable or disable the USB Password Validation support feature.

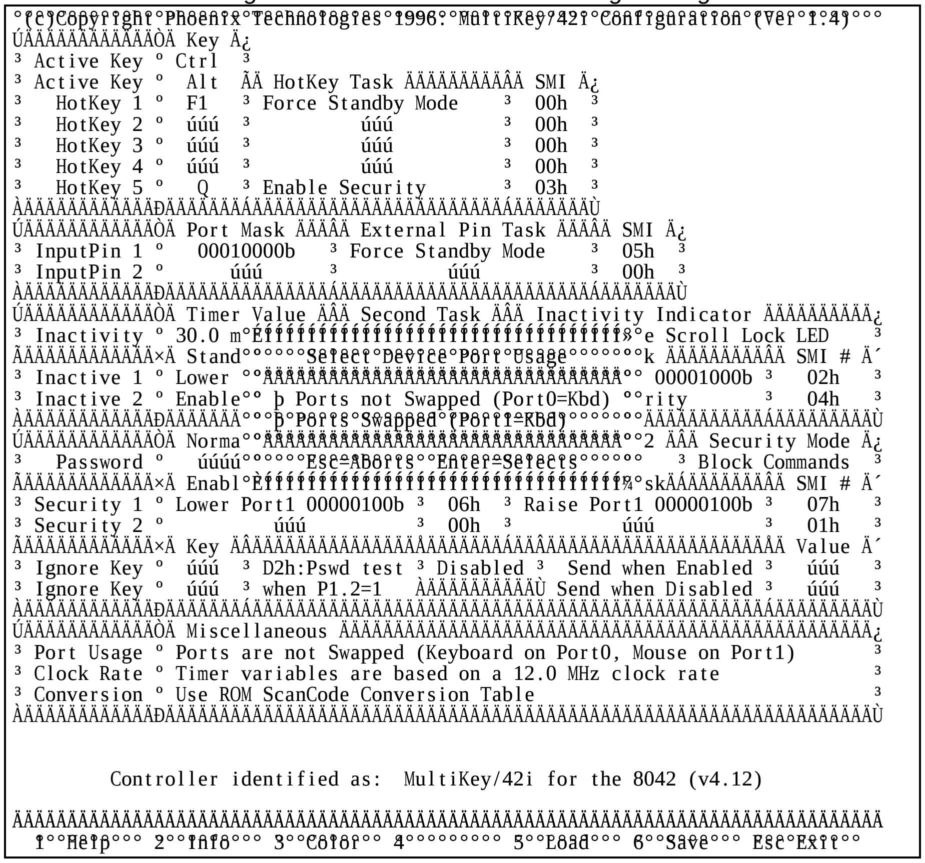
Setting the Ignore Key values and the data to be sent to the System on Enable/Disable have been included here since they affect the overall Security operation. These features are original IBM defined features. The Ignore Key values are normally set to left SHIFT key and right SHIFT key to remove all case-sensitivity with the Password. The Ignore Key values are modified exactly like setting the HotKey Scan Codes in Section 4.2.1.

The last remaining Security feature is the data sent to the System when Security is enabled and disabled. The idea behind this feature is to setup unique numbers that cannot be confused with the Keyboard Scan Codes so the System receives an indication when Security has been enabled and disabled, however this feature is rarely used since there are other ways of getting this information. Once the cursor has been moved to this feature, the numeric value can be typed-in pressing the ENTER key when completed or pressing the ENTER key first will pop-up a Dialog Box prompting the user for a numeric value. The value of zero disables the feature.

4.2.5 Configuring Miscellaneous Features

Once the Miscellaneous Configuration window is active (i.e. the window is highlighted and cursor is enabled), the user can choose to modify with the ARROW keys the Port Usage, the Clock Rate (which should be set at 12MHz), and whether the Scan Code Conversion table is taken from ROM or RAM.

Figure 4-12. CFG42i.EXE Port Usage Dialog.



```
°(c)Copyright°Phoenix°Technologies°1996:°MultiKey/42i°Configuration°(Ver°1.4)°°°
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Key Ä¿
³ Active Key º Ctrl  ³
³ Active Key º  Alt  ÃÄ HotKey Task ÄÄÄÄÄÄÄÄÄÂÄ SMI Ä¿
³   HotKey 1 º  F1   ³ Force Standby Mode    ³  00h  ³
³   HotKey 2 º  úúú  ³         úúú           ³  00h  ³
³   HotKey 3 º  úúú  ³         úúú           ³  00h  ³
³   HotKey 4 º  úúú  ³         úúú           ³  00h  ³
³   HotKey 5 º   Q   ³ Enable Security       ³  03h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Port Mask ÄÄÄÂÄ External Pin Task ÄÄÄÂÄ SMI Ä¿
³ InputPin 1 º   00010000b   ³ Force Standby Mode    ³  05h  ³
³ InputPin 2 º      úúú      ³         úúú           ³  00h  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Timer Value ÄÂÄ Second Task ÄÂÄ Inactivity Indicator ÄÄÄÄÄÄÄÄÄ¿
³ Inactivity º 30.0 m°ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»°e Scroll Lock LED    ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Stand°°°°°°Select°Device°Port°Usage°°°°°°°k ÄÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Inactive 1 º Lower °°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ°° 00001000b ³   02h   ³
³ Inactive 2 º Enable°° þ Ports not Swapped (Port0=Kbd) °°rity       ³   04h   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄ°°°þ°Ports°Swapped°(Port1=Kbd)°°°°°°°ÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Norma°°ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ°°2 ÄÂÄ Security Mode Ä¿
³   Password º   úúúú°°°°°°Esc=Aborts°°Enter=Selects°°°°°°   ³ Block Commands  ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Enabl°ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼°skÁÄÄÄÄÄÄÄÄÂÄ SMI # Ä´
³ Security 1 º Lower Port1 00000100b ³  06h  ³ Raise Port1 00000100b ³   07h   ³
³ Security 2 º          úúú          ³  00h  ³          úúú          ³   01h   ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ×Ä Key ÄÂÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÅÄÄÄÄÄÄÄÁÄÄÂÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÅÄ Value Ä´
³ Ignore Key º  úúú  ³ D2h:Pswd test ³ Disabled ³  Send when Enabled ³   úúú   ³
³ Ignore Key º  úúú  ³ when P1.2=1   ÀÄÄÄÄÄÄÄÄÄÄÙ Send when Disabled ³   úúú   ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÒÄ Miscellaneous ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ Port Usage º Ports are not Swapped (Keyboard on Port0, Mouse on Port1)        ³
³ Clock Rate º Timer variables are based on a 12.0 MHz clock rate               ³
³ Conversion º Use ROM ScanCode Conversion Table                                ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÐÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ


        Controller identified as:  MultiKey/42i for the 8042 (v4.12)

ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
  1°°Help°°° 2°°Info°°° 3°°Color°° 4°°°°°°°°° 5°°Load°°° 6°°Save°°° Esc°Exit°°
```

The Clock Rate has been included in the MultiKey/42i Configuration Utility for completeness and should always be set at 12MHz. One of the MultiKey/42i compromises which comes along with the additional features in a 2K package is that the Keyboard Controller’s Clock Rate must be 12MHz.

Both selecting Port Usage (PortSwapping) and selecting ROM/RAM Scan Code Conversion Table present a simple Dialog Box to the user when selected, as shown in Figure 4-12.

4.2.6 Exiting Configuration Utility

The user can exit the Configuration Utility by pressing the ESC key. If changes have been made and not saved, the Configuration Utility will prompt the user with a Dialog Box suggesting the current configuration needs to be saved. If the Configuration Utility is not running in Fake Hardware mode (Command Line switch: /F), the user is prompted with a Dialog Box asking if the current configuration is to be downloaded to the MultiKey/42i Keyboard Controller. The Passwords will also be downloaded, which means after this the MultiKey/42i configuration will not able to be changed until the System is powered down. If either Password has been downloaded before the Configuration Utility was run, the user will be given an Error Dialog Box indicating the MultiKey/42i configuration cannot be updated.

This page left blank.

Chapter 5
MultiKey Keyboard Controller Routines

Many MultiKey features are RAM loaded variables that can be changed or selected external to the Keyboard Controller. This Chapter is devoted to Code that talks to the Keyboard Controller, the Keyboard, and the PS/2 Mouse. This Code can be used in the System BIOS or in the Power Management routines.

5.1 Routines Overview

The following is a list of the types of Keyboard Controller routines contained in this Chapter.

- MultiKey Control Routines, Disabling/Enabling the Keyboard & Mouse Interface routines.
- MultiKey Support Routines, Keyboard Controller Interface routines.
- MultiKey/42i BIOS Routines, Keyboard Controller Configuration routines.
- MultiKey/42i Setup Routines, End-User Configuration routines.
- Keyboard/Mouse POST Routines, Reset/Init Keyboard Controller, Keyboard and the Mouse.

5.1.1 MultiKey Control Routines

The MultiKey Control routines provide clean access to the MultiKey Controller through foreground and background (i.e. SMI) environments. The type of Keyboard Controller and its configuration can also be determined from these routines. Table 5-1 lists the MultiKey Control Routines.

Table 5-1. MultiKey Control Routines.

Name	Description
kbdDevicesOff	Used in SMI code where no background Interrupt Service Routines are allowed to service the Mouse and Keyboard devices. This routine saves the Keyboard Controller’s current state and disables the Keyboard and Mouse device interfaces without disturbing incoming Keyboard and Mouse data.
kbdDevicesOn	Used in SMI code where no background Interrupt Service Routines are allowed to service the Mouse and Keyboard devices. This routine restores the original Keyboard Controller’s state and clears any spurious interrupts.
kbdDisDevices	Used in the foreground code where background Interrupt Service Routines handle the Mouse and Keyboard devices. This routine saves the Keyboard Controller’s current state and disables the Keyboard and Mouse device interfaces without disturbing incoming Keyboard and Mouse data.
kbdEnaDevices	Used in the foreground code where background Interrupt Service Routines handle the Mouse and Keyboard devices. This routine restores the original Keyboard Controller’s state and clears any spurious interrupts.
kbdChkProcessor	Determines the Type of Keyboard Controller present in the System along with the revision level of the processor and its current configuration.

5.1.2 MultiKey Support Routines

The MultiKey Support routines provide interface routine for MultiKey Keyboard Controller Variables and Memory along with the basic interface routines. Additional support includes System delay routines. Table 5-2 lists the MultiKey Support Routines.

Table 5-2. MultiKey Support Routines.

Name	Description
kbdWait4IBE	Waits for the Keyboard Controller's Input Buffer to be Empty.
kbdWait4OBF	Waits for the Keyboard Controller's Output Buffer to be Full and returns the Keyboard Controller's Data.
kbdStatusOBF	Waits for the Keyboard Controller's Output Buffer to be Full and returns the current Keyboard Controller's Status along with the Keyboard Controller's Data.
kbdWait4Quiet	Waits for the Keyboard Controller to be 'not busy'.
kbdDelay15us	Timed delay of 15 micro-seconds based on the Refresh Timer.
kbdDelay1ms	Timed delay of CX number milli-seconds based on the Refresh Timer.
kbdDelay1msOBF	Waits up to CX number of milli-seconds for the Keyboard Controller's Output Buffer to be Full and returns the Keyboard Controller's Data or Timeout Error.
kbdGetVariable	Reads MultiKey Keyboard Controller Variable through the extended MultiKey Command interface.
kbdSetVariable	Write MultiKey Keyboard Controller Variable through the extended MultiKey Command interface.
kbdGetProcRAM	Reads MultiKey Keyboard Controller Memory through the extended MultiKey Command interface
kbdSetProcRAM	Writes MultiKey Keyboard Controller Memory through the extended MultiKey Command interface.

5.1.3 MultiKey/42i BIOS Routines

The MultiKey/42i BIOS routines provide complete feature configuration for the MultiKey/42i product. These features include the Inactivity Timer, HotKeys, Input Pin Events, Security, and Miscellaneous Keyboard Controller functions. Table 5-3 lists the MultiKey/42i BIOS routines.

Table 5-3. MultiKey/42i BIOS Routines.

Name	Description
kbdCfgController	Configures all of the MultiKey/42i features from a table created from the MultiKey/42i Configuration Utility (CFG42i.EXE).
kbdPortSwapping	Configures which port has a Mouse attached and which port has a Keyboard attached.
kbdCalibrateTmrs	Calibrates the MultiKey/42i internal clock variables to allow the Inactivity Timer to be accurate.

5.1.4 MultiKey/42i Setup Routines

The MultiKey/42i Setup routines provide the End-User support routines for some of the MultiKey/42i run-time features. Table 5-4 lists the MultiKey/42i Setup Routines.

Table 5-4. MultiKey/42i Setup Routines.

Name	Description
kbdWait4Security	Waits for the User/Supervisor to enter a Password which will disable Security and then inquires of MultiKey/42i which Password was entered.
kbdSetInactiveTmr	Updates and starts the Inactivity Timer. The value of zero disables the Inactivity Timer.
kbdGetInactiveTmr	Reads the current Inactivity Timer value. The value of zero indicates the Inactivity Timer is disabled.
kbdLoadPassword	Loads specified (User/Supervisor) Password from a table.

5.1.5 Keyboard/Mouse POST Routines

The Keyboard/Mouse POST routines are used to initialize the external Devices from power-on. These routines can be used in POST (Power-On Self Test) or Resume to test and initialize the Devices before programming them to the desired state. Table 5-5 lists the MultiKey Keyboard/Mouse POST routines.

Table 5-5. MultiKey Keyboard and Mouse POST Routines.

Name	Description
kbdStopDevices	Disables the Keyboard from scanning the Matrix of keys and disables the Mouse from creating Mouse packets.
kbdFlushDevices	Checks and flushes Data from the Keyboard Controller and Devices.
kbdRstController	Resets the Keyboard Controller using the “Self Test” Command.
kbdRstKeyboard	Resets the Keyboard and leave the device disabled.
kbdRstPS2Mouse	Resets the PS/2 Mouse and leave the device disabled.
kbdSend2Keyboard	Sends Command/Data to the Keyboard Device and waits for an acknowledgment.
kbdSend2Mouse	Sends Command/Data to the PS/2 Mouse Device and waits for an acknowledgment.

5.1.6 Sample Keyboard Controller Code

Figure 5-1 illustrates a sample of the MultiKey code base.

Figure 5-1. Sample Keyboard Controller Code. (sheet 1 of 24)

```
;---------------------------------------------------------------------------
;
;       Copyright (c) 1992-1996 Phoenix Technologies Ltd.
;       This program contains proprietary and confidential information. All
;       rights reserved except as may be permitted by prior written consent.
;
;       Content: Common Keyboard Controller, Keyboard, & PS/2 Mice support
;                routines.
;
;---------------------------------------------------------------------------
; Local Equates - Used in this Module
;---------------------------------------------------------------------------

BITF            EQU     1000000000000000b
BITE            EQU     0100000000000000b
BITD            EQU     0010000000000000b
BITC            EQU     0001000000000000b
BITB            EQU     0000100000000000b
BITA            EQU     0000010000000000b
BIT9            EQU     0000001000000000b
BIT8            EQU     0000000100000000b
BIT7            EQU     0000000010000000b
BIT6            EQU     0000000001000000b
BIT5            EQU     0000000000100000b
BIT4            EQU     0000000000010000b
BIT3            EQU     0000000000001000b
BIT2            EQU     0000000000000100b
BIT1            EQU     0000000000000010b
BIT0            EQU     0000000000000001b

;---------------------------------------------------------------------------

        DATA    SEGMENT public 'DATA'
        ASSUME  DS:DATA

;---------------------------------------------------------------------------
; Data Segment Variables - MultiKey Variable Definitions
;---------------------------------------------------------------------------
;
;{Read 64h} 8042 Status              {saveKCCB} KB Controller Command Byte
; B7 - Parity Error                   B7 - Reserved
; B6 - Timeout (AT=Rcv Timeout)       B6 - Cnvt ScanCodes
; B5 - Aux OBF (AT=Xmt Timeout)       B5 - Aux Disabled (AT=PC Mode)
; B4 - KeyLock switch inactive        B4 - Kbd Disabled
; B3 - Command/Data                   B3 - Reserved (AT=Override switch)
; B2 - System Flag                    B2 - System Flag
; B1 - IBF                            B1 - Aux IntrEnabled
; B0 - OBF                            B0 - Kbd IntrEnabled
;
;{verInfo1} Version Information I    {verInfo2} Version Information II
; B7 - Processor Type (bit2)          B7 - IRQ12 software Flip/Flop
; B6 - Battery Management Support     B6 - IRQ12 software inverted
; B5 - Kbd Scanning support           B5 - IRQ1 software Flip/Flop
; B4 - Power Down Support             B4 - IRQ1 software inverted
; B3 - Processor Type (bit1)          B3 - Clock speed (bit 3)
; B2 - PS/2 Mouse Emulation           B2 - Clock speed (bit 2)
; B1 - AT Environment (0=PS/2)        B1 - Clock speed (bit 1)
; B0 - Processor Type (bit0)          B0 - Clock speed (bit 0)
;
; Processor Type: bit2 bit1 bit0     {verFlags} Version Flags
;         M38802   0    0    0        B7 - AMI Keyboard Controller
;        80C51SL   0    0    1        B6 - Phoenix Keyboard Controller
;          80x86   0    1    0        B5 - MultiKey Keyboard Controller
;        H8/3332   0    1    1        B4 - PS/2 Mouse Environment
;          V144L   1    0    0        B3 - PS/2 Mouse Wrap Mode Set
;           8042   1    0    1        B2 - PS/2 Mouse Attached
;       Reserved   1    1    0        B1 - Keyboard Attached
;       Reserved   1    1    1        B0 - Reserved
;---------------------------------------------------------------------------
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 2 of 24)

```
verFlags        DB      0               ; Processor Version information
verInfo1        DB      0               ; Phoenix Version Information byte I
verInfo2        DB      0               ; Phoenix Version Information byte II
revision        DW      0               ; Phoenix PVCS Revision number
memoryIdx       DB      0               ; Phoenix extended memory index
saveKCCB        DB      0               ; Storage for the KCCB
saveData        DB      0               ; Storage for the Data (@ Port 60h)
saveStatus      DB      0               ; Storage for the Status (Port 64h)
dummyVector     DD      0               ; Storage for Kbd Interrupt Vector

;--------------------------------------------------------------------------

        DATA    ENDS

;--------------------------------------------------------------------------

        CODE    SEGMENT public 'CODE'
        ASSUME  CS:CODE

;==========================================================================
;              M U L T I K E Y   C O N T R O L   R O U T I N E S
;==========================================================================

;+-------------------------------------------------------------------------
;
; kbdDevicesOff - Saves the current Keyboard Controller Command Byte and
;                 disables the Keyboard and Auxiliary Devices.  This routine
;                 to be used in a non-interruptable environment (i.e. SMI
;                 type handler routine).
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: saveKCCB, saveData, saveStatus, and the Minor Flags.
;
;         Note: Problems covered by kbdDevicesOff & kbdDevicesOn:
;               1) kbdDevicesOff executed while Kbd/Aux data coming in. The
;                  Data will be saved until the kbdDevicesOn Routine, before
;                  Devices are turned off with no gaps between Commands that
;                  would allow other Device input.
;               2) The Data will be correctly identified as Kbd or Aux Data
;                  even if is an AT Type Keyboard Controller (no PS/2 Mouse
;                  support), since the AuxOBF line in the Status port is
;                  defined as Transmit Error not Receive Error on the AT Type
;                  Keyboard Controller.
;               3) The Kbd interface is disabled and if the AuxDevice exists
;                  it is also disabled.  kbdDevicesOn will restore the
;                  original state.
;               4) In the kbdDevicesOn Routine the Output Buffer is flushed
;                  to fix if port 60h was read too quickly on Systems that
;                  raise IRQ1 & IRQ12 in software rather than hardware, which
;                  would cause a spurious interrupt.
;
;         Note: Problems not covered by kbdDevicesOff & kbdDevicesOn:
;               1) The "sticky" PICs problem -- "sticky" PICs are System were
;                  reading port 60h with interrupts disabled will still cause
;                  a Keyboard Interrupt (spurious interrupt).
;               2) These Routines will not work with non-MultiKey Controllers
;                  configured as an AT type Keyboard Controller (all PS/2
;                  Types OK). All MultiKey products will work in both AT &
;                  PS/2 modes.
;
;   Processing: Wait for the Controller to finish up working on the Keyboard
;               Mouse Transmission in progress and save the Data if present,
;               then disable both Devices' interfaces.  This routine is used
;               in Higher priority interrupts (i.e. NMI, SMI, & Timer) than
;               the Keyboard Interrupt.
;
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 3 of 24)

```
kbdDevicesOff  PROC  NEAR  PUBLIC
        push    ax
        mov     al, 07Dh                        ; Write RAM Location 3Dh
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al                        ; Issue Cmd of 2-byte Cmd only
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        in      al, 064h                        ; Get the Status
        mov     BYTE PTR ds:[saveStatus], al
        test    BYTE PTR ds:[saveStatus], BIT0  ; Is Output Buffer Full?
        jz      off1                            ; Jmp if no
        in      al, 060h                        ; Get the Controller Data
        mov     BYTE PTR ds:[saveData], al
off1:   mov     al, 020h                        ; Read KCCB command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, 0ADh                        ; Disable Keyboard interface
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, 0A7h                        ; Disable AuxDevice interface
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, 0FFh                        ; Null Command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al                            ; Make last Command complete
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
        mov     BYTE PTR ds:[saveKCCB], al      ; Save so it can be restored
        pop     ax
        ret
kbdDevicesOff  ENDP

;+-----------------------------------------------------------------------
;
; kbdDevicesOn - Restores the original Keyboard Controller Command Byte.
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: Minor Flags.
;
;   Processing: Puts the Keyboard/AuxDevice transmission Data back in the
;               Output Buffer and restores the original state of the Keyboard
;               and AuxDevice interfaces, from before the kbdDevicesOff
;               Routine if and only if kbdDevicesOff was originally Called.
;
kbdDevicesOn  PROC  NEAR  PUBLIC
        push    ax
        mov     al, 0FFh                        ; Null Command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        call    kbdWait4IBE                     ; Wait for last intr=complete
        in      al, 060h                        ; Flush Output Buffer
        test    BYTE PTR ds:[saveStatus], BIT0  ; Was Output Buffer Full?
        jz      on2                             ; Jmp if no
        mov     al, 0D2h                        ; Echo Keyboard ScanCode
        test    BYTE PTR ds:[saveStatus], BIT5  ; Was it an AuxDevice OBF?
        jz      on1                             ; Jmp if no
        mov     al, 0D3h                        ; Echo AuxDevice Data
on1:    call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, BYTE PTR ds:[saveData]      ; Get Controller Data
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
on2:    mov     al, 060h                        ; Write KCCB command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, BYTE PTR ds:[saveKCCB]      ; Get original KCCB
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
        pop     ax
        ret
kbdDevicesOn  ENDP
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 4 of 24)

```
;+-----------------------------------------------------------------------
;
; kbdDisDevices - Saves the current Keyboard Controller Command Byte and
;                disables the Keyboard and Auxiliary Devices.  This routine
;                to be used in an interruptable environment.
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: saveKCCB, dummyVector, and Minor Flags.
;
;         Note: Problems covered by kbdDisDevices & kbdEnaDevices:
;               1) kbdDisDevices executed while Kbd/Aux data coming in. The
;                  Data will go to the original Interrupt routine before
;                  Devices are turned off with no gaps between Commands that
;                  would allow other Device input.
;               2) The Kbd interface is disabled and if the AuxDevice exists
;                  it is also disabled.  kbdEnaDevices will restore the
;                  original state.
;               3) The Dummy_Interrupt allows "sticky" PICs to work, "sticky"
;                  PICs are System were reading port 60h with interrupts
;                  disabled will still cause a Keyboard Interrupt (spurious
;                  interrupt).
;               4) The Dummy_Interrupt fixes reading port 60h too quickly on
;                  Systems that raise IRQ1 & IRQ12 in software rather than
;                  hardware, which would also cause a spurious interrupt.
;               5) The Specific EOI in kbdEnaDevices allows level sensitive
;                  PICs (i.e. PS/2 Systems) to work, since the dummyInterrupt
;                  does not read Port 60h, and therefore would cause
;                  contineous interrupts.
;
;   Processing: After waiting for the Controller to finish up working on
;               the Keyboard/Mouse Transmission in progress, turn Off
;               both Devices and Disable Keyboard Controller Interrupts.
;               This routine is used in the Main Line code where
;               Interrupt Routines are handling both the Kbd/Aux Devices.
;
kbdDisDevices  PROC  NEAR  PUBLIC
        push    ax
        mov     al, 07Dh                        ; Write RAM Location 3Dh
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al                        ; Issue Cmd of 2-byte Cmd only
        call    kbdWait4Quiet                   ; Wait for IBE & OBE
        call    installInterrupt                ; Kbd intr => Dummy_Interrupt
        mov     al, 020h                        ; Read KCCB command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, 060h                        ; Write KCCB command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, 074h                        ; Aux/Kbd interface & IRQs off
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
        mov     BYTE PTR ds:[saveKCCB], al      ; Save so it can be restored
        pop     ax
        ret
kbdDisDevices  ENDP
;+-----------------------------------------------------------------------
;
; kbdEnaDevices - Restores the Keyboard Controller Command Byte.
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: Minor Flags.
;
;   Processing: Restore the original state of the Keyboard and Mouse
;               interfaces, and clear the pending spurious interrupt if
;               present, from the kbdDisDevices Routine.
;
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 5 of 24)

```
kbdEnaDevices  PROC  NEAR  PUBLIC
        push    ax
        mov     al, 060h                        ; Write KCCB command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, BYTE PTR ds:[saveKCCB]      ; Get original KCCB
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
        call    restoreInterrupt                ; Restore Keyboard interrupt
        pop     ax
        ret
kbdEnaDevices  ENDP

;-----------------------------------------------------------------------------
;
; installInterrupt - Installs the Dummy Interrupt Vector.
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: Keyboard Interrupt Vector and dummyVector.
;
;   Processing: Replace original Keyboard Interrupt vector with Dummy
;               Keyboard Interrupt Routine.
;
installInterrupt  PROC  NEAR
        pushf
        push    es
        push    ax
        cli
        mov     ax, 0
        mov     es, ax                          ; ES => Segment zero
        mov     ax, WORD PTR es:[9*4+0]         ; Get original Offset
        mov     WORD PTR ds:[dummyVector+0], ax ; Save original Offset
        mov     WORD PTR es:[9*4+0], OFFSET cs:dummyInterrupt
        mov     ax, WORD PTR es:[9*4+2]         ; Get original Segment
        mov     WORD PTR ds:[dummyVector+2], ax ; Save original Segment
        mov     WORD PTR es:[9*4+2], cs         ; Install new Vector
        pop     ax
        pop     es
        popf
        ret
installInterrupt  ENDP

;-----------------------------------------------------------------------------
;
; restoreInterrupt - Restore original Keyboard Interrupt routine vector.
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: Keyboard Interrupt Vector and dummyVector.
;
;   Processing: Restore original Keyboard Interrupt vector saved from the
;               installInterrupt routine.
;
restoreInterrupt  PROC  NEAR
        pushf
        push    es
        push    ax
        cli
        mov     ax, 0
        mov     es, ax                          ; ES => Segment zero
        mov     ax, WORD PTR ds:[dummyVector+0] ; Get original Offset
        mov     WORD PTR es:[9*4+0], ax
        mov     ax, WORD PTR ds:[dummyVector+2] ; Get original Segment
        mov     WORD PTR es:[9*4+2], ax
        mov     al, 061h
        out     020h, al                        ; Specific EOI to the PIC1
        pop     ax
        pop     es
        popf
        ret
restoreInterrupt  ENDP
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 6 of 24)

```
;---------------------------------------------------------------------------
;
; dummyInterrupt - Prevent spurious Keyboard Interrupts.
;
;       Entry: None.
;
;        Exit: None.
;
;    Modifies: None.
;
;  Processing: Several chipsets have problems and create spurious Keyboard
;              Interrupts even when interrupts are disabled.
;
dummyInterrupt  PROC  NEAR
        iret
dummyInterrupt  ENDP

;+--------------------------------------------------------------------------
;
; kbdChkProcessor - Determine the Type of Keyboard Controller present in the
;                   System.
;
;       Entry: DS = DATA segment.
;
;        Exit: None.
;
;    Modifies: revision, verInfo1, verInfo2, verFlags, and the Minor Flags.
;
;        Note: verInfo1 ANDed with 11111101b produces...
;                       00010000b = MultiKey/3880
;                       00110000b = MultiKey/3880L
;                       00110001b = MultiKey/51L
;                       00110101b = MultiKey/51LM
;                       00111001b = MultiKey/H8L
;                       00111101b = MultiKey/H8LM
;                       01111001b = MultiKey/H8LB
;                       01111101b = MultiKey/H8LMB
;                       10110000b = MultiKey/144L
;                       10000001b = MultiKey/42
;                       10000001b = MultiKey/42i Revision > 4.00
;                       10000001b = MultiKey/42E Revision > 3.00
;                       10010001b = MultiKey/42G
;                       10110001b = MultiKey/42L
;  Processing: The first step is to check for an AMI Keyboard Controller.
;              The next step is to determine if and what type of Phoenix
;              Keyboard Controller is present and finally check if there
;              is Auxiliary Device Support.
;
kbdChkProcessor  PROC  NEAR  PUBLIC
        push    ax
        mov     BYTE PTR ds:[verInfo1], 000h    ; Clear all Version flags
        mov     BYTE PTR ds:[verInfo2], 000h
        mov     BYTE PTR ds:[verFlags], 000h
        call    kbdDisDevices                   ; Disable Device interfaces
        call    chkKnown8042                    ; Is there a Known Controller?
        call    chkPhoenixKBC                   ; Is it a Phoenix Controller?
        call    chkMultiKeyKBC                  ; Is it a MultiKey Controller?
        call    chkAuxDevice                    ; Is there AuxDevice support?
        call    kbdEnaDevices                   ; Restore Device interfaces
        pop     ax
        ret
kbdChkProcessor  ENDP
;---------------------------------------------------------------------------
;
; chkKnown8042 - Check if is an AMI Keyboard Controller product.
;
;       Entry: DS = DATA segment.
;
;        Exit: None.
;
;    Modifies: revision, verFlags, and Minor Flags.
;
;  Processing: Issue a special AMI Command (A1h) and see if it generates a
;              response, if so issue another special AMI Command (CAh) to
;              see if AuxDevice is supported or not.
;
```

Figure 5-1.   Sample Keyboard Controller Code. (sheet 7 of 24)

```
chkKnown8042  PROC  NEAR
        push    cx
        push    ax
        in      al, 060h                        ; Flush Output Buffer
        mov     al, 0A1h                        ; Output Controller Version
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     cx, 6                           ; Wait up to 6ms for OBF
        call    kbdDelay1msOBF                  ; Wait CX * 1ms for OBF
        jc      know1                           ; AMI 8042 Cmds? (jmp if no)
        mov     ah, 000h
        mov     WORD PTR ds:[revision],ax       ; Save verison info
        mov     al, 0CAh                        ; Read Controller Mode
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     cx, 6                           ; Wait up to 6ms for OBF
        call    kbdDelay1msOBF                  ; Wait CX * 1ms for OBF
        jc      know1                           ; AMI 8042 Cmds? (jmp if no)
        or      BYTE PTR ds:[verFlags], BIT7    ; Set AMI Keyboard Product
        test    al, BIT0                        ; Is it a PS/2 environment?
        jz      know1                           ; Jmp if no
        or      BYTE PTR ds:[verFlags], BIT4    ; Set AuxDevice Support
know1:  pop     ax
        pop     cx
        ret
chkKnown8042  ENDP

;----------------------------------------------------------------------------
;
; chkPhoenixKBC - Check if is a Phoenix Keyboard Controller product.
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: verFlags and Minor Flags.
;
;   Processing: Issue a special Phoenix Command, which is also a MultiKey
;               Command, (BAh) and see if it generates a response.
;
chkPhoenixKBC  PROC  NEAR
        push    cx
        push    ax
        test    BYTE PTR ds:[verFlags], BIT7    ; Is it an AMI Processor?
        jnz     cpkbc1                          ; Jmp if yes
        in      al, 060h                        ; Flush Output Buffer
        mov     al, 0BAh                        ; Read RAM @Index
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     cx, 6                           ; Wait up to 6ms for OBF
        call    kbdDelay1msOBF                  ; Wait CX * 1ms for OBF
        jc      cpkbc1                          ; A Phoenix KBC? (jmp if no)
        or      BYTE PTR ds:[verFlags], BIT6    ; Set a Phoenix Product
cpkbc1: pop     ax
        pop     cx
        ret
chkPhoenixKBC  ENDP
;----------------------------------------------------------------------------
;
; chkMultiKeyKBC - Check if is a MultiKey Keyboard Controller product.
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: revision, verFlags, verInfo1, verInfo2, and Minor Flags.
;
;   Processing: Issue special MultiKey Commands (B8h & B9h) and see if it
;               generates a response, if so issue other MultiKey Commands
;               to get the Revision level and the Version Information.
;
```

Figure 5-1.   Sample Keyboard Controller Code. (sheet 8 of 24)

```
chkMultiKeyKBC  PROC  NEAR
        push    cx
        push    ax
        test    BYTE PTR ds:[verFlags], BIT6    ; Is it a Phoenix Processor?
        jz      cmkbc1                          ; Jmp if no
        in      al, 060h                        ; Flush Output Buffer
        mov     al, 0B9h                        ; Read RAM Index.
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     cx, 6                           ; Wait up to 6ms for OBF
        call    kbdDelay1msOBF                  ; Wait CX * 1ms for OBF
        jc      cmkbc1                          ; Jmp if not MultiKey
        mov     BYTE PTR ds:[memoryIdx], al     ; Save Memory Index for Resume
        mov     al, 0D5h                        ; Read Phoenix PVCS revision
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
        mov     ah, al
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
        mov     WORD PTR ds:[revision], ax      ; Save the PVCS revision #
        mov     al, 0D6h                        ; Read Version Info bytes
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
        mov     BYTE PTR ds:[verInfo1], al      ; Processor type, features...
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
        mov     BYTE PTR ds:[verInfo2], al      ; Speed & IRQ1/IRQ12 line type
        or      BYTE PTR ds:[verFlags], BIT5    ; Set MultiKey Kbd Controller
cmkbc1: pop     ax
        pop     cx
        ret
chkMultiKeyKBC  ENDP

;---------------------------------------------------------------------------
;
; chkAuxDevice - Check if this environment supports a PS/2 Mouse.
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: verFlags and Minor Flags.
;
;   Processing: Issue Test AuxDevice Interface Command (A9h) and see if it
;               generates a response.  This Command generates a response
;               whether or not the Mouse is plugged in.  However, the AMI
;               Keyboard Controller not in the PS/2 environment also responds
;               (incorrectly) to this Command, that's why AMI Keyboard
;               Controller is explictly check for.
;
chkAuxDevice  PROC  NEAR
        push    cx
        push    ax
        test    BYTE PTR ds:[verFlags], BIT7    ; Is it an AMI Processor?
        jnz     cad1                            ; Jmp if no
        in      al, 060h                        ; Flush Output Buffer
        mov     al, 0A9h                        ; Test AuxDevice Interface
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     cx, 6                           ; Wait up to 6ms for OBF
        call    kbdDelay1msOBF                  ; Wait CX * 1ms for OBF
        jc      cad1                            ; AuxDevice Cmds? (jmp if no)
        or      BYTE PTR ds:[verFlags], BIT4    ; Set AuxDevice Support
cad1:   pop     ax
        pop     cx
        ret
chkAuxDevice  ENDP
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 9 of 24)

```
;==========================================================================
;             M U L T I K E Y   S U P P O R T   R O U T I N E S
;==========================================================================

;+-------------------------------------------------------------------------
;
; kbdWait4IBE - Waits for the Keyboard Controller Input Buffer to be Empty.
;
;        Entry: None.
;
;         Exit: None.
;
;     Modifies: Minor Flags.
;
;   Processing: Polls Port 64h waiting for Input Buffer to be Empty (IBE).
;
kbdWait4IBE  PROC  NEAR  PUBLIC
        push    ax
ibe1:   in      al, 064h                        ; Read 8042 status
        test    al, BIT1                        ; Is Input Buffer Empty?
        jnz     ibe1                            ; Jmp if no
        pop     ax
        ret
kbdWait4IBE  ENDP

;+-------------------------------------------------------------------------
;
; kbdWait4OBF - Waits for the Keyboard Controller Output Buffer to be full.
;
;        Entry: None.
;
;         Exit: AL = Keyboard Controller Data.
;
;     Modifies: AL and Minor Flags.
;
;   Processing: Polls Port 64h waiting for Output Buffer to be Full (OBF),
;               then reads Port 60h.
;
kbdWait4OBF  PROC  NEAR  PUBLIC
        push    ax
obf1:   in      al, 064h                        ; Read 8042 status
        test    al, BIT0                        ; Is Output Buffer Full?
        jz      obf1                            ; Jmp if no
        pop     ax
        in      al, 060h                        ; Read Output Buffer data
        ret
kbdWait4OBF  ENDP
;+-------------------------------------------------------------------------
;
; kbdStatusOBF - Waits for the Output Buffer to be full and returns Status.
;
;        Entry: None.
;
;         Exit: AL = Keyboard Controller Data.
;               AH = Keyboard Controller Status.
;
;     Modifies: AX and Minor Flags.
;
;   Processing: Polls Port 64h waiting for Output Buffer to be Full (OBF),
;               then reads Port 60h.
;
kbdStatusOBF  PROC  NEAR  PUBLIC
        in      al, 064h                        ; Read 8042 status
        test    al, BIT0                        ; Is Output Buffer Full?
        jz      kbdStatusOBF                    ; Jmp if no
        mov     ah, al                          ; Save the Status
        in      al, 060h                        ; Read output buffer
        ret
kbdStatusOBF  ENDP
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 10 of 24)

```
;+-----------------------------------------------------------------------
;
; kbdWait4Quiet - Waits for the Keyboard Controller to be not busy.
;
;       Entry: None.
;
;        Exit: None.
;
;     Modifies: AL and Minor Flags.
;
;   Processing: Polls Port 64h waiting for Output Buffer and Input Buffer to
;               be empty.
;
kbdWait4Quiet  PROC  NEAR  PUBLIC
        push    ax
idle:   in      al, 064h                        ; Read 8042 Status
        and     al, BIT1+BIT0                   ; Are both Buffers Empty?
        jnz     idle                            ; Jmp if no
        pop     ax
        ret
kbdWait4Quiet  ENDP

;------------------------------------------------------------------------
;
; kbdDelay15us - Timed Delay based on Refresh Timer.
;
;       Entry: CX = Number of 15us to Delay.
;
;        Exit: None.
;
;     Modifies: CX and the Minor Flags.
;
;   Processing: The accuracy of this delay routine is zero to plus 15us,
;               since the timing loop must synchronize on the refresh status
;               signal (the output of Timer 1 through a flip-flop), which is
;               a symmetric square wave with a cycle of 30us.  That's 15us
;               per transition.  The execution of the entry/exit code,
;               including the jumps, must also be added into the total delay
;               time.  Since both entry and exit sources of error always
;               total less than 20us maximum, for a loop time of 120us, the
;               loop variance is at most 16.7%.
;
;  Assumptions: System BIOS is shadowed and Timer1 is for standard 15us
;               refresh (Mode 2, count 012h).
;
kbdDelay15us  PROC  NEAR
        push    ax
        mov     ah, not BIT4                    ; Force miscompare, preload
dly1:   out     0EDh, al                        ; System I/O BUS Delay
        in      al, 061h                        ; Get current Port B Status
        and     al, BIT4                        ; Isolate refresh status
        cmp     al, ah                          ; Does it match last Status?
        je      dly1                            ; Jmp if yes
        mov     ah, al                          ; Save new refresh status
        loop    dly1                            ; Repeat until timer expires
        pop     ax
        ret
kbdDelay15us  ENDP

;+-----------------------------------------------------------------------
;
; kbdDelay1ms - Timed Delay based on Refresh Timer.
;
;       Entry: CX = Number of milli-seconds to Delay.
;
;        Exit: None.
;
;     Modifies: CX and the Minor Flags.
;
;   Processing: This delay loop provides milli-seconds based timing from the
;               BIOS's 15us refresh rate timer.
;
;        Note: The "kbdDelay15us" was measured to be 25us delay.
;
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 11 of 24)

```
kbdDelay1ms  PROC  NEAR  PUBLIC
        push    cx
;       mov     cx, 67                          ; Number of 15us in a 1ms
        mov     cx, 40                          ; Number of 15us in a 1ms
        call    kbdDelay15us                    ; Delay 15us
        pop     cx
        loop    kbdDelay1ms                     ; Repeat until Number of ms
        ret
kbdDelay1ms  ENDP

;+-----------------------------------------------------------------------
;
; kbdDelay1msOBF - Wait a number of milli-seconds for a Controller response.
;
;        Entry: CX = Number of milli-seconds to Wait.
;
;         Exit:  C = Routine timed out (AL = 000h).
;               NC = Successful read (AL = Controller Data).
;
;     Modifies: CX, AL, and the Minor Flags.
;
;   Processing: This routine uses the BIOS delay to decide when to give up
;               waiting for a Keyboard Controller response.  This routine
;               only checks every 1ms for a response.
;
;         Note: The "kbdDelay15us" was measured to be 25us delay.
;
kbdDelay1msOBF  PROC  NEAR  PUBLIC
        push    cx
        in      al, 064h                        ; Read Controller Status
        test    al, BIT0                        ; Is Output Buffer Full?
        jnz     dobf1                           ; Jmp if yes
;       mov     cx, 67                          ; Number of 15us in a 1ms
        mov     cx, 40                          ; Number of 15us in a 1ms
        call    kbdDelay15us                    ; Delay 15us
        pop     cx
        loop    kbdDelay1msOBF
        mov     al, 000h                        ; Clear return data
        stc                                     ; Clr read port 60h
        jmp     dobf2
dobf1:  pop     cx
        in      al, 060h                        ; Read output buffer
        clc                                     ; Set read port 60h
dobf2:  ret
kbdDelay1msOBF  ENDP
;+-----------------------------------------------------------------------
;
; kbdGetVariable - Reads MultiKey Keyboard Controller Variables.
;
;        Entry: AH = MultiKey Variable Index.
;
;         Exit: AL = Variable Value.
;
;     Modifies: AL and Minor Flags.
;
;   Processing: Issue MultiKey Commands B8h (set Index) and BCh to read
;               indexed MultiKey variable.
;
kbdGetVariable  PROC  NEAR  PUBLIC
        mov     al, 0B8h                        ; Set memory index
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, ah                          ; Index number
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
        mov     al, 0BCh                        ; Read virtual memory
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, 0FFh                        ; Null Command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al                        ; Make sure data is from BCh
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
        ret
kbdGetVariable  ENDP
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 12 of 24)

```
;+-----------------------------------------------------------------------------
;
; kbdSetVariable - Writes MultiKey Keyboard Controller Variables.
;
;        Entry: AH = MultiKey Variable Index.
;               AL = Variable Data.
;
;         Exit: None.
;
;     Modifies: Minor Flags.
;
;   Processing: Issue MultiKey Commands B8h (set Index) and BDh to write
;               indexed MultiKey variable.
;
kbdSetVariable  PROC  NEAR  PUBLIC
        push    ax
        mov     al, 0B8h                        ; Set memory index
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, ah                          ; Index number
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
        mov     al, 0BDh                        ; Write virtual memory
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        pop     ax
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
        ret
kbdSetVariable  ENDP

;+-----------------------------------------------------------------------------
;
; kbdGetProcRAM - Reads MultiKey Keyboard Controller Memory.
;
;        Entry: AH = MultiKey Memory Index.
;
;         Exit: AL = Variable Value.
;
;     Modifies: AL and Minor Flags.
;
;   Processing: Issue MultiKey Commands B8h (set Index) and BAh to read
;               indexed MultiKey Memory byte.
;
kbdGetProcRAM  PROC  NEAR  PUBLIC
        mov     al, 0B8h                        ; Set memory index
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, ah                          ; Index number
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
        mov     al, 0BAh                        ; Read 8042 memory
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, 0FFh                        ; Null Command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al                        ; Make sure data is from BAh
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
        ret
kbdGetProcRAM  ENDP

;+-----------------------------------------------------------------------------
;
; kbdSetProcRAM - Writes MultiKey Keyboard Controller Memory.
;
;        Entry: AH = MultiKey Memory Index.
;               AL = Variable Data.
;
;         Exit: None.
;
;     Modifies: Minor Flags.
;
;   Processing: Issue MultiKey Commands B8h (set Index) and BBh to write
;               indexed MultiKey memory byte.
;
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 13 of 24)

```
kbdSetProcRAM  PROC  NEAR  PUBLIC
        push    ax
        mov     al, 0B8h                    ; Set memory index
        call    kbdWait4IBE                 ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, ah                      ; Index number
        call    kbdWait4IBE                 ; Wait for Input Buffer Empty
        out     060h, al
        mov     al, 0BBh                    ; Write 8042 memory
        call    kbdWait4IBE                 ; Wait for Input Buffer Empty
        out     064h, al
        pop     ax
        call    kbdWait4IBE                 ; Wait for Input Buffer Empty
        out     060h, al
        ret
kbdSetProcRAM  ENDP
;==============================================================================
;             M U L T I K E Y / 4 2 I   C O N F I G U R A T I O N
;==============================================================================

kcState         DB      001h    ; (1) Keyboard Controller State flags
kcTmrFlgs       DB      000h    ; (2) Timer Miscellaneous State flags
kcTmRate1       DB      0F7h    ; (3) Timer value 380us, Device Bit time
kcTmRate2       DB      0C4h    ; (4) Timer value 2.4ms, Byte Receive time
kcTmRate3       DB      000h    ; (5) Timer value 11.7ms, Start Bit time
kcTmRate4       DB      0CFh    ; (6) Timer value 0.5s, Flashing LED time
kcTmRate5       DB      000h    ; (7) Timer value 30s-128m, Inactivity time
kcKState1       DB      000h    ; (8) Keyboard ScanCode Set & LED State
kcKState2       DB      000h    ; (9) Keyboard Typematic Delay & Rate

kcMisc          DB      004h    ; Keyboard Controller Miscellaneous flags
kcTst1Pin       DB      000h    ; External Input Event Pin mask (PIN1TSK)
kcTst2Pin       DB      000h    ; External Input Event Pin mask (PIN2TSK)
kcPswNull1      DB      000h    ; Sent when Password enabled (if not 0)
kcPswNull2      DB      000h    ; Sent when Password disabled (if not 0)
kcPswScan1      DB      000h    ; Ignored ScanCode when Password = enabled
kcPswScan2      DB      000h    ; Ignored ScanCode when Password = enabled

kcHotTasks      DW      000F0h, 00000h, 00000h, 00000h, 000D3h
kcLckTasks      DW      00000h, 00000h
kcTmrTask       DW      00801h
kcPinTasks      DW      00000h, 00000h

kcHotKeys       DB      03Bh, 000h, 000h, 000h, 010h, 01Dh, 038h

;------------------------------------------------------------------------------
                ;        ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ MultiKey Variable Index
                ;        ³    ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄ Feature Data Mask
                ;        ³    ³
kcVarsTable     DB      001h,009h, 002h,003h, 003h,0FFh, 004h,0FFh
                DB      005h,0FFh, 006h,0FFh, 007h,0FFh, 008h,000h
                DB      009h,080h, 000h

kcMemoryIdx     DB      002h,01Ch,01Dh,033h,034h,036h,037h,040h,041h,042h
                DB      043h,044h,045h,046h,047h,048h,049h,04Ah,04Bh,04Ch
                DB      04Dh,04Eh,04Fh,050h,051h,052h,053h,054h,055h,056h
                DB      057h,058h,059h,05Ah,000h
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 14 of 24)

```
;==============================================================================
;   K E Y B O A R D   C O N T R O L L E R ' S   C O N F I G U R A T I O N
;==============================================================================

;+-----------------------------------------------------------------------------
;
; kbdCfgController - Download the configuration to the Keyboard Controller.
;
;       Entry: None.
;
;        Exit: None.
;
;    Modifies: All Keyboard Controller variables and the Minor Flags.
;
;  Processing: This routine downloads the Keyboard Controller Configuration
;              and sets all the MultiKey/42i features based on the above
;              table. The above information is a .42i file (exactly as shown)
;              created from the CFG42.EXE (Version 1.4 or above), and
;              imported into the Code. Both the Mouse & Keyboard are
;              disabled. Then the Keyboard Controller Configuration is
;              downloaded through the VirtualRAM (MultiKey Variable) and
;              direct Memory access Commands.
;
;  Assumption: Keyboard Controller is a MultiKey/42i Product and both Mouse
;              and Keyboard device interfaces are Disabled.
;
kbdCfgController  PROC  NEAR  PUBLIC
        push    si
        push    dx
        push    bx
        push    ax
        mov     ah, 000h                        ; Reference Virtual Table Size
        call    kbdGetVariable                  ; AH=Index : AL=Data
        mov     dl, al                          ; DL = # of Supported Indexes
        mov     bx, OFFSET cs:kcState           ; Pointer to the top of Data
        mov     si, OFFSET cs:kcVarsTable       ; Table of Variable indexes
cfg1:   cmp     BYTE PTR cs:[si+0], 0           ; Is this the Table End?
        jz      cfg3                            ; Jmp if yes
        mov     ah, BYTE PTR cs:[si+0]          ; Get index to update
        cmp     ah, dl                          ; Does KBC support variable?
        ja      cfg2                            ; Jmp if no
        call    kbdGetVariable                  ; AH=Index : AL=Data
        mov     dh, BYTE PTR cs:[si+1]          ; Get Mask information
        not     dh                              ; Build up an AND Mask
        and     al, dh                          ; Clear current feature bits
        mov     dh, BYTE PTR cs:[bx]            ; Get Variable Value
        and     dh, BYTE PTR cs:[si+1]          ; Isolate only feature bits
        or      al, dh                          ; Combine the bits
        call    kbdSetVariable                  ; AH=Index, AL=Data
cfg2:   add     si, 2                           ; Next index
        inc     bx                              ; Next data
        jmp     cfg1
cfg3:   mov     bx, OFFSET cs:kcMisc            ; Pointer to the top of Data
        mov     si, OFFSET cs:kcMemoryIdx       ; Table of Memory indexes
cfg4:   cmp     BYTE PTR cs:[si], 0             ; Is this the Table End?
        jz      cfg5                            ; Jmp if yes
        mov     ah, BYTE PTR cs:[si]            ; Get index to update
        mov     al, BYTE PTR cs:[bx]            ; Get Data to update
        call    kbdSetProcRAM                   ; AH=Index, AL=Data
        inc     si                              ; Next index
        inc     bx                              ; Next data
        jmp     cfg4
cfg5:   pop     ax
        pop     bx
        pop     dx
        pop     si
        ret
kbdCfgController  ENDP
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 15 of 24)

```
;+------------------------------------------------------------------------
;
; kbdPortSwapping - Configures which port has a Mouse & which has a Keyboard.
;
;        Entry: None.
;
;         Exit: None.
;
;     Modifies: Keyboard Controller kcState variable and the Minor Flags.
;
;   Processing: To determine which port has a Mouse and which port has
;               Keyboard, issue an ECh (Reset Echo Command) to both ports
;               and observe the responses.
;
;                    ÚÄÄÄÄYÄÄ AT Environment (kcState.1 = 1)?
;                    ³        Set Ports Not Swapped (kcState.0 = 1)
;                    ³        Set Devices = Active
;                    ³        Flush Both Device Ports
;                    ³        Issue Aux Cmd = ECh
;                    ³ ÚÄÄYÄÄ Valid Aux Device (Rsp = FAh)?
;                    ³ ³      Device Attached (Sts = no Error)? ÄÄNÄÄ¿
;                    ³ ³      Issue Kbd Cmd = ECh                    ³
;                    ³ ÃÄÄNÄÄ Valid Aux Device (Rsp = FAh)?          ³
;                    ³ ³      Set Ports Swapped (kcState.0 = 0) <ÄÄÄÄÙ
;                    ³ ÀÄÄÄÄ> Set Devices = Inactive
;                    ÀÄÄÄÄÄÄ> Done
;
;         Note: This Routine must be executed before the Device Reset Code.
;
;   Assumption: Keyboard Controller = /42G,/C42,/42E, or /42i Product
;
kbdPortSwapping  PROC  NEAR  PUBLIC
        push    cx
        push    ax
        mov     ah, 001h                        ; Reference kcState
        call    kbdGetVariableFar               ; AH=Index : AL=Data
        test    al, BIT1                        ; Is it an AT Environment?
        jnz     port4                           ; Jmp if yes
        mov     al, 060h                        ; Write KCCB command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, 044h                        ; Set Devices = Active
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
        mov     ah, 001h                        ; Reference kcState
        call    kbdGetVariableFar               ; AH=Index : AL=Data
        or      al, BIT0                        ; {kcState.0} set Not Swapped
        call    kbdSetVariable                  ; AH=Index, AL=Data
port1:  mov     cx, 6                           ; Wait up to 6ms for any data
        call    kbdDelay1msOBF                  ; Wait CX * 1ms for OBF
        jnc     port1                           ; Jmp if there was data
        mov     al, 0ECh                        ; Reset Echo (Wrap) mode Cmd
        call    kbdSend2Mouse                   ; AL => Mouse wait for rsp
        jz      port3                           ; An Acknowledge? (jmp if yes)
        test    ah, BIT6                        ; Is there a Device Attached?
        jz      port2                           ; Jmp if yes
        mov     al, 0ECh                        ; Invalid Keyboard Command
        call    kbdSend2Keyboard                ; AL => Keyboard wait for rsp
        jnz     port3                           ; An Acknowledge? (jmp if no)
port2:  mov     ah, 001h                        ; Reference kcState
        call    kbdGetVariableFar               ; AH=Index : AL=Data
        and     al, not BIT0                    ; {kcState.0} set Swapped
        call    kbdSetVariable                  ; AH=Index, AL=Data
port3:  mov     al, 060h                        ; Write KCCB command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, 074h                        ; Set Devices = Inactive
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
port4:  pop     ax
        pop     cx
        ret
kbdPortSwapping  ENDP
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 16 of 24)

```
;+--------------------------------------------------------------------------
;
; kbdCalibrateTmrs - Calibrate MultiKey/42i's Inactivity Timer values.
;
;        Entry: None.
;
;         Exit: None.
;
;     Modifies: Keyboard Controller tmRate4 variable and the Minor flags.
;
;   Processing: Calibrate the MultiKey/42i's Inactivity Timers by measuring
;               the time it takes to respond from the AAh (Self Test) Command.
;               The AAh Command re-initializes the KCCB (Keyboard Controller
;               Command Byte), so it must be saved and restored.  The Timers
;               are adjusted by updating tmRate4 variable as per the
;               following:
;
;                  clock = The Keyboard Controller Input Clock (MHz)
;                  cycle = Processor Cycle Execution Time (æs)
;                  cycle = 15/clock
;                  clk10 = 10*clock
;                      T = Measured Length of the AAh Command (æs)
;                      T = [2400+(76*cycle)]/0.838
;                   2400 = cycle*32*(256-TMRATE2)
;                      T = [cycle*32*(256-TMRATE2)+(76*cycle)]/0.838
;                      T = [(15/clock)*32*(256-TMRATE2)+(76*(15/clock))]/0.838
;                  clock = [572.8*(256-TMRATE2)+1360.4]/T
;                      clk10 = [5728*(256-TMRATE2)+13604]/T
;                 0.1172 = 30/256
;                 117200 = cycle*32*256*(256-TMRATE4)
;                  11720 = [122880*(256-TMRATE4)]/clk10
;                tmRate4 = 256-[((95*clk10)+500)/1000]
;
;   Assumption: Keyboard Controller = MultiKey/42i Product
;
kbdCalibrateTmrs  PROC  NEAR  PUBLIC
        pushf
        push    ax
        push    bx
        push    dx
        cli
        call    getTimerCnfg                ; Get Current Timer0 state
        push    ax                          ; Save for restoration
        mov     al, 00110100b               ; Tmr0 => LSB 1st,Mode2,binary
        mov     bx, 0FFFFh                  ; 54.9ms Interrupt output
        call    configureTimer              ; Program the 8254 Timer Chip
        mov     al, 020h                    ; Issue Read KCCB Command
        call    kbdWait4IBE                 ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, 0FFh                    ; Issue Null Command
        call    kbdWait4IBE                 ; Wait for Input Buffer Empty
        out     064h, al
        call    kbdWait4IBE                 ; Wait for Input Buffer Empty
        call    kbdWait4OBF                 ; Wait for Output Buffer Full
        push    ax                          ; Save KCCB for later
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 17 of 24)

```
kbc1:   in      al, 060h                        ; Flush 8042 output buffer
        mov     al, 0AAh                        ; 8042 Self Test Command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        call    readTimer
        mov     bx, ax                          ; Save for time calculation
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
        cmp     al, 055h                        ; Was it a successful test?
        jnz     kbc1                            ; Jmp if no
        call    readTimer                       ; Tick=1.193182 MHz clock rate
        sub     bx, ax                          ; Calculate delta time (T)
        js      kbc1                            ; If rollover, do again
        mov     ah, 004h                        ; Reference tmRate2
        call    kbdGetVariable                  ; AH=Index : AL=Data
        mov     ah, 000h
        neg     al                              ; 256-TMRATE2
        mov     dx, 5728
        mul     dx                              ; 5728 * (256-tmRate2)
        add     ax, 13604                       ; 5728 * (256-tmRate2) + 13604
        adc     dx, 0                           ; Add in carry if present
        div     bx                              ; Divide by T (measured)
        mov     bx, 95
        mul     bx                              ; AX = (95 * clk10)
        add     ax, 500                         ; Add in 0.5
        mov     dx, 00000h                      ; DXAX = [95 * clk10 + 500]
        mov     bx, 003E8h                      ; BX = 1000
        div     bx                              ; AX = [95(10*clk)+500]/1000
        neg     ax                              ; AL = 0-[95(10*clk)+500]/1000
        mov     ah, 006h                        ; Timer Compensation (tmRate4)
        call    kbdSetVariable                  ; AH=Index, AL=Data
        mov     al, 060h                        ; Write KCCB command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        pop     ax                              ; Get original KCCB
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
        pop     ax                              ; Get original Timer0 state
        mov     bx, 0FFFFh                      ; 54.9ms Interrupt output
        call    configureTimer                  ; Program the 8254 Timer Chip
        pop     dx
        pop     bx
        pop     ax
        popf
        ret
kbdCalibrateTmrs  ENDP
;----------------------------------------------------------------------------
;
; readTimer - Latches the 8254 (System Timer) Data.
;
;       Entry: None.
;
;        Exit: AX = Timer Value.
;
;    Modifies: AX and Minor Flags.
;
;  Processing: Latch the 8254 Data using an Out to Port EDh as the "Wait for
;              I/O" delay.
;
readTimer  PROC  NEAR
        mov     al, 0                           ; Set counter latch for 8254
        out     043h, al
        out     0EDh, al                        ; System I/O BUS Delay
        in      al, 040h                        ; Read low byte
        mov     ah, al
        out     0EDh, al                        ; System I/O BUS Delay
        in      al, 040h                        ; Read high byte
        xchg    ah, al
        ret
readTimer  ENDP
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 18 of 24)

```
;------------------------------------------------------------------------------
;
; configureTimer - Program the System Timer (8254) to a particular Mode.
;
;       Entry: AL = Timer Control Byte.
;              BX = The Interrupt Rate.
;
;        Exit: None.
;
;    Modifies: Minor Flags.
;
;  Processing: Write to the 8254's Control register the Control Byte.
;
configureTimer  PROC  NEAR
        push    ax
        out     043h, al                        ; Set Timer Mode as pass in
        out     0EDh, al                        ; System I/O BUS Delay
        mov     ax, bx                          ; 1.193182 MHz clock rate
        out     040h, al                        ; Program LSByte first
        out     0EDh, al                        ; System I/O BUS Delay
        mov     al, ah
        out     040h, al                        ; Program MSByte second
        pop     ax
        ret
configureTimer  ENDP

;------------------------------------------------------------------------------
;
; getTimerCnfg - Read and Save the Original Timer0 State of the 8254.
;
;       Entry: None.
;
;        Exit: AL = Timer Control Byte.
;
;    Modifies: AL = Minor Flags.
;
;  Processing: Read the Timer0 Control Byte from the Control Register.
;
getTimerCnfg  PROC  NEAR
        mov     al, 11100010b                   ; Set Latch Status of Timer0
        out     043h, al
        out     0EDh, al                        ; System I/O BUS Delay
        in      al, 40h                         ; Read the Status byte
        and     al, 00111111b                   ; Setup Timer0 for restore
        ret
getTimerCnfg  ENDP

;==============================================================================
;           M U L T I K E Y / 4 2 I   S E T U P   R O U T I N E S
;==============================================================================

;+-----------------------------------------------------------------------------
;
; kbdWait4Security - Waits for Security to be disabled.
;
;       Entry: None.
;
;        Exit: AL = SMI value for Password Entered.
;
;    Modifies: Minor Flags.
;
;  Processing: Waits for Security to be disabled by watching the Status port
;              bit 4 (uninhibited).  Then inquires which Password was
;              entered.
;
;  Assumption: Keyboard Controller = MultiKey/42i Product
;
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 19 of 24)

```
kbdWait4Security  PROC  NEAR  PUBLIC
        push    bx
        mov     bx, ax                          ; Save AH value
kws1:   in      al, 064h                        ; Get Controller Status
        test    al, BIT4                        ; Is Security still enabled?
        jz      kws1                            ; Jmp if yes
        mov     ah, 00Ah                        ; Reference SMI Function value
        call    kbdGetVariable                  ; AH=Index : AL=Data
        mov     ah, bh                          ; Restore AH value
        pop     bx
        ret
kbdWait4Security  ENDP

;+-----------------------------------------------------------------------
;
; kbdSetInactiveTmr - Updates the Inactivity Timer value.
;
;        Entry: AL = Timer value in 30 second intervals.
;               AL = 000h, Disables Inactivity Timer and Resumes.
;
;         Exit: None.
;
;     Modifies: Keyboard Controller variables and the Minor Flags.
;
;   Processing: Update the Inactivity Timer in the Keyboard Controller.
;
;   Assumption: Keyboard Controller = MultiKey/42i Product
;
kbdSetInactiveTmr  PROC  NEAR  PUBLIC
        push    ax
        mov     al, 0AFh                        ; Setup Inactivity Timer
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        pop     ax
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al
        ret
kbdSetInactiveTmr  ENDP

;+-----------------------------------------------------------------------
;
; kbdGetInactiveTmr - Reads the Inactivity Timer's current value.
;
;        Entry: None.
;
;         Exit: AL = Timer value in 30 second intervals.
;               AL = 000h, Inactivity Timer is disabled.
;
;     Modifies: AX and the Minor Flags.
;
;   Processing: Read the MultiKey variable (index 7) from the Keyboard
;               Controller. Since the Keyboard Controller stores the variable
;               internally as the Two's Compliement of the value set in the
;               kbdSetInactiveTmr routine.
;
;
;   Assumption: Keyboard Controller = MultiKey/42i Product
;
kbdGetInactiveTmr  PROC  NEAR  PUBLIC
        mov     ah, 007h                        ; Reference kcTmRate5 value
        call    kbdGetVariable                  ; AH=Index : AL=Data
        neg     al                              ; Build up two's compliement
        ret
kbdGetInactiveTmr  ENDP
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 20 of 24)

```
;--------------------------------------------------------------------------
;
; kbdLoadPassword - Down the Password to the Keyboard Controller.
;
;        Entry: AL = Load Password Command (A5h=Normal, A3h=Extended).
;               DS:BX => Password (null terminated string).
;
;         Exit: None.
;
;     Modifies: Minor Flags.
;
;   Processing: Load either the normal or extended Password based on the
;               Command passed through AL. Send all ScanCodes in the Password
;               String including the Zero.  Password maximum size is 16 bytes
;               plus 1 for the null (000h) value.
;
;         Note: Once one of the Passwords is loaded the Keyboard Controller's
;               Configuration cannot be changed.
;
kbdLoadPassword  PROC  NEAR
        push    bx
        push    ax
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al                        ; Issue Password Command
klp1:   mov     al, BYTE PTR ds:[bx]            ; Get Password ScanCode
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     060h, al                        ; Send Password Data
        inc     bx                              ; Next Password entry
        cmp     al, 000h                        ; Was that the Password end?
        jnz     klp1                            ; Jmp if no
        pop     ax
        pop     bx
        ret
kbdLoadPassword  ENDP
;==========================================================================
;         K E Y B O A R D / M O U S E   P O S T   R O U T I N E S
;==========================================================================

;+-------------------------------------------------------------------------
;
; kbdStopDevices - Disable Keyboard and Auxiliary Devices at Devices.
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: Minor Flags.
;
;   Processing: Issue Keyboard Command F5h (disable Scanning) and turn off
;               Keyboard LEDs to save power (17ma per external LED).  If this
;               PS/2 Environment issue Mouse Command F5h (disable Movement).
;               Leave both interfaces disabled in addition, since the User
;               could plug in the Keyboard and enabled the Device.
;
kbdStopDevices  PROC  NEAR  PUBLIC
        push    ax
        test    BYTE PTR ds:[verFlags], BIT4    ; Is it a PS/2 environment?
        jz      tdo1                            ; Jmp if no
        mov     al, 0F5h                        ; Disable AuxDevice @Mouse
        call    kbdSend2Mouse                   ; AL => Mouse wait for rsp
        mov     al, 0A7h                        ; Disable AuxDevice interface
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
tdo1:   mov     al, 0F5h                        ; Disable Keyboard @Device
        call    kbdSend2Keyboard                ; AL => Keyboard wait for rsp
        mov     al, 0EDh                        ; Set/Clear LED Command
        call    kbdSend2Keyboard                ; AL => Keyboard wait for rsp
        mov     al, 000h                        ; Turn off all LEDs
        call    kbdSend2Keyboard                ; AL => Keyboard wait for rsp
        mov     al, 0ADh                        ; Disable Keyboard interface
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        pop     ax
        ret
kbdStopDevices  ENDP
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 21 of 24)

```
;+-------------------------------------------------------------------------
;
; kbdFlushDevices - Check and flush Data from Keyboard Controller and Devices.
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: Minor Flags.
;
;   Processing: This is a very important routine which trys to determine when
;               the Devices are ready to accept Commands and at the same time
;               flushes the Devices Buffers.  This routines leaves both
;               Devices Disabled.  If either Device after all the retrys
;               fails the Command the Device is marked not present. When the
;               Keyboard and AuxDevice commands are issued the interface is
;               automatically open that what makes this routine affective.
;
kbdFlushDevices  PROC  NEAR  PUBLIC
        push    cx
        push    ax
        and     BYTE PTR ds:[verFlags], not BIT1; Set no Keyboard attached
        mov     cx, 3                           ; Number of retries
flsh1:  push    cx
        mov     cx, 500                         ; Delay 0.5s for devices
        call    kbdDelay1ms                     ; Wait CX * 1ms
        in      al, 060h                        ; Flush 8042 output buffer
        mov     al, 0F5h                        ; Disable keyboard @device
        call    kbdSend2Keyboard                ; AL => Keyboard wait for rsp
        pop     cx
        cmp     al, 0FAh                        ; Is it an Acknowledgement?
        loopnz  flsh1                           ; Retry if no
        jnz     flsh2                           ; Jmp if no
        or      BYTE PTR ds:[verFlags], BIT1    ; Set Keyboard attached
flsh2:  test    BYTE PTR ds:[verFlags], BIT4    ; Is it a PS/2 environment?
        jz      flsh4                           ; Jmp if no
        and     BYTE PTR ds:[verFlags], not BIT2; Set no PS/2 Mouse attached
        mov     cx, 3                           ; Number of retries
flsh3:  push    cx
        mov     cx, 500                         ; Delay 0.5s for devices
        call    kbdDelay1ms                     ; Wait CX * 1ms
        in      al, 060h                        ; Flush 8042 output buffer
        mov     al, 0F5h                        ; Disable AuxDevice @device
        call    kbdSend2Mouse                   ; AL => Mouse wait for rsp
        pop     cx
        cmp     al, 0FAh                        ; Is it an Acknowledge?
        loopnz  flsh3                           ; Retry if no
        jnz     flsh4                           ; Jmp if no
        or      BYTE PTR ds:[verFlags], BIT2    ; Set PS/2 Mouse attached
flsh4:  mov     cx, 5                           ; Wait up to 5ms
        call    kbdDelay1msOBF                  ; Wait CX * 1ms for OBF
        jnc     flsh4                           ; Jmp if there was data
        pop     ax
        pop     cx
        ret
kbdFlushDevices  ENDP

;+-------------------------------------------------------------------------
;
; kbdRstController - Reset Keyboard Controller using the "SelfTst" Command.
;
;        Entry: DS = DATA segment.
;
;         Exit: None.
;
;     Modifies: Minor Flags.
;
;   Processing: Issue Command AAh (Controller Self Test) and wait for the
;               response.  This must be the first Controller Command.
;
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 22 of 24)

```
kbdRstController  PROC  NEAR  PUBLIC
        push    ax
        mov     al, 0AAh                        ; 8042 Self Test command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        mov     al, 0FFh                        ; Null Command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al                        ; Make last Command complete
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
        mov     al, 0ADh                        ; Disable kbd device interface
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        test    BYTE PTR ds:[verFlags], BIT4    ; Is it a PS/2 environment?
        jz      rCtrl1                          ; Jmp if no
        mov     al, 0A7h                        ; Disable aux device interface
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
rCtrl1: pop     ax
        ret
kbdRstController  ENDP
;+-----------------------------------------------------------------------
;
; kbdRstKeyboard - Reset Keyboard and leave the Device Disabled.
;
;       Entry: DS = DATA segment.
;
;        Exit: None.
;
;    Modifies: Minor Flags.
;
;  Processing: If a Keyboard is attached, issue Keyboard Command FFh (Reset)
;              and wait for the responses.  The Keyboard is then disabled
;              at the Device along with the interface.
;
kbdRstKeyboard  PROC  NEAR  PUBLIC
        push    cx
        push    ax
        test    BYTE PTR ds:[verFlags], BIT1    ; Is there a Kbd attached?
        jz      krst2                           ; Jmp if no
        in      al, 060h                        ; Flush Output Buffer
        mov     al, 0FFh                        ; Reset Keyboard command
        call    kbdSend2Keyboard                ; AL => Keyboard wait for rsp
        cmp     al, 0FEh                        ; Is it an transmission error?
        jz      krst2                           ; Jmp if yes
        cmp     al, 0AAh                        ; Is it the answer already?
        jz      krst1                           ; Jmp if yes
        cmp     al, 0FAh                        ; Is it an Acknowledge?
        jnz     krst1                           ; Jmp if no
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
krst1:  mov     al, 0F5h                        ; Disable keyboard @device
        call    kbdSend2Keyboard                ; AL => Keyboard wait for rsp
krst2:  mov     al, 0ADh                        ; Disable keyboard interface
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        pop     ax
        pop     cx
        ret
kbdRstKeyboard  ENDP

;+-----------------------------------------------------------------------
;
; kbdRstPS2Mouse - Reset the Auxiliary Device and leave the Device Disabled.
;
;       Entry: DS = DATA segment.
;
;        Exit: None.
;
;    Modifies: Minor Flags.
;
;  Processing: If it is a PS/2 Environment and a Mouse is attached, issue
;              Mouse Command FFh (Reset) and wait for the responses.  The
;              Mouse is then disabled at the Device along with the interface.
;
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 23 of 24)

```
kbdRstPS2Mouse  PROC  NEAR  PUBLIC
        push    ax
        test    BYTE PTR ds:[verFlags], BIT4    ; Is it a PS/2 environment?
        jz      mrst4                           ; Jmp if no
        test    BYTE PTR ds:[verFlags], BIT2    ; Is there a Mouse attached?
        jz      mrst3                           ; Jmp if no
        in      al, 060h                        ; Flush Output Buffer
        mov     al, 0FFh                        ; Reset Aux Device command
        call    kbdSend2Mouse                   ; AL => Mouse wait for rsp
        cmp     al, 000h                        ; Is it the answer already?
        jz      mrst2                           ; Jmp if yes
        cmp     al, 0AAh                        ; Is it the answer already?
        jz      mrst1                           ; Jmp if yes
        cmp     al, 0FAh                        ; Is it an Acknowledge?
        jnz     mrst3                           ; Jmp if no (Error)
        call    kbdWait4OBF                     ; Wait for Output Buffer Full
        cmp     al, 000h                        ; Is it the answer already?
        jz      mrst2                           ; Jmp if yes
        cmp     al, 0AAh                        ; Is it the answer?
        jnz     mrst2                           ; Jmp if no (Error)
mrst1:  call    kbdWait4OBF                     ; Wait for Output Buffer Full
mrst2:  mov     al, 0F5h                        ; Disable AuxDevice @device
        call    kbdSend2Mouse                   ; AL => Mouse wait for rsp
mrst3:  mov     al, 0A7h                        ; Disable aux device interface
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
mrst4:  pop     ax
        ret
kbdRstPS2Mouse  ENDP

;+-----------------------------------------------------------------------------
;
; kbdSend2Keyboard - Sends Command/Data to Keyboard Device.
;
;         Entry: AL = Command/Data to be sent.
;
;          Exit: AH = Controller Status.
;                AL = Response from Device.
;
;      Modifies: AL and Minor Flags.
;
;    Processing: Send Command/Data to Port 60h (send to KbdDevice) and wait
;                for the response.
;
kbdSend2Keyboard  PROC  NEAR  PUBLIC
        push    ax
        in      al, 060h                        ; Flush Output Buffer
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        pop     ax
        out     060h, al                        ; Send Command/Data to Kbd
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        call    kbdStatusOBF                    ; Wait for OBF (AH = Status)
        ret
kbdSend2Keyboard  ENDP

;+-----------------------------------------------------------------------------
;
; kbdSend2Mouse - Sends Command/Data to Auxiliary Device.
;
;         Entry: AL = Command/Data to be sent.
;
;          Exit: AH = Controller Status.
;                AL = Response from Device.
;
;      Modifies: AL and Minor Flags.
;
;    Processing: Issue Command D4h (send to AuxDevice) and wait for the
;                response.
;
```

Figure 5-1. Sample Keyboard Controller Code. (sheet 24 of 24)

```
kbdSend2Mouse  PROC  NEAR  PUBLIC
        push    ax
        mov     al, 0D4h                        ; Send to AuxDevice command
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        out     064h, al
        call    kbdWait4IBE                     ; Wait for Input Buffer Empty
        in      al, 060h                        ; Flush Output Buffer
        pop     ax                              ; Get Data to be sent
        out     060h, al
        call    kbdStatusOBF                    ; Wait for OBF (AH = Status)
        ret
kbdSend2Mouse  ENDP

;-----------------------------------------------------------------------------

        CODE    ENDS
```

This page left blank.

Index

This page left blank.

Abbreviated Tables

The following pages contain tables that are linked to text in the preceding chapters. They are part of the interactive file.

these pages are not standard elements of the MultiKey/42i Developer's Technical Reference. They do not contain page numbers. The bottom of each page is marked:

Remove this page before photocopying and binding this document.

If you intend to print this file, to keep it as a hard copy resource, make sure you remove this page and all of the following pages, afterwards.

Table 2-2. Memory Map. (sheet 1 of 3)

Symbol	RAM Location (Range)	Description
KCMISC	02h	Keyboard Controller Miscellaneous Flags Bit7 - Auxiliary Expecting Response (bit1) Bit6 - Auxiliary Expecting Response (bit0) Bit5 - Keyboard Expecting Response (bit1) Bit4 - Keyboard Expecting Response (bit0) Bit3 - Auxiliary Expecting Four Responses Bit2 - No D2h Command Password checking Bit1 - Password Loaded, Memory is Read-Only Bit0 - Security is Enabled

Table 2-2. Memory Map. (sheet 3 of 3)

Symbol	RAM Location (Range)	Description
KEY1TSK	040h-041h	HotKey1 Pin Control Task Value (2 bytes)
KEY2TSK	042h-043h	HotKey2 Pin Control Task Value (2 bytes)
KEY3TSK	044h-045h	HotKey3 Pin Control Task Value (2 bytes)
KEY4TSK	046h-047h	HotKey4 Pin Control Task Value (2 bytes)
KEY5TSK	048h-049h	HotKey5 & Inactivity Timer Pin Control Task Value (2 bytes)

Table 2-2. Memory Map.(sheets 2 and 3)

Symbol	RAM Location (Range)	Description
TMRATE1	027h	Timer value 380µs, Device Bit Time
TMRATE2	028h	Timer value 2.4ms, Byte Receive Time
TMRATE3	029h	Timer value 11.7ms, Start Bit Time
TMRATE4	038h	Timer value 0.12s, Compensation Time
TMRATE5	039h	Timer value 30s-128m, Inactivity Time

Table 2-2. Memory Map. (sheet 2 of 3)

Symbol	RAM Location (Range)	Description
TMRFLGS	01Eh	Timer Miscellaneous State flags Bit7 - Flashing LED Counter (bit1) Bit6 - Flashing LED Counter (bit0) Bit5 - Reserved Bit4 - Reserved Bit3 - Flashing LED Task Pending Bit2 - Keyboard Controller Suspended Bit1 - KEY5TSK is only for HotKey 5 Bit0 - Flashing LED when Suspended

Table 5-2. Memory Map. (sheet 2 of 3)

Symbol	RAM Location (Range)	Description
TST1PIN	01Ch	External Input Event Pin Mask (PIN1TSK)
TST2PIN	01Dh	External Input Event Pin Mask (PIN2TSK)

Table 2-2. Memory Map. (sheet 3 of 3)

Symbol	RAM Location (Range)	Description
PIN1TSK	050h-051h	External Input Event1 Pin Control Task Value (2 bytes)
PIN2TSK	052h-053h	External Input Event2 Pin Control Task Value (2 bytes)

Table 3-2. Standard Command Set.

Command	Description
20h-3Fh	Read the contents of the designated RAM locations (20h-3Fh) and send it to System
60h-7Fh	Get a byte of data from System and write into one of locations (20h-3Fh)

Table 3-2. Standard Command Set.

Command	Description
D2h	Send data back to the System as if it came from the Keyboard
D3h	Send data back to the System as if it came from the Auxiliary Device (PS/2 Mouse)

Table 3-3. Extended Command Set.

Command	Description
B8h	Setup Phoenix Extended Memory Access INDEX
B9h	Get current Phoenix Extended Memory Access INDEX
BAh	Get current Phoenix Extended Memory referenced by INDEX Cannot read the Password Storage Area
BBh	If neither Password is loaded, write Phoenix Extended Memory referenced by INDEX. Cannot write the Password Storage Area. Once the Password is loaded, memory is locked

Table 3-3. Extended Command Set.

Command	Description
AFh	Set Inactivity Timer value from 0.5 to 128 minutes (zero disables timer)

Table 2-2. Memory Map. (sheet 3 of 3)

Symbol	RAM Location (Range)	Description
TMR1TSK	04Eh-04Fh	Inactivity Timer Pin Control Task Value (2 bytes)

Table 3-3. Extended Command Set.

Command	Description		
BCh - BDh	Read/Write the following MultiKey variables referenced by INDEX:		
	LENGTH	(0)	Number of MultiKey variables
	KCSTATE	(1)	Keyboard Controller State flags
	TMRFLGS	(2)	Timer Miscellaneous State flags
	TMRATE1	(3)	Timer value 380ms, Device Bit Time
	TMRATE2	(4)	Timer value 2.4ms, Byte Receive Time
	TMRATE3	(5)	Timer value 11.7ms, Start Bit Time
	TMRATE4	(6)	Timer value 0.12seconds, Compensation Time
	TMRATE5	(7)	Timer value 30 seconds to 128 minutes, Inactivity Time
	KSTATE1	(8)	Keyboard Scan Code Set & LED state
	KSTATE2	(9)	Keyboard Typematic Delay & Rate
	FUNCTION	(A)	Interrupt Function Request value

Table 2-2. Memory Map. (sheet 3 of 3)

Symbol	RAM Location (Range)	Description
FUNCTION	035h	Interrupt Function Request Value

Remove this page before photocopying and binding this document

Table 3-3. Extended Command Set.

Command	Description		
BCh - BDh	Read/Write the following MultiKey variables referenced by INDEX:		
	LENGTH	(0)	Number of MultiKey variables
	KCSTATE	(1)	Keyboard Controller State flags
	TMRFLGS	(2)	Timer Miscellaneous State flags
	TMRATE1	(3)	Timer value 380ms, Device Bit Time
	TMRATE2	(4)	Timer value 2.4ms, Byte Receive Time
	TMRATE3	(5)	Timer value 11.7ms, Start Bit Time
	TMRATE4	(6)	Timer value 0.12seconds, Compensation Time
	TMRATE5	(7)	Timer value 30 seconds to 128 minutes, Inactivity Time
	KSTATE1	(8)	Keyboard Scan Code Se t & LED state
	KSTATE2	(9)	Keyboard Typematic Delay & Rate
	FUNCTION	(A)	Interrupt Function Request value

Table 2-2. Memory Map.

Symbol	RAM Location (Range)	Description
KSTATE1	018h	Keyboard Scan Code Set and LED State Bit7 - Keyboard Disabled at Device Bit6 - Reserved Bit5 - Scan Code Set Bit1 Bit4 - Scan Code Set Bit0 Bit3 - Reserved Bit2 - Caps Lock LED Bit1 - Num Lock LED Bit0 - Scroll Lock LED
KSTATE2	019h	Keyboard Typematic Delay and Rate Bit7 - Transparent Security Mode Bit6 - Typematic Delay Bit1 Bit5 - Typematic Delay Bit0 Bit4 - Typematic Rate Bit4 Bit3 - Typematic Rate Bit3 Bit2 - Typematic Rate Bit2 Bit1 - Typematic Rate Bit1 Bit0 - Typematic Rate Bit0

Table 2-2. Memory Map. (sheet 3 of 3)

Symbol	RAM Location (Range)	Description
LCK1TSK	04Ah-04Bh	Normal Password Pin Control Task Value (2 bytes)
LCK2TSK	04Ch-04Dh	Extended Password Pin Control Task Value (2 bytes)

Table 2-2. Memory Map. (sheet 1 of 3)

Symbol	RAM Location (Range)	Description
KCSTATE	03h	Keyboard Controller State Flags Bit7 - OBF Data is not pending Bit6 - Internal Device Command flag Bit5 - Auxiliary Device Disabled Bit4 - Keyboard Device Disabled Bit3 - Use RAM Scan Code Conversion Table Bit2 - Not Waiting for Keyboard LED Data Bit1 - AT Environment (0=PS/2) Bit0 - Keyboard/Auxiliary Ports Not Swapped

Table 2-2.   Memory Map. (sheet 2 of 3)

Symbol	RAM Location (Range)	Description
KCCB	020h	Keyboard Controller Command Byte Bit7 - Reserved Bit6 - Convert Scan Codes Bit5 - Auxiliary Disabled Bit4 - Keyboard Disabled Bit3 - Reserved Bit2 - System Flag Bit1 - Auxiliary Interrupt Enabled Bit0 - Keyboard Interrupt Enabled

Table 3-3. Extended Command Set.

Command	Description
BCh - BDh	Read/Write the following MultiKey variables referenced by INDEX: LENGTH (0) Number of MultiKey variables KCSTATE (1) Keyboard Controller State flags TMRFLGS (2) Timer Miscellaneous State flags TMRATE1 (3) Timer value 380ms, Device Bit Time TMRATE2 (4) Timer value 2.4ms, Byte Receive Time TMRATE3 (5) Timer value 11.7ms, Start Bit Time TMRATE4 (6) Timer value 0.12seconds, Compensation Time TMRATE5 (7) Timer value 30 seconds to 128 minutes, Inactivity Time KSTATE1 (8) Keyboard Scan Code Set & LED state KSTATE2 (9) Keyboard Typematic Delay & Rate FUNCTION (A) Interrupt Function Request value

Table 2-2. Memory Map. (sheet 2 of 3)

Symbol	RAM Location (Range)	Description
TMRFLGS	01Eh	Timer Miscellaneous State flags Bit7 - Flashing LED Counter (bit1) Bit6 - Flashing LED Counter (bit0) Bit5 - Reserved Bit4 - Reserved Bit3 - Flashing LED Task Pending Bit2 - Keyboard Controller Suspended Bit1 - KEY5TSK is only for HotKey 5 Bit0 - Flashing LED when Suspended