The ADB uC Tool Set REV 1.1

Peter Baum May. 15, 1986

The ADB uC tool set is used to send commands and data between the Single-Chip Microcontroller (uC) and the system. Typically, the tool set will be used to control ADB activity, but other commands, which are used by diagnostic routines and the control panel, are available.

The tool set follows the standard convention of passing data and pointers in the stack, then calling the tool locator with the tool set and function number in the X register.

Tool Set Number = 9

Some commands can return an error code indicating busy. This usually means that part of another command is currently active.. Rather than queue up the command, the tool set is putting the burden on the calling routine to try again. A calling routine can retry the tool set immediately or it can try later (maybe using the heartbeat chain to remind itself to try again).

More details of each function can be found in the Single-Chip Microcontroller (SKI) documentation, the Apple Desktop Bus Microcontroller documenation, and the ADB specification. This document assumes the user has knowledge of how ADB works.

An application which intends on using specific ADB devices other than the mouse or keyboard must handle them with a driver. This driver will consist of some setup routines and data handling routines. The setup routines are used to identify devices on ADB, possibly changing ADB addresses and handlers. The data handling routines consist of a completion routine, which is called by the system when data is received from a device, and any other routines which may be used to operate on the data.

A special tool function can be used to automatically poll ADB for data from specific devices. If data is received then that devices completion routine is called. This mechanism is called the SRQ list. The system will automatically start polling the devices in the list whenever any device on the bus asserts SRQ.

The SRQ function, SRQPL, is the most efficient way for a single user application to gather data from a ADB device. This mechanism assumes that the user rarely switches between devices. When SRQ is detected, the system will always start looking for data by polling the last device used.

In multi-user applications such as two (or more) player games the SRQ list will not work efficiently, since it always give priority to the last device which returned data. For these applications, each device should be polled seperately using the POLL ADB command. This allows the application to guarantee that devices can be read in an arbitrary fashion with no device getting priority (unless application wants that) and also allows the application to regulate how often data is read. The latter feature is very important since it allows a game to adjust depending on the number of players.

To poll each device seperately it must have a unique address. Currently there is no support for this. There are two suggested methods for an application to assign a unique address to each device.

A simple method is to request a player to hold down the activator button (mouse button or keyboard open apple key) and then use the ADB Change Address When Activated command. This command changes the address of any device(s) that is currently activated. After verifying that a device has changed addresses then tell the player to release the button. Repeat this for each player, giving each one a new and different address.

Another mechanism for moving each ADB device to a unique address involves using collision detection. The host requests the ID at a specific address (TALK-REG.3). The change address command is then issued using the Collision Detect Handler. Any device which did NOT detect a collision will change its address. There is a chance that two (or more) devices may not detect a collision and both will move to the new address. To alleviate this problem, the device(s) at the new address should be moved many times. This raises the chances that the devices will collide and only a single device will be moved. Theoretically, two devices may collide 1 out of 16 times. If there are more than two devices on the bus then the chances of a collision rise quickly. With three devices then there a 1/5 chance that two devices may not collide. (4 devices = 1/3 & 5 devices = 1/2).

For example, if an application wanted to distinguish 4 keyboards from each other then it should send the ID command (TALK Reg. 3 Address 2) then issue the change address command, to address 8, with the collision detect handler (=$FE). Any device that didn't detect a collision (at least one) will change to the new address. This same scenario should be repeated but instead changing address 8 to 9. Move any device which stays at address 8 (it lost the collision) back to address 2. Continue swapping the device another 30 times between addresses 8 and 9, always moving any losers back to address 2. Swapping 32 times gives a very good statistical chance that only one device will have had its address changed to 8 from the original keyboard address. This process should be repeated for each keyboard, using two open addresses (such as 9 & 10, then 10 & 11 etc.).

After each keyboard has been moved to a new address the application should ask each user to press a key. The key press can then be used to identify the address of each user.

The SRQ list and ABSOLUTE flag do not get cleared on RESET. Applications which install devices should use the CLRSRQTBL command during RESET.

The order that the parameters are listed for each function represent the order which they should be pushed on the stack. (i.e. Command is usually the last value pushed on the stack).

The following examples are written in 16-bit mode with the following equates:

```
UCTOOL EQU  $9        ;ADB TOOL NUMBER
SND    EQU $9         ;SEND FUNCTION NUMBER
```

Here is an example of how to enable SRQ on a device at address 7:

```
ENSRQ EQU *
   PEA  $0000        ;COUNT OF 0 BYTES (=HOWMANY)
   PEA  $0000        ;DUMMY ADDRESS - NOT USED SINCE HOWMANY=0
   PEA  $0000
   PEA  $0057        ;ENABLE SRQ OF ADDRESS 7
   LDX  #SND*256+UCTOOL ;SET UP FUNCTION CALL FOR SEND
   JSL  =TOOL.LOCATOR
   BCS  ERROR
```

The following example shows how to make a tool call to change the handler of a ADB device at address 7. It uses the uC tool set function called SEND to transmit 2 bytes to register 3 at address 7:

```
   LDA  #$0207       ;CHANGE DEVICE AT ADDRESS 7 TO HANDLER 2
   STA  DATABUF      ;  INTO DATA BUFFER
   PEA  $0002        ;COUNT OF 2 BYTES TO BE SENT ON ADB (HOWMANY)
   PEA  ^DATABUF     ;HI WORD (BANK) OF DATA BUFFER ADDRESS
   PEA  DATABUF      ;LOW WORD OF DATA BUFFER ADDRESS
   PEA  $00B7        ;TRANSMIT 2 DATA BYTES TO REG.3 OF ADB ADDRESS 7
   LDX  #SND*256+UCTOOL ;SET UP FUNCTION CALL FOR SEND
   JSL  =TOOL.LOCATOR
   BCS  ERROR
```

Here is an example of sending data to an ADB device:

```
DATASND  EQU *
       PEA  $0005    ;5 DATA BYTES (4 data & 1 ADB command)
       PEA  ^DATA    ;ADDRESS OF DATA BUFFER W/ AxBy
       PEA  DATA
       PEA  $004B    ;CMD TO uC (TRANSMIT 4 DATA BYTES)
       LDX  #SND*256+UCTOOL
       JSL  =TOOL.LOCATOR
       BCS  ERROR

DATA     DS  $8A,1,2,3,4  ;1ST BYTE IS COMMAND:
*                  ;   DEVICE @ ADDRESS 8, LISTEN, REG.0
*                    ;OTHER BYTES ARE DATA
```

And finally an example that explains how to poll a device at address 7, register 0 for data:

```
ADBPOLL EQU $D         ;ADB POLL FUNCTION NUMBER
   PEA ^CPLTVC         ;HI WORD (BANK) OF COMPLETION ROUTINE
   PEA  CPLTVC         ;LOW WORD OF COMPLETION ROUTINE
   PEA  $00C7          ;REGISTER 0, ADDRESS 7 (IF REG. 3 THEN $F7)
   LDX  #ADBPOLL*256+UCTOOL ;SETUP FUNCTION CALL FOR POLL
   JSL  =TOOL.LOCATOR
   BCC  OK             ;EVERY THING OK
   CMP  #UCTOOL*256+BUSY ;CHECK IF BUSY ERROR
   BEQ  ADBPOLL        ;POLL AGAIN IF BUSY
   BRA  ERROR
OK  EQU  *
END RTS                ;END
```

COMPLETION VECTOR ROUTINE DESCRIPTION:

All completion routines are called in 8-bit native mode. There are currently two types of completion vectors defined, the ADB Poll and the SRQ List.

ADB Poll & ADB Receive Completion Vector

The ADB Poll Completion Vector Routine grabs Data from Buffer pointed to by address on top of Stack. The first byte in the buffer contains the number of data bytes in the buffer. The 1st data byte received from ADB is the next byte in the buffer, with subsequent data bytes received from ADB stored sequentially in the buffer. The last (nth) byte received is the n+1 byte in the buffer.

```
CPLTVC EQU *        ;COMPLETION VECTOR FOR ADB POLL
   PHD          ;Move direct page onto stack-1
   TSC          ;Stack now has RTL address    (3 bytes)
   TCD                    Old Direct Page (2 bytes)
   LDA  [6]      ;Get length byte from buffer
   BEQ  ENDPOLL  ;Got no data
   TAY          ;Set index to get 1st data byte
LP  LDA  [6],Y    ;Get data byte
   STA  BUF,Y    ;Move to application buffer
   DEY          ;Set index for next data byte
   BPL  LP
ENDPOLL EQU *
   PLD          ;Restore Direct page
   RTL          ;RETURN FROM COMPLETION ROUTINE
```

SRQ List Completion Vector

The SRQ Completion routine is very similar to the routine used by ADB Poll. The only major difference is that an extra return address is on the stack when the routine is called. (These extra 3 bytes are left by the SRQ list handler). SRQ completion routines will find the address to the data buffer 3 bytes into the stack, instead of on the top of the stack.

```
SRQCPLT  EQU *        ;COMPLETION VECTOR FROM SRQ LIST
      PHD           ;SAVE DIRECT PAGE
      TSC           ;MOVE DIRECT PAGE ONTO STACK
      TCD           ;AND DO INDIRECT INDEXED LONG TO
      LDA [9]       ;GET DATA LENGTH (NOT 0)
      TAY
      LDA [9],Y     ;GET.LAST DATA BYTE
      DEY           ;MORE DATA, ETC.
*
ENDABS  PLD         ;RESTORE STACK AND RETURN
      RTL
```

This table is used to show the tool set function protocol. The Input Words/Long are stored on the stack by pushing the first item from the top of the list onto the stack first. (i.e. When the tool locator is called the Command is typically on the top of stack). The DataoutPtr and DatainPtr are pointers to a data structure of bytes. These data structures are shown in more detail in Appendix A.

Command Structure: (FUNCTION NUMBER)		Error:
SEND: (9)	Input Word: HowMany Input Long: DataoutPtr Input Word: Command	$10: Command Not Completed

Send data to the uC. The command and data to be sent to uC is documented in Appendix A&B.

RCV: (A)	Input Word: Howmany Input Long: DatainPtr Input Word: Command	$10: Command Not Completed

Receive data from the uC. The command and data to be received from uC is documented in Appendix A&B.

RDmem: (B)	Input Long: DataoutPtr Input Long: DatainPtr Input Word: Command	$10: Command Not Completed

Used to read a data byte from the uC memory ROM ($1400-$1FFF) or RAM ($0-$5F). The command and data to be received is documented in Appendix A&B.

Reserved
(C)

ADBpoll: (D)	Input Long: Completion Vector Input Word: Command	$10: Command Not Completed $82: Busy (Command Pending)

Receive data from a ADB device. The ADB command byte sent assumes that the command type is Talk, which tells the addressed device to send data to the host (i.e. Talk).

ADBrcv: (E)	Input Word: ADB Command Code Input Long: Completion Vector Input Word: Command	 $10: Command Not Completed $82: Busy (Command Pending)

Receive data from a ADB device. First byte sent is command byte to be sent on ADB, which should include the ADB command type, address, and register. Normally this would only be used instead of the ADBrcv function (D) if the command type was neither a

ADB Listen or ADB Talk command. The command and data to be received is documented in Appendix A&B.

ABSON: NO parameters
(F)
ABSOFF:
(10)
Used to disable/enable automatic polling of an absolute device. Default is to automatically gather data from an absolute device and interpret as absolute mouse positioning.

RDABS: Output Word: On/Off
(11)
Read flag to determine if automatic polling of absolute device is on or off.

SCALE: Input Long: DatainPtr
(12)
Sets up scaling for absolute devices. By predefining a tool call, a generic scaling desk accesory can be written to support almost any size/brand graphics tablet.
Each of these values is stored as a word (16-bits) though the multiply values will only operate on the low 8-bits.

X-Divide
Y-Divide
X-Offset
Y-Offset
X-Multiply
Y-Multiply

RDSCALE: Input Long: DataoutPtr
(13)
Read absolute device scaling values.
Each of these values is stored as a word (16-bits) though the multiply values will only operate on the low 8-bits.

X-Divide
Y-Divide
X-Offset
Y-Offset
X-Multiply
Y-Multiply

SRQPL: Input Long: Completion Vector $10: Command Not Completed
(14) Input Word: ADB Address $83: Device Not Present(@Address)
$84: List Full

This routine adds a device to the SRQ list (if the device exists) so that an application can be notified when this device has data. Whenever an SRQ is generated the system will automatically poll any device in the SRQ list to see if it has data ready. If data is available then it will vector to the completion routine with the data and notify the application.

SRQRMV: Input Word: ADB address $10: Command Not Completed
$82: Busy (Poll Active)

Removes a device from the SRQ list.

CLRSRQTBL: No Parameters

Clears the SRQ list of all entries.

Some other processes are supported in the ADB uC tool set, since certain interrupt conditions can originate from the uC.

Interrupt Handlers:

SRQ - Maintains SRQ List & Pointer. If end of list encountered then cleans up by disabling SRQ of all devices (except the keyboard), then enabling the SRQ of every device in the SRQ list.

ABORT - Attempts SYNCH command: If ignored System death, else RTL and continues. This interrupt will reset many of the defaults, including ADB devices and the control panel. If this error could be fatal to an application then the ABORT vector should be patched into so that it can be detected by the application (and then the application should jump to wherever the old ABORT vector was pointing).

RESPONSE - Reads data then Vectors to Completion Routine. Only a single completion vector can be active at a time. If an application wants to poll many devices sequentially then it should use the completion vector to initiate a poll of the next device.

Appendix A - Commands:

```
               HOW
Function CMD MANY  Single-Chip Microcontroller Command (From system to uC)
---------------------------------------------------------------------
SEND      01  0  ABORT
SEND      02  0  RESET KEYBOARD uC
SEND      03  0  FLUSH KEYBOARD
SEND      10  0  RESET SYSTEM
SEND      40  0  RESET ADB
SEND      5x  0  ENABLE SRQ  (x=ADB address in low nybble)
SEND      7x  0  DISABLE SRQ (x=ADB address in low nybble)
SEND      6x  0  FLUSH BUFFER ON ADB DEVICE (x=ADB address in low nybble)
SEND      04  1  SET MODES
               INPTR <- Input Byte:   Mode
SEND      05  1  CLEAR MODES
               INPTR <- Input Byte:   Mode
SEND      06  3  SET CONFIGURATION
               INPTR <- Input Byte:   ADB adrs. keyboard & mouse
               INPTR <- Input Byte:   Layout/Lang.
               INPTR <- Input Byte:   Repeat Delay/Rate
SEND      07  4  SYNCH
               INPTR <- Input Byte:   Mode
               INPTR <- Input Byte:   ADB adrs. keyboard & mouse
               INPTR <- Input Byte:   Layout/Lang.
               INPTR <- Input Byte:   Repeat Delay/Rate
SEND      08  2  WRITE uC MEMORY
               INPTR <- Input Byte:   Zero Page Memory Address
               INPTR <- Input Byte:   Data
SEND      11  1  SEND ADB KEYCODE
               INPTR <- Input Byte:   Keycode
SEND      4y n+1 TRANSMIT ADB BYTES (y=7+n & n=# of bytes to transmit {n<>0})
               INPTR <- Input Byte:   ADB Command (ADB type,adrs.,reg.)
               INPTR <- Input Bytes:  2-8 data bytes
SEND      8r  2  TRANSMIT 2 ADB BYTES (r=ADB address)
               INPTR <- Input Byte:   1st Data Byte
               INPTR <- Input Byte:   2nd Data Byte
RDmem      09    READ uC MEMORY
               INPTR <- Input Byte:   Low  Memory Address
               INPTR <- Input Byte:   High Memory Address
               OUTPTR <- Output Byte:  Data
RCV      0A  1  READ MODES
               OUTPTR <- Output Byte:  Mode
RCV      0B  3  READ CONFIGURATION
               OUTPTR <- Output Byte:  Repeat Delay/Rate
               OUTPTR <- Output Byte:  Layout/Lang.
               OUTPTR <- Output Byte:  ADB adrs. keyboard & mouse
RCV      0C  1  READ ADB ERROR BYTE
               OUTPTR <- Output Byte:  Error Code
RCV      0D  1  READ VERSION NUMBER
               OUTPTR <- Output Byte:  Version Number
```

APPENDIX B - Single-Chip Microcontroller (SKI) Commands:

COMMANDS TO uC:

```
BIT  76543210
-------------------------------------------------------
   00000000  -
   00000001  ABORT COMMAND
   00000010  RESET KEYBOARD uC
   00000011  FLUSH KEYBOARD

   00000100  SET MODES using next byte as follows:
   00000101  CLR MODES using next byte as follows:

             Bit  Function
             ----------------------
              7   Reset on RESET key only (CONTROL not needed)
              6   Set XOR LOCK-SHIFT mode
              5   Change ADB Keyboard layout to //e layout
              4   Buffer keyboard mode
              3   4X repeat enabled, instead of Dual (2X) repeat
              2   Include Spacebar, Delete key on Dual repeat
              1   Disable Auto-poll of ADB mouse
              0   Disable Auto-poll of ADB keyboard

   00000110  SET CONFIGURATION BYTES using next 3 bytes as follows:
             Byte 1:
               HI Nybble - ADB mouse    address
               LO Nybble - ADB keyboard address

             Byte 2:
               HI Nibble - Char.set (needed for certain langs.)
                       MSB set if keypad '.' swapped with ','
               LO Nybble - Set Keyboard Layout Language
                       LAYOUT/LANG. = CODE:
                       --------------------
                       US      (US) =  0
                       UK      (UK) =  1
                       FRENCH  (FR) =  2
                       DANISH  (DN) =  3
                       SPANISH (SP) =  4
                       ITALIAN (IT) =  5
                       GERMAN  (GR) =  6
                       SWEDISH (SW) =  7
                       DVORAK  (DV) =  8
                       CANADIAN (CN) = 9
```

Byte 3:
  HI Nybble - Set Delay to repeat rate (3 bits)
    0: 1/4 sec.
    1: 1/2 sec.
    2: 3/4 sec.
    3: 1 sec.
    4: NO REPEAT
  LO Nybble - Set Auto-repeat rate (3 bits)
    0: 40 keys/sec
    1: 30 keys/sec
    2: 24 keys/sec
    3: 20 keys/sec
    4: 15 keys/sec
    5: 11 keys/sec
    6: 08 keys/sec
    7: 04 keys/sec

00000111 SYNCH COMMAND

Sets MODES byte (See Command 4 or 5 above) followed by Configuration bytes (Command 6). This command is issued by the system after reset to reset the keyboard. After receiving the command the uC will reset itself back to its internal power up state and then reset ADB devices.

00001000 WRITE uC MEMORY
Send 1 byte address (for RAM) followed by 1 byte of data

00001001 READ uC MEMORY
Send 2 bytes address of uC location (ROM or RAM).
1st byte=low adrs byte & 2nd byte=hi adrs byte(=0 if RAM)

00001010 READ MODES BYTE (See command 4 or 5 above)

00001011 READ CONFIGURATION BYTES - Returned in Data latch:
See Set Configuration for values

Byte 1:
  HI Nybble - ADB mouse address
  LO Nybble - ADB keyboard address

Byte 2:
  HI Nybble - Char.set (needed for certain langs.)
  LO Nybble - Set Keyboard Layout Language

Byte 3:
  HI nibble - Set Delay to repeat rate (3 bits)
  LO nibble - Set Auto-repeat rate (3 bits)

00001100 READ THEN CLEAR FBD ERROR BYTE - Returned in Data latch

00001101 GET VERSION NUMBER - Returned in Data latch
(Also returns PORT R, which is undefined input port on uC, in HI nybble.)

00001110 READ CHARACTER SETS AVAILABLE - Returns # of bytes, then data
This command is used by control panel to determine which character sets are available in the system. This assumes that each uC is paired with a specific mega chip. (Though mega chips may be paired w/ more than one uC). The order that the character sets are returned is important. The first number returned corresponds to the character set 0 in the mega, while the next number is character set 1, etc.

00001111 READ LAYOUTS AVAILABLE - Returns # of bytes, then the data
This command is used by control panel to determine which keyboard layouts are available in the system. Again, like the character sets available command the order that the number are returned is important. The first number returned represents layout 0 in the uC. A predefined table defines which number corresponds to which layout language.

00010000 RESET THE SYSTEM - Pulls the reset line low for 4 ms.

00010001 SEND ADB KEYCODE - Pretend that 2nd byte is ADB keycode
This command can be used to emulate a ADB keyboard, by accepting keycodes from a device and then sending them to the uC to be processed as keystrokes. This command will not process either RESET up or RESET down codes, so they must be trapped out before using this command. This command can be used to watch for key up sequences.

0001---1 -

001----- -

01000000 RESET ADB - Pulls ADB low for 4 ms.
Care must be taken with this command because resetting a ADB keyboard will clear any pending commands including all key up events. This means that if a keystroke is used to launch this command while the key is released, then the key up code will be lost and the key will auto-repeat until another key is pressed. All keys should be up before this command is executed.

01001000 RECEIVE BYTES - Command, w/ address, is in 2nd byte
The system starts by sending a command byte on ADB and then waits for the uC to pass back any data that it receives. Returns bytes in opposite order (n->1).

01001num TRANSMIT num BYTES - Command, w/ address, is in 2nd byte
Note: If num=0 then command is RECEIVE BYTES described above
Else num = # of data bytes-1
The system starts by sending a command followed by between 2 to 8 data bytes (num+1) to the uC, which are to be transmitted over ADB. The command sent will be transmitted directly as the ADB command byte, which is the first byte received after the TRANSMIT num BYTES command.

0101abcd ENABLE SRQ ON ADB DEVICE AT ADDRESS abcd

0110abcd FLUSH BUFFER ON ADB DEVICE AT ADDRESS abcd
This command is dangerous - see RESET ADB description

0111abcd DISABLE SRQ ON ADB DEVICE AT ADDRESS abcd
This command may be dangerous. If data is pending when this command is executed then the pending data may be lost. For example if SRQ is disabled on the ADB keyboard then all key up codes may be lost. Also see RESET ADB description

10xyabcd TRANSMIT 2 BYTES:
Address - abcd
Register- xy

Assumes a two byte transfer of data using the ADB Listen command.

11xyabcd Poll ADB device:
Address - abcd
Register- xy

This command is used to get data from a specific device. It uses the ADB Talk command then waits for the device to either send back data or timeout. The uC waits until all data has been received then responds back to the system with a status byte which indicates the number of bytes received followed by the data. It returns the bytes in opposite order than received on ADB (n->1).

* All commands which require more than a 1 byte transfer, will automatically timeout in 10 ms. if there is no response, except for the SYNCH cmd which may require 20 ms. to process the ADB address byte.