Code Secrets of Wolfenstein 3D IIgs

Eric Shepherd
Fast Screen Refresh with “PEI Slamming”

Or, “Dirty Tricks with the Direct Page”
IIGs Features We Can Abuse

- Super high-resolution graphics shadowing
- Bank $01 stack and direct page
- Relocatable stack and direct page pointers
Super High-Resolution Shadowing

The Apple IIgs has only one SHR graphics page, in bank $E1, from $2000-$9FFF.
Super High-Resolution Shadowing

But you can draw graphics into bank $01 in the same memory range...
So that when you draw into bank $01, the data is “shadowed” into bank $E1 by the Apple IIgs hardware.
Super High-Resolution Shadowing

Why is this helpful?

- Banks $00$ and $01$ are “fast” memory, while $E0$ and $E1$ are “slow” memory.
Writing into Bank $01
Even Faster

- The Direct Page and Stack are special areas of memory used for special purposes.
- They have special opcodes that are faster for moving data.
- They’re usually in bank $00...
Writing into Bank $01
Even Faster

...but you can move
them to bank $01!
Writing into Bank $01
Even Faster

...but you can move them to bank $01!
Writing into Bank $01 Even Faster

Softswitches

- $C005 and $C003 enable writing and reading to bank $01 as DP and stack
- $C004 and $C002 disable writing and reading from bank $01 as DP and stack
Relocating the Stack and DP Pointers

As usual, you can use the TCD (Transfer Accumulator to Direct Page Pointer) and TCS (Transfer Accumulator to Stack Pointer) opcodes to relocate the direct page and stack.

This works even when the DP and stack are in bank $01.
Putting It All Together

Step 1: Turn off shadowing

SEP #$20
LDA >$E0C035
ORA #$08
STA >$E0C035
REP #$20
Putting It All Together

Step 2: Draw your graphics, treating bank $01 as if it were bank $E1.
Putting It All Together

Step 3: Turn shadowing back on.

- SEP #$20
- LDA >$E0C035
- AND #$F7
- STA >$E0C035
- REP #$20

Bank $01

$2000

$A000
Putting It All Together

Step 4: Save entry DP and stack, disable interrupts, and switch to bank $01 stack and direct pages.

```asm
! tdc
sta EntryDP
! tsc
sta EntryStack
! sei
! shortm
sta >$00C005
sta >$00C003
! longm
```
Putting It All Together

Why disable interrupts?
Putting It All Together

Why disable interrupts?

Because if an interrupt happens while we’ve moved the direct page and stack into a strange place, the system will probably crash.
Putting It All Together

Step 5: Point the Direct Page Pointer at $2000, the start of SHR memory.

LDA #$2000
TCD

Bank $01
$2000
$A000
Putting It All Together

Step 5: Point the Direct Page Pointer at \$2000, the start of SHR memory.

LDA \#\$2000
TCD
Putting It All Together

Step 6: Point the Stack Pointer at $20FF, the top of the first page of the SHR buffer.

CLC
ADC #$00FF
TCS
Putting It All Together

Step 6: Point the Stack Pointer at $20FF, the top of the first page of the SHR buffer.

CLC
ADC #$00FF
TCS
Putting It All Together

Step 7: Copy a page of graphics data on top of itself fast.

Why? Because this will cause the hardware to shadow it over to bank $E1.
How PEI Slamming Works

PEI (Push Effective Indirect) fetches a word from the direct page and pushes it onto the stack.
How PEI Slamming Works

-$2000/$2001$

- The stack starts at $20FF$ and works backward toward $2000$.

- The direct page starts at $2000$ and works forward toward $20FF$. 
How PEI Slamming Works

PEI $FE

This pushes the word at offset $FE ($20FE-$20FF) on the direct page onto the stack, which puts it at the same spot!
How PEI Slamming Works

**PEI $FE**

This pushes the word at offset $FE ($20FE-$20FF) on the direct page onto the stack, which puts it at the same spot!
How PEI Slamming Works

PEI $FE

This takes just 6 cycles (and two bytes of code) to refresh those two bytes of video to the screen.
How PEI Slamming Works

$2000/$2001

$20FE-$20FF

Stack

Direct Page

PEI $FE

This takes just 6 cycles (and two bytes of code) to refresh those two bytes of video to the screen.
How PEI Slamming Works

Do 128 PEIs in a row to copy the entire 256-byte page.
How PEI Slamming Works

Do 128 PEIs in a row to copy the entire 256-byte page.
Putting It All Together

Step 8: Keep moving the DP and stack pointers and copying another page until you reach $9D00 (or $A000 if you need to copy palettes and scan control bytes).
Putting It All Together

But periodically, you need to move the DP and stack back to bank $00 and re-enable interrupts to let MIDI Synth, GS/OS, and so forth keep running normally.
Let Those Interrupts Run

Enabling Interrupts

shortm
sta >$00C004
sta >$00C002
longm
lda EntryStack
tcs
lda EntryDP
tcd
cli

Disabling Interrupts

sei
shortm
sta >$00C005
sta >$00C003
longm
The End Result
Reading Multiple Keys Down at Once

Or, “Abusing the ADB for Fun and Profit... Well, Mostly Fun”
Things to Note about ADB

- Apple Desktop Bus
- Transmits packets describing state changes of connected devices
- You can hook in at a low level to be informed when the state changes
Intercepting Low-Level Keyboard Events

- Set up an array with the state of every key on the keyboard
- Watch for changes to key states, and record them in the array
Sending an ADB Command

CallSendInfo: A routine that sends X bytes of data using ADB command code Y.

```
CallSendInfo  STA >ADBTemp
  PHX
  PEA ADBTemp|-16
  PEA ADBTemp
  PHY
  _SendInfo
    RTS
ADBTemp       DS 6
```
Installing an SRQ Completion Routine

Step 1: Zero the key state array

```
KeyArray DS 128
Clear LDX #128-2
     STZ KeyArray,X
     DEX
     DEX
     BPL Clear
```
Installing an SRQ Completion Routine

Step 2: Disable ADB autopolling.

LDX #1
LDY #setModes
LDA #1
JSR CallSendInfo
Installing an SRQ Completion Routine

Step 3: Install the SRQ completion routine by passing a pointer to our completion routine and the ADB device ID (2 for a keyboard) to the SRQPoll ADB Tool Set call.

PEA SRQCompRoutine|-16
PEA SRQCompRoutine
PEA $0002
_SRQPoll
Handling ADB Events

Step 1: Write the SRQCompRoutine code to receive events from the ADB. After it sets up its bank and DP as needed, it needs to look to see if data has arrived. A pointer to the received data is on the stack, at offset DataPtr.

```
LDA [DataPtr]           ; # bytes?
BEQ SRExit              ; No data
```
Handling ADB Events

Step 2: Fetch the ADB data out of the data buffer and preprocess it. We have to check

```
REP #$30
LDY #1
LDA [DataPtr],Y
TAY               ;Save a copy
AND #$7F7F
CMP #$7F7F        ;Reset key?
BEQ SRSpecial     ;Yes, handle
```
Handling ADB Events

Step 3: Pull the two ADB data bytes out.

TYA               ;Get it back
AND #$FF00        ;First byte
XBA               ;Swap to LOB
TAX               ;Save in X
TYA
AND #$00FF        ;Second byte
BRA SRMerge1
Handling ADB Events

Step 4: Handle the reset key if need be.

SRSpecial
TYA
LDX #$00FF ; Invalid

SRMerge1
PHX ; Save 2nd
JSR ProcessReset
Handling ADB Events

Step 5: Update the key states.

JSR PostIt
PLX ; Get 2nd
PHA ; Save new #1
TXA
JSR PostIt
PLX
Handling ADB Events

Step 6: Forward the keys to the ADB microcontroller.

TXA ;1st byte
JSR PassADBKeyIfOK
PLA ;2nd byte
JSR PassADBKeyIfOK
Updating the Key State Array

- Set the key’s entry if down, clear it if up.

<table>
<thead>
<tr>
<th>Assembler Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHA</td>
<td>Save key</td>
</tr>
<tr>
<td>CMP #$80</td>
<td>Set/clear c</td>
</tr>
<tr>
<td>AND #$7F</td>
<td>Keycode idx</td>
</tr>
<tr>
<td>TAX</td>
<td></td>
</tr>
<tr>
<td>LDA #$00</td>
<td>Key state</td>
</tr>
<tr>
<td>ROL</td>
<td>0 for keyup</td>
</tr>
<tr>
<td>EOR #$01</td>
<td></td>
</tr>
<tr>
<td>STA &gt;KeyArray,X</td>
<td></td>
</tr>
<tr>
<td>PLA</td>
<td></td>
</tr>
<tr>
<td>RTS</td>
<td></td>
</tr>
</tbody>
</table>
Sending the Key to ADB

Pass keys to the ADB when appropriate.

PassADBKeyIfOK
  CMP #$00E0        ; Pfx code?
  BGE PAExit
  CMP #$0036        ; Spec. case?
  BLT PASendADB
  CMP #$003B         
  BGE PASendADB
  TAX               ; Code to X
  SEC
  SBC #$0036        ; Table index
  ASL
Sending the Key to ADB

Pass keys to the ADB when appropriate.

TAY ;Idx to Y
JSR GetModKeyReg ;Get keymods
AND KeyModTbl,Y ;Down?
BNE PAExit ;Yes
TXA
LDX #$0001
LDY #keyCode
JSR CalISendInfo
RTS
Reading the Keyboard

Now your code can check the state of keys.

```java
if (KeyArray[keyLeft] || KeyArray[0x3B]) {
    /* left arrow or keypad 4 is down */
}

if (KeyArray[keyUp] || KeyArray[0x2B]) {
    /* up arrow or keypad 8 is down */
}
```
Reading the Keyboard

Your code can detect multiple keys being held down at the same time, enabling much more powerful player controls.

See page 3-22 of the Apple IIgs Toolbox Reference, Volume 1 for the ADB key codes (which are different from ASCII codes).

Read the ADB chapters in that and in the Firmware Reference.
Handling System Reset

The ProcessReset routine should look to see if it’s a key up event on key code $7F7F.

If it is, and the Control and Command keys are also down, the resetSys command should be sent to the ADB, to cause the system to reboot.
Things to Add

- When TOBRAMSETUP is called, the SRQ completion routine is disabled. You may want to use the GetVector and SetVector Misc Tool Set calls to intercept this call so you can re-enable your completion routine.

- Don’t forget to remove your patch to this vector when your application quits!
Q & A

Or, “Huh? That didn’t make any sense.”