PAM PACKAGE

ATARI

inter-office memo

TO: All PAM Programmers

FROM: Rob Zdybel

DATE: October 28, 1981

SUBJECT: PAM Controllers

A PAM controller consists of:

1) A fifteen key keypad
(3 keys will have standardized functions)

2) An analog joystick
(2 pot lines)

3) A ' hard ' trigger button
(comes in on hardware trigger line)

4) A ' soft ' trigger button
(actually a key, but generates its own, unique IRQ)

Each PAM system may have up to 4 controllers attached.

Individual keypads are selected by writing to D1 & DØ of the CONSOL Register. This also selects which soft-trigger will be monitored.

All pot lines and all hard trigger lines are always available.

RZ/db

inter-office memo

ATARI

TO: All PAM Programmers

FROM: Rob Zdybel

DATE: October 30, 1981

SUBJECT: PAM O.S. Partial Description

The PAM O.S. consists of an Interrupt Handler Facility and a Power-up Monitor.

I) <u>Interrupt Handler:</u>

The hardware interrupt vectors are embedded in O.S. ROM. The O.S. determines the interrupt cause, resets it, and transfers control thru the associated O.S. vector.

A) <u>NMI's:</u>

No registers are saved on the stack.

I) VVBLKI= VBLANK (Immediate) Vector.
(this vector is initialized to SYSVBL).

II) VSDLST= Display List Vector.
(this vector is initialized to O.S. DLI Service).

B) <u>IRQ Immediate:</u>

No registers are saved on the stack.
VIMIRQ= IRQ (Immediate) Vector.
(this vector is initialized to SYSIRQ).

Note: IRQ cause is <u>NOT</u> cleard

C) <u>IRQ Deferred (SYSIRQ):</u>

The Accumulator is saved on the stack.
(IRQ's in priority order)

I) VSERIN= Serial Input Ready Vector

II) VTRIGR= Soft-Trigger Vector

III) VKYBDI= Keyboard Immediate Vector
(this vector is initialized to SYSKBD)

IV) VSEROR= Serial Output Ready Vector

V) VSEROC= Serial Output Complete Vector

VI) VTIMR1= Pokey Timer #1 Vector

VII) VTIMR2= Pokey Timer #2 Vector

VIII) VTIMR4= Pokey Timer Vector

IX) VBRKOP= BRK Opcode Vector
(Note: X register is also saved on the stack).

D) Vblank Deferred (SYSVBL):

All registers are saved on stack.

Normally control passes thru VVBLKD when System Vblank is complete. However, if CRITIC is non-zero or the processor I-bit was set, then System Vblank is aborted and control is NOT passed thru VVBLKD.
(this vector is initialized to exit).

E) System Keyboard Handling (SYSKBD):

All registers are saved on stack.

The Keycode is fetched from POKEY, converted to Ø-F and the result left in the accumulator. Control then passes thru VKYBDF.
(this vector is initialized to exit).

II) Monitor:

At Power-up time the O.S. zeros all hardware addresses and clears zero page. SDMCTL, PRIOR, CHBASE, SKCTL and NMIEN are re-initialized to non-zero values. The Copyright message is displayed and control is passed to the cartridge. An O.S. DLI is providing the rainbow logo when the cartridge receives control.

III) O.S. RAM Allocation:

A) Zero-Page RAM from Ø to 1F is reserved for O.S. shadows. If SYSVBL is not allowed to run, then O.S. Zero-Page RAM required consists of only byte Ø (POKMSK). If SYSVBL and SYSIRQ are both disabled, then no Zero-Page is required for the O.S.

B) 1ØØ-Page is reserved for the stack.

C) 2ØØ-Page from 2ØØ to 21F is reserved for O.S. vectors. If SYSIRQ is not allowed to run, then only 2ØØ to 2Ø7 is required for O.S. vectors.

IV) O.S. Linkage:

Cartridges communicate w/the O.S. via Shadow Registers, Vector-Page, and the Cartridge Communication Area.

A) Shadow bytes are all on Zero-Page:

I) POKMSK= shadow IRQEN
II) SDLSTL, SDLSTH= shadow DLIST ptr.
III) SDMCTL= shadow DMACTL
IV) PCOLRØ-PCOLR3= shadow Player Colors

V) COLORØ-COLOR4= shadow Memory Map Colors
VI) PADDLØ-PADDL7= shadow Pot Readings

B) Vectors are all on 2ØØ-Page. The associated interrupt must be disabled while a vector is being altered.

C) The Cartridge Communication Area is the last 24 (decimal) bytes of each cartridge, starting at BFE8:

I) 20 bytes of cartridge name information (in ASCII+4Ø display mode).
II) 1 byte, Copyright Decade information (in ASCII+4Ø display mode).
III) 1 byte, Copyright Year information (in ASCII+4Ø display mode).

IV) 1 word, Cartridge start address. Control will be passed indirect thru BFFE to your cartridge.

D) ROM vs. Blackbox O.S.:

A) The O.S. Startup address for Blackboxes is FØØØ.
B) If you wish your cartridge org'ed for ROM, define the label, ZZZROM. If this label is undefined for PAMEQ, it will produce a Blackbox version of your cartridge.

This is an interrim document (hell, it's an interrim O.S.!) and subject to frivolous change at any time. The best source for O.S. information is, as always, an O.S. source listing...

RZ/db

cc: M. Ebertin

ATARI®

inter-office memo

TO: ALL PAM PROGRAMMERS

FROM: ROB ZDYBEL

DATE: NOVEMBER 13, 1981

SUBJECT: PRIORITY OF PLAYER5

Player5 (the 4 missiles used in combination) does not have the priority indicated by your hardware manual. Instead, the color for Player5 is drawn from COLPF3 and OR'ed with the color-luminance of the playfield which Player5 is over-lapping.

This means that a black (ØØ) Player5 (also PF3) will seem to have lower priority than all playfield and a white (FF) Player5 will seem to have higher priority than all playfield. Colors can be (carefully) chosen to provide almost any intermediate priority.

RZ/db

inter-office memo

TO: PAM Programmers

FROM: Rob Zdybel

DATE: December 3, 1981

SUBJECT: PAM KEYBOARDS

1) How to read a Keyboard

A) The Process:

1) Point the desired vector to the proper area of your own program.

a) VKYBDI= Key Depressed IRQ Vector (Immediate) Control is transferred thru this vector by the PAM O.S. IRQ Handler when a Keyboard interrupt is received.

b) VKYBDF= Key Depressed IRQ Vector (Defferred) Control is transferred thru this vector by the PAM O.S. System Keyboard Handler.

2) Enable keyboard interrupt bit in both POKMSK and IRQEN. Enable IRQ-level interrupts on the processor.

3) Enable POKEY keyscan & disable debounce by writing 2 to bits D1 & D0 of SKCTL.

4) Select desired keyboard by writing to bits D1 & D0 of CONSOL (0= Player 1 Keyboard... 3= Player 4 Keyboard).

B) The Theory:

POKEY automatically scans the keyboard when bit D1 of SKCTL is true. There is no way to scan the keyboard under software control. POKEY Keyscan (see diagram) is driven by a 6-bit counter which is clocked at the HSYNC rate. PAM keyboards have only 16 keys, therefore, only 4 of the 6 counter lines are actually used by the PAM hardware (the hi & lo-order lines being unused).

Debounce (bit D0 of SKCTL) must be disabled or no keyboard interrupts will be generated (see diagram).

Bits D1 & D0 of CONSOL are run to MUXs which select which of the 4 keyboards will be scanned for keyboard and soft-trigger depression.

2) Key Values & Pre-Defined Keys

The PAM O.S. Keyboard Handler will accept POKEY keyboard IRQs and translate the value in KBCODE into 0-F. The keystroke values are defined as:

C	D	E
1	2	3
4	5	6
7	8	9
A	0	B

NOTE: F should NEVER occur

Keys with special pre-defined functions (see PAM Standards) are:

A= OPTION key for Select/Option Schemes
   CLEAR key for string input
B= SELECT key for Select/Option Schemes
   ENTER key for string input
C= START key
D= PAUSE key
E= RESET key

3) <u>Keyboard Problems</u>

A) <u>Repeat Rate:</u>
Normally, a single key being held down wil generate roughly seven keyboard IRQs per Frame. Thus, a single user keystroke will generate many IRQ requests. In this sense, it will be neccessary to provide some software debounce capability (specific suggestions/routines are forthcoming).

B) <u>Multi-key Depression:</u>
When more than one key is depressed simultaneously, multiple key readings are returned. No 'ghosting' has been observed, although, it may occur with proper key combos. Each key held down will generate its own IRQ several times per Frame. The relative frequency of each key is highly dependent on both the key in question and the other key(s) depressed.

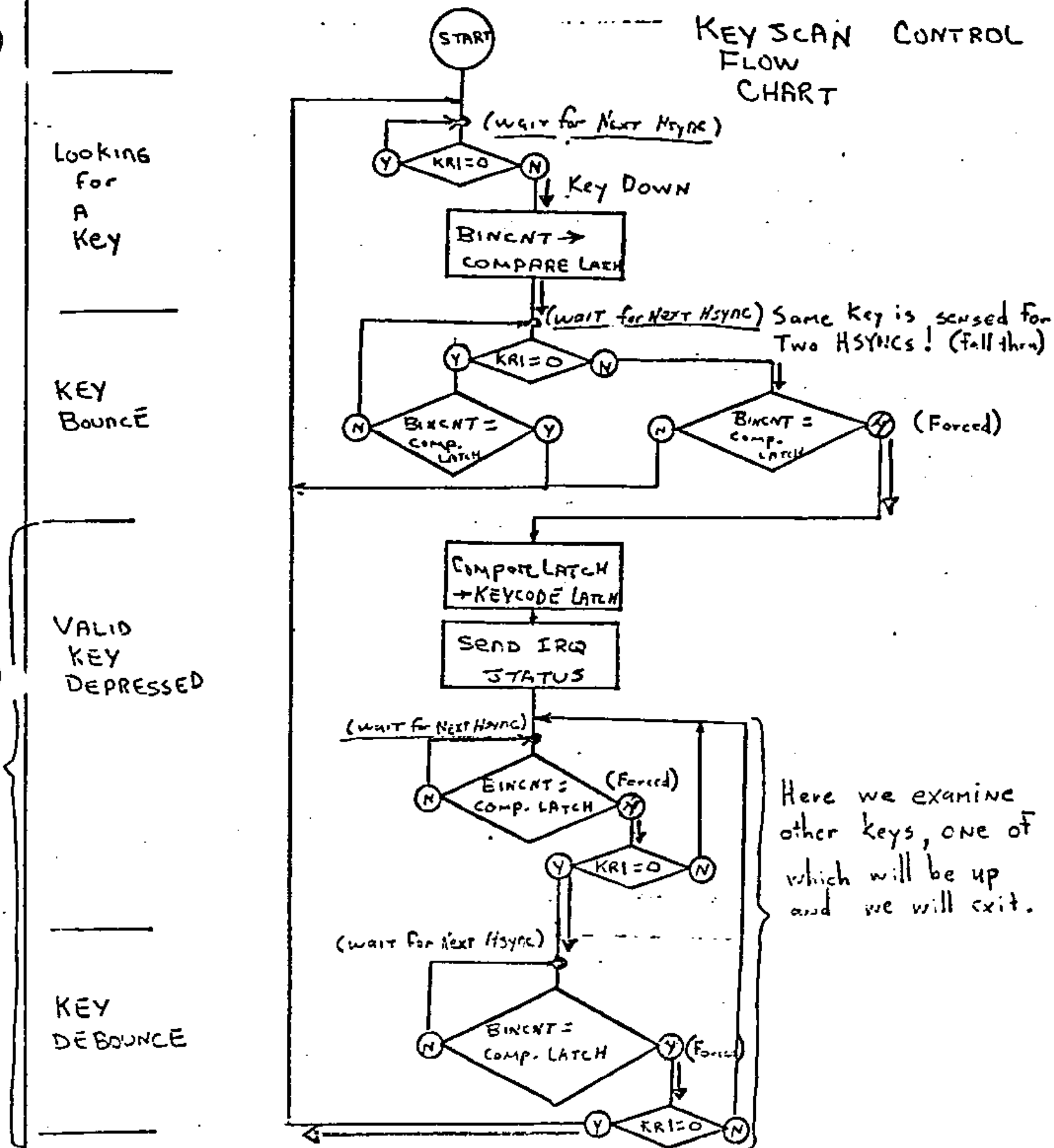
4) <u>SKSTAT:</u>
Two keyboard status bits are provided in the POKEY keyboard status register (SKSTAT). Their utility is questionable, but they are:

A) <u>Keyboard Overrun (D5):</u>
This bit is reset when a second keyboard interrupt is generated before the first keyboard interrupt has been cleared.

B) <u>Last Key Still Depressed (D2):</u>
Not what the name implies, this bit is controlled by the current state of the POKEY keyboard scan algorithm (see diagram). This bit is reset between keycode latching & the end of the POKEY debounce loop (see diagram).

RZ/db

cc: Ron Stephens



Ⓨ ≡ IF DEBJUNE Disable is in effect THEN BINCNT = Compare LATCH

ATARI®

inter-office memo

TO: All Programmers

FROM: Rob Zdybel

DATE: December 21, 1981

SUBJECT: PAM Controllers II

I would like to discuss several methods of reading the PAM analog joysticks and keyboard. These are not required algorithms, nor do I guarantee that they are the best possible algorithms, these represent only one implementation of several common tasks. Please feel free to criticise and suggest improvements to these routines.

The Self-Centering Pot: As you may know different POKEYs may return a wide range of values with different pots. To compensate for this, we must determine the mid-range value for each Pot-POKEY pair in software.

This algorithm assumes (justifiably) that the mid-range value is the average of the maximum value and the minimum value returned by the Pot-POKEY pair.

```
(Note:  MAX, MIN and MID must be initialized        )
                LDA     POT
                CMP     MAX
                BCC     SKIPMAX
                STA     MAX                 ; New Maximum Value
                BCS     COMPMID
SKIPMAX:        CMP     MIN
                BCS     EXIT
                STA     MIN                 ; New Minimum Value
COMPMID:        LDA     MAX                 ; Compute the Mid-point
                CLC
                ADC     MIN
                ROR
                STA     MID                 ; New Mid-Range Value
EXIT:           ...
```

Problems: The Pot will exhibit a very limited range for the first several frames.

<u>Absolute Positioning:</u> In this technique the position of the cursor (or whatever) is derived directly from the pot-reading. Thus, when the stick is Full-left the cursor is at the left edge of the screen, Full-right is the right-edge of the screen, etc.

(Note: Requires a center value, probably from self-centering pot routine)

```
                 LDA     POT
                 SEC
                 SBC     MID                ; A= Delta from middle
                 CLC
                 ADC     # ScreenCenter     ; Scale to screen
                 STA     POSN
```

<u>Velocity Positioning:</u> This method uses the position of the joystick (relative to center) to determine the vector (magnitude & direction) of the cursor (or whatever) motion. Thus, when the stick is Full-left the cursor moves leftward as rapidly as possible, stick in the center results in no cursor movement, etc.

(Note: Requires a center value, probably from self-centering pot routine)

```
                 LDA     POT
                 SEC
                 SBC     MID                ; Find Delta from center
                 PHP                        ; save direction
                 BCS     Positive Result
                 EOR     # OFF
                 ADC     # 1                ; Negative, invert result
Positive-Result: LSR
                 LSR
                 LSR                        ;speed magnitude (0 to 10)
                 PLP                        ; restore direction
                 BCS     Exit
                 EOR     # OFF              ; re-invert negative result
                 ADC     # 1
  Exit:          CLC
                 ADC     POSN
                 STA     POSN               ; update cursor position
```

<u>Binary Positioning:</u> This amounts to turning an analog joystick back into a digital joystick. This must be done with care. If the null-zone (that is, not left and not right) is too small, the user will have a difficult time stopping cursor motion. If the null zone is too large, the stick will have a 'dead' feel and exhibit an irritating delay between move-left and move-right. This routine could also be implemented with a modified velocity-positioning algorithm.

(Note: Requires a center value, probably from self-centering pot routine)

```
                    LDA    POT
                    SEC                        ;find Delta from center
                    PHP
                    BCS    Positive-Result
                    EOR    # OFF               ;invert result
                    ADC    # 1
Positive Result:    CMP    # Min-value        ; check against minimum swing
                    BCS    OK-value
                    PLP
                    JMP    Exit                ; clear stack
OK Value:           PLP
                    BCS    Was Positive
                    DEC    POSN                ; negative, decrement
                    JMP    JOIN
Was Positive:       INC    POSN
  JOIN:             ...
```

<u>Pot Jitter</u>: For a variety of reasons (RFI, thermal, etc.) the reading returned by a given pot may vary (usually + or - 1) even though the pot has not been physically moved!! This problem must be addressed in parallel with the method of reading the controller (please note that none of the preceding algorithms take jitter into account!). There are many possibilities for resolving the jitter problem, some of them are:

A) The Running Average: This routine combines the current pot reading the average of all preceding pot readings.

```
LDA    POT
ADC    OLDPOT
ROR
STA    OLDPOT
```

Advantages: 1) Routine is short, takes little RAM
2) Exhibits good damping for + or - 1 noise

Problems: 1) May take several frames to catch up to a major change. Stick responds sluggishly, a big problem.
2) Any authentic change of pot-value that is less than two has no effect!

B) The Instantaneous Average: Here the result depends only on the last 2 pot values read

```
        LDA     POT
        CLC
        ADC     OLDPOT
        ROR
        STA     AVGPOT
        LDA     POT
        STA     OLDPOT
```

Advantages: 1) Always perceives pot change within 2 frames.
2) good resistance to + or - 1 noise (unless the frequency of the noise is less than 30Hz, e.x. 31,31,32,32,31,31,...)

Problems: 1) Requires more RAM, ROM and time
2) "long-term" noise (duration > = to 2 frames) will defeat averaging

C) The Absolute Filter: With this technique, we simply reject all pot changes which are less than a specific threshold (the Filter value).

```
                  LDA     POT
                  SEC
                  SBC     OLDPOT             ; Find delta from last reading
                  BCS     Positive-Result
                  EOR     # OFF
                  ADC     # 1                ; invert result if negative
Positive-Result:  CMP     # Filter           ; test against Filter threshold
                  BCC     No change
                  LDA     POT
                  STA     OLDPOT
   No Change:     .
                  .
                  .
```

Advantages: 1) Unlimited noise rejection (dependent on filter value)
2) Takes only a little RAM

Problems: 1) Will not respond to authentic input if it is less than the filter value.

Key Debounce: POKEY will return about 7 IRQ's per frame when a key is held down. When multiple keys are depressed, all key values (no 'ghosting' has been observed yet) are returned in a peculiar mix (Ex. keys A,B & C held down might return "CACBCCBACC...") Our goal is to:

1) Detect first strike of a key
2) Handle Multi-key Depression/Rollover
3) Do Auto-Repeat

The main problem is that POKEY won't tell when the keyboard is clear, the secondary problem is the bizarre distribution of returned key values in the multi-key depressed case. Some possible solutions are:

A) Tung Method: Since POKEY IRQ's are latched, we can collect them at the POKEY end and ignore them at the CPU end until we are ready. Every 256 or so frames we enable IRQ's on the processor and immediately disable them.
This single instruction 'window' allows just one keyboard IRQ every 4 seconds or so.

Advantages: 1) Does Auto-Repeat trivially

Problems: 1) Incompatible with use of any other IRQ's
2) Usually slow to detect first key stroke
3) May have trouble with multiple keys.

B) Counter Method: Uses a counter (incremented at VBLANK time) to count the # of frames since the last keyboard IRQ.

```
@ VBLANK:  LDA   CTR          @ KEYBOARD SERVICE:  LDA   CTR
           BMI   SKIP                              CMP   # MIN-FRAMES
           INC   CTR                               BCC   EXIT
                                                    :
  SKIP:     :                      EXIT:           LDA   #0
                                                   STA   CTR
```

Advantages: 1) Detects first key stroke

Problems: 1) Won't Repeat
2) Doesn't handle multiple key case

C) Counter Method Two: Like the Counter-Method except that it would utilize a 15- byte array of counters. One counter (as in Counter-Method) for each key on the keyboard.

Advantages: 1) Detects first key stroke
2) Detects first of multiple keystrokes (Rollover)

Problems: 1) Doesn't Repeat
2) RAM intensive

D) Counter Method Three: Like Counter-Method-Two except it utilizes 2 one-byte counters for each key. One counter would be for implementing Counter-Method-Two. The second counter would be incremented every keyboard IRQ that the key was detected, it would be cleared with no key had been detected for MIN-FRAMES frames, thus providing repeat.

Advantages: A) detects first key-stroke
B) Does Repeat (even with multiple keys down)
C) Handles Rollover

Problems: A) very RAM intensive

RZ/db

cc: Dave Remson
Pete Gerrard
Michel Ebertin

Inter Office Memo

ATARI®

Consumer Electronics Division

To: All Programmers

From: Rob Zdybel

Subject: PAM O.S. Revision

Date: 1-18-82

The newest version of the PAM O.S. is now available.

System Vblank no longer monitors the state of the I-bit of the processor status word. This means that if you have time-critical or interrupt-driven routines which may conflict with system Vblank it is necessary to use the CRITIC flag or NMIEN to disable system Vblank. For reasons that should be obvious, INC and DEC instructions are recommended for changing the state of CRITIC flag.

Remember the system Vblank routine increments the real-time clock and then checks the CRITIC flag, if the CRITIC flag is set (non-zero) no further processing occurs and an RTI is executed.

This revision should simplify the job of handling keyboard IRQ's. For those who require a faster or more intelligent Vblank handler, it is strongly recommended that you substitute a custom Vblank-Immediate routine of your own devising. Never attempt to "share" O.S. code as it may move at any time.

Should you have any questions regarding details of the revision, a source version of the O.S. is available on SY:PAMOS.MAC.

RZ/db

Inter Office Memo

ATARI®

Consumer Electronics Division

To: All PAM Programmers

From: Rob Zdybel

Subject: Late Change

Date: 2-8-82

There has been a recent revision of the PAM hardware which will probably entail a revision of your cartridge.

Originally, bit D2 of the CONSOL register was to be a dedicated trackball calibrate signal allowing software determination of the trackball rest value. Unfortunately, due to the limited number of pins available at the connector interface, this signal now doubles as the reference voltage for the analog joystick.

Writing a zero to bit D2 of the CONSOL register will now put the trackball in calibrate mode AND DISABLE ANALOG JOYSTICKS!

Writing a one to bit D2 of the CONSOL register will put both types of controllers into operation mode.

REMEMBER: bit D2 of CONSOL should be set to 1 for normal operation of controllers.

This change will be reflected by all development systems as soon as possible.

RZ/db

inter-office memo

TO: All PAM Programmers

FROM: Rob Zdybel

DATE: February 19, 1982

SUBJECT: "PAM Standard"

NOTE: The following are guidelines only. Exceptions can be made in special cases as necessary.

1) There are 3 keys on each keypad with special functions to be defined in software. The keys and their functions are:

   RESET- Puts game in Select Mode. Reset should not clear existing options. (For more on Select Mode, see below.)

   START- Starts or restarts currently-selected game.

   PAUSE- Toggle to stop/continue game. PAUSE should freeze current game state exactly, except that it should go into attract mode after 9 minutes (see below). (PAUSE may need to be deactivated during some parts of games, such as the middle of a football play.)

NOTE: Player 1 must have full RESET, START & PAUSE capability.

2) ATRACT Mode- All games should enter color attract after 9 minutes. The O.S. handles color cycling automatically during SYSVBL. Cartridges which use the O.S. should zero ATRACT each frame while the game is actively playing. Cartridges which directly use the color hardware registers or disable SYSVBL must provide their own attract function.

3) Select Mode- Entered via RESET and at Power-up. Currently-active options should be indicated. Allows selection of game number, options, etc. There are 2 options as to how to implement this:

   a) Separate Menu Display- There should be a separate menu display. Menu items should be individually selectable. Screen item values should be determined either by picking them from the screen display via cursor or by entering data via keypad or joystick using "enter" and "clear" functions to be implemented via the keypad keys immediately to the left and right of the "0" key respectively.

b) Show select options over game display - each key has an option attached to it (i.e. - game number, difficulty, etc) which can be cycled by depressing that key. If the number of options is large, the user must be able to directly select the desired option number via keyboard.

4) Power Up- Unit should power up in Select Mode in game #1.

5) End Game- At end of current game, results of game (final screen, scores, etc.) as well as currently-active options should be indicated in some manner.

6) Controllers should be assigned left to right (i.e. - player 1 is left controller).

7) All games must have 1-player versions as a minimum.

8) The bonzo version (if any) must not be game #1.

9) PAL- The primary PAL difference is that PAL frames occur at 50HZ instead of 60HZ for NTSC. For those carts which need to adjust for the difference in frame timing, a register (PAL=D014) is available in the GTIA. This location will read "0F" for NTSC and "01" for PAL. Use this as needed to select PAL or NTSC speed tables for your moving objects.

In addition, PAL ANTIC provides 50 additional lines of Vertical Blank. The typical display list uses BLANK instructions at the top of the screen and a JUMPWT (Jump & Wait for Vertical Blank) at the bottom of the screen. These instructions will display background color on the screen. (The programmer may wish to add more playfield. Use the PAL register to select or modify the display list if desired.)

RZ/db

inter-office memo

ATARI

TO: ALL PAM PROGRAMMERS

FROM: Rob Z.

DATE: February 26, 1981

SUBJECT: Trakball Controller

The long awaited PAM trakball has arrived and all development systems have been modified to accept it.

Reading the trakball is a simple two-step procedure:

(1) "calibrate" the trakball
(2) read pots normally and treat as you would a velocity stick

The trakball is placed in calibrate mode by writing a zero to bit D2 of CONSOL. When in calibrate mode, the trakball will return the reading it would normally return if the ball were not moving. This will allow you to determine the "stand-still" value for the trakball (velocity will be relative to this value). Hardware recommends waiting 10 to 20 frames after entering calibrate mode to allow large capacitors in the trakball to settle before attempting to read the stand-still value. Don't forget to return the machine to normal operating mode after calibration is complete!

Trakball motion is implemented by use of a velocity positioning method. Please see the memo "PAM Controllers II" for details of the procedure and a suggested algorithm. Use the "stand-still" value of the trakball as your center value.

I recommend that you calibrate the trakball as frequently (ex. Missile Command calibrates at the start of each wave) as is reasonable during the game. One calibration at power-up may not yield acceptable results due to jitter.

All June '82 releases must use the analog joystick, but trakball may be included as an optional controller. This option could be directly selected via the keypad much like any other option. Alternatively, it is possible to determine in software which controller is being used. Remember that calibrate mode will cause the trakball to read at it's "stand-still" value (69$\pm$30 Hex). However, this will disable an analog joystick causing it to return a value of E4. Thus, if calibrate mode returns an E4 value you have either an analog joystick or an empty port!

MEMO TO ALL PROGRAMMERS

Finally, you might wish to put all players on port one when in trakball mode (simply because the trakball will probably be fairly expensive). Multiple players would alternate turns by passing the trakball.

RZ:p

ATARI

inter-office memo

TO: All PAM Programmers

FROM: Rob Zdybel

DATE: March 16, 1982

SUBJECT: Important changes

The most recent revision of the PAM O.S. is always available to you via the command "DLOS". Use this command instead of keeping a copy of the PAM O.S. in your own file space. This way you are always assured of using the most recent revision.

IMPORTANT!! Be sure to test your cart using the DLOS O.S. before you release it as complete!

RZ/pc

ATARI®

inter-office memo

TO: ALL PAM PROGRAMMERS

FROM: Rob Zdybel DATE: March 18, 1982

SUBJECT: "FINAL" PAM O.S.

The final version of the PAM O.S. is now available. Please test your cart with this latest version and report any bugs to me IMMEDIATELY! The most recent changes shouldn't affect anyone's code but I'm not taking any chances.

Remember that the DLOS command will always download the most recent version of the O.S.

Finally, the O.S. is exactly as described in the memo 'PAM O.S. Partial Description' with the exception that O.S. zero-page usage is bytes 0 thru 18 (hex),inclusive. That is, you are free to use any zero-page memory from 19 (hex) on up. More zero-page may be recoverable if you aren't using all of the O.S. capability, see the above-mentioned memo.

RZ/db

Inter Office Memo

ATARI®
Consumer Electronics Division

To: ALL PAM PROGRAMMERS

From: ROB ZDYBEL

Subject: SELF-CENTERING POTS   Date: 4-14-82

There was a bug in the Self-Centering-Pot-Algorithm as described in the 12/21/81 memo, "PAM Controllers II" (corrected copies are available from Denese).

In that algorithm MAX, MIN & MID were initialized to 80(hex). This failed when the stick was moved only in one direction, causing MID to follow the endpoint which was being moved (either MAX or MIN), thus making it impossible to achieve a full-left or full-right position. The bug would disappear when the stick was extended to the opposite extreme, causing MID to take on a value more truly representative of the mid-range value of the pot.

In general, it is not wise to initialize your self-centering pot routine to a too limited range. The pots are guaranteed a minimum range of 160 counts on any POKEY and you are not required to make a broken controller work! The desire to make marginal controllers function effectively is appreciated, but let us not do it at the expense of all other players whose controller functions properly.

Given that hardware is shooting for a mid-range value of 70(hex) for all pots, I would recommend initializing MID to 70(hex), MAX to B0(hex) and MIN to 30(hex). All these values should fall well within minimum tolerances for all pots and yet avoid the problems exhibited by specifying a too narrow starting range.

The RESET key has been accidently pushed while using the the joystick in the heat of play. You may wish to guard against this irritation by inhibiting RESET while game-play is active.

To make it more difficult to trash a game, but still possible to escape from an unwanted contest. Two possible approaches are:

1) require that RESET be held-down for a fraction of a second
2) require RESET be pressed twice