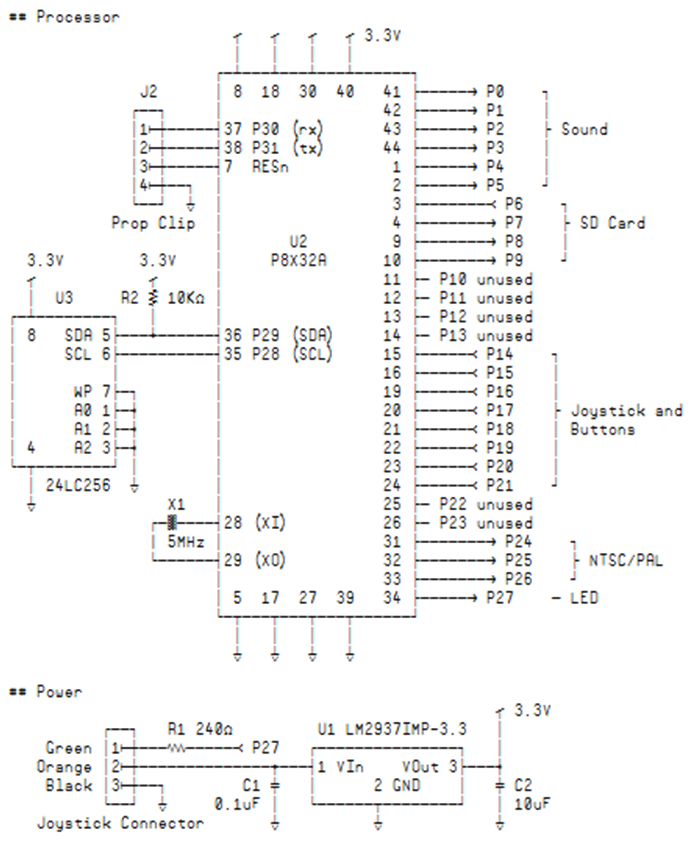
**PEGS (Propeller Empowered Game System)**

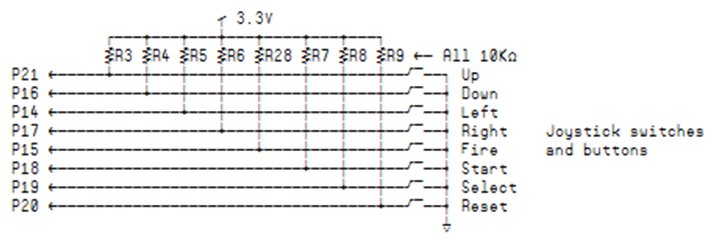
**Project Number** <TOPHER find this>

**Project Description**

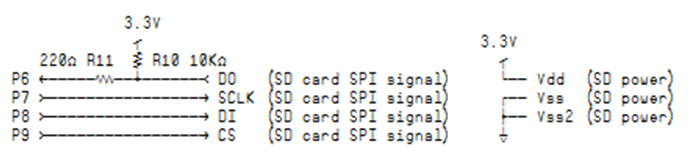
**Block Diagram**



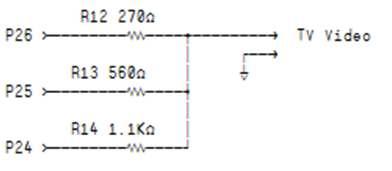
Processor and Power (Boot.spin)



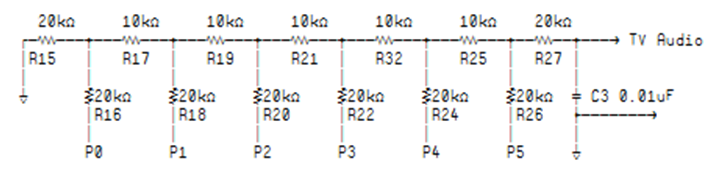
Joystick Switches/Buttons (Interpreter.spin)



SD Card Interface (DiskCOG.spin)



TV Video Interface (TV8x8.spin)



TV Audio Interface (SoundCOG.spin)

**Block Diagram**

>< Message box exchange. The VideoCOG commands can read/write variables.

>< Dual sprite generators increase the number of sprites on a single line

>< COGs can be loaded from disk. A different video engine. Use only one sprite engine and load a second interpreter to multitask game logic. Code and data are encapsulated. Variables in the VariableCOG. Waveforms in SoundCOG. Program stack in the Interpreter. Isolates and protects. Modular.

Interpreter

VariableCOG

DiskCOG

VideoCOG

SoundCOG

TV8x8

Sprite

Sprite

**Source Code**

>< The COGTALK function is used by all the COGs to talk to one-another. Command structure (long and short). One-way command support.

>< Flow commands drift in and out of clusters. Keep stack of return cluster/offset for CALL/RETURN. BRANCH-IF.

>< COLORMIX MIX and compiled. Conversion of C if/loop structures. Written in binary or SPIN as here for default-cluster in BOOT.SPIN. The ASCII art in the source.

***Functions COGTALK/COGTALKWAIT from INTERPRETER.SPIN***

' -----------------------------------------------------------------------------

' This function sends a command to the requested cog.

' box - mailbox number to send to

' cStat - command to send

' data1-data5 - data words

' ofs - offset value

' Returns (next function waits for the reply)

' com - return status

' data1 - return value

'

cogTalk mov ptr,box ' 32 bytes ...

shl ptr,#5 ' ... per box

add ptr,baseBox ' Point to target box

add ptr,#4 ' Write command lastly

cogTalk1 lockset box wc ' Wait for lock ...

if\_nc jmp #cogTalk1 ' ... on the mailbox

wrlong data1,ptr ' Write data1

add ptr,#4 '

wrlong data2,ptr ' Write data2

add ptr,#4 '

wrlong data3,ptr ' Write data3

add ptr,#4 '

wrlong data4,ptr ' Write data4

add ptr,#4 '

wrlong data5,ptr ' Write data5

add ptr,#4 '

wrlong data6,ptr ' Write data6

add ptr,#4 '

wrlong ofs,ptr ' Write the offset

sub ptr,#7\*4 ' Now write the ...

wrlong cStat,ptr ' ... command value

cogTalk\_ret

ret

'

cogTalkWait

rdlong com,ptr ' Wait for ...

shl com,#1 nr, wc ' ... status bit to ...

if\_c jmp #cogTalkWait ' clear out

add ptr,#4 ' Read the ...

rdlong data1,ptr ' ... return value

lockclr box ' Release our lock

cogTalkWait\_ret

ret

***Flow-control interpreter commands from INTERPRETER.SPIN***

'' BRANCH-IF c:o

'' Change program counter to offset o within cluster c if last COG.

'' command (usually a VariableCOG command) was non-zero.

'' 0\_000\_011\_cccccccccccccccc\_ooooooooo

ifCommand

cmp lastCOGRet,#0 wz

if\_nz jmp #gotoCommand

jmp #mainLoop ifnotCommand

cmp lastCOGRet,#0 wz

if\_nz jmp #mainLoop

''

'' GOTO c:o

'' Change program counter to offset o within cluster c.

'' 0\_000\_000\_cccccccccccccccc\_ooooooooo

gotoCommand

mov pc,com ' Set offset in current cluster

cmp tmp,C\_FFFF wz ' If requesting another clsuter ...

if\_nz call #changeCluster ' ... change to requested cluster

jmp #mainLoop ' Next command

''

'' CALL c:o

'' Change program counter to offset o within cluster c.

'' The return cluster/offset is pushed onto call stack.

'' 0\_000\_001\_cccccccccccccccc\_ooooooooo

callCommand

mov tmp2,stackPtr ' Point to next slot in ...

add tmp2,#stack ' ... COG stack memory

movd sp1,tmp2 ' Use this pointer later

add stackPtr,#1 ' Bump the stack pointer

mov tmp2,clusterNumber ' Clurent cluster number ...

shl tmp2,#9 ' ... shifted to top of long

mov t1,pc ' Program counter ...

shr t1,#2 ' ... must be long-aligned (save bits)

or tmp2,t1 ' Combine cluster and offset

sp1 mov 0,tmp2 ' Save return cluster/offset on stack

jmp #gotoCommand ' A regular GOTO from here

''

'' RETURN

'' Pop the cluster/offset from the call stack.

'' 0\_001\_0000000000000000000000000000

returnCommand

sub stackPtr,#1 ' Decrement stack pointer

mov tmp2,stackPtr ' Point to last ...

add tmp2,#stack ' ... slot on stack

movs sp2,tmp2 ' Store the pointer

nop ' Stall before using pointer

sp2 mov tmp,0 ' Pull the return cluster/offset

mov com,tmp ' Get the ...

shr tmp,#9 ' ... return cluster

and com,#$1FF ' Get the ...

shl com,#2 ' ... return offset

jmp #gotoCommand ' A regular GOTO from here

'

changeCluster

cmp tmp,clusterNumber wz ' If the requested cluster is ...

if\_z jmp changeCluster\_ret ' ... current cluster, ignore request

mov box,#0 ' The DiskCOG's mailbox

mov cStat,C\_LOAD\_CLUSTER ' LOAD command

or cStat,tmp ' Put requested cluster in command value

mov ofs,baseCluster ' Our current cluster (releasing)

call #cogTalk ' Fetch the cluster

mov clusterNumber,tmp ' Current cluster is now loaded

mov baseCluster,data1 ' New memory offset to loaded page

changeCluster\_ret

ret

***Snippet of the COLORPICK.MIX program written in the MIX language.***

/\*

This program shows all the possible color values that can be produced by the graphics system.

There are 3 pages of colors with the hex-values for each shown below each. Press enter to

flip among the pages.

\*/

// General purpose variables

variable x,y,z,c,v,ptr,cs

// Variables used to draw rectangles

variable rx,ry,rw,rh,rt

// Variables used to set the cursor coordinates

variable cx, cy

// 4 solid tiles ... one for each color of a 4-color set

INITTILES solids, 10,4

// Background color a soft blue

mem(M\_ColorScheme\_0) = 0x3B

while(true) {

cs = 2 // 0 and 1 are NOT valid colors

call drawPics // Draw first page

call getNextColor // Next valid color

call drawPics // Draw second page

call getNextColor // Next valid color

call drawPics // Draw third page

}

/\*\*

\* This fuction draws a single page of color values. CS contains the starting

\* color value

\*/

drawPics:

cls

**... code deleted ...**

solids:

Tile {

........ RRRRRRRR GGGGGGGG WWWWWWWW

........ RRRRRRRR GGGGGGGG WWWWWWWW

........ RRRRRRRR GGGGGGGG WWWWWWWW

........ RRRRRRRR GGGGGGGG WWWWWWWW

........ RRRRRRRR GGGGGGGG WWWWWWWW

........ RRRRRRRR GGGGGGGG WWWWWWWW

........ RRRRRRRR GGGGGGGG WWWWWWWW

........ RRRRRRRR GGGGGGGG WWWWWWWW

}

***The COLORPICK.MIX program compiled into binary (SPIN format)***

' Cluster ''

' INITTILES solids, 10,4

long %1\_111\_0010\_\_\_00\_0\_01000\_\_\_0000001001011000

long %0000000000001010\_\_0000000000000100

' mem(M\_ColorScheme\_0) = 0x3B

long %1\_111\_0001\_\_01\_000\_011\_10111010\_0000\_1011

long %00000000\_000000000000\_000000111011

long %1000000000111111101110000

' \_if\_1\_1:

' \_if\_1\_expression:

' \_loop\_1\_start:

' \_loop\_1\_continue:

' true

long %1\_111\_0001\_\_00\_000\_111\_00101010\_0111\_1011

long %00000000\_000000000000\_000000000001

' BRANCH-IFNOT \_if\_1\_false

long %0\_000\_010\_1111111111111111\_000010000

' \_if\_1\_true:

' cs = 2

long %1\_111\_0001\_\_00\_000\_111\_10001010\_0000\_1011

long %00000110\_000000000000\_000000000010

' call drawPics

long %0\_000\_001\_1111111111111111\_000010000

' call getNextColor

long %0\_000\_001\_1111111111111111\_001101001

' call drawPics

long %0\_000\_001\_1111111111111111\_000010000

' call getNextColor

long %0\_000\_001\_1111111111111111\_001101001

' call drawPics

long %0\_000\_001\_1111111111111111\_000010000

' GOTO \_loop\_1\_start

long %0\_000\_000\_1111111111111111\_000000101

' drawPics:

' \_loop\_1\_end:

' \_if\_1\_end:

' \_if\_1\_false:

' cls

long %1\_111\_0010\_\_\_00\_0\_00000\_\_\_00000000\_\_\_00000000

long %00100000\_00011010\_0000000000100000

**... code deleted ...**

' solids:

' Tile {

byte $0, $0, $0, $0, $0, $0, $0, $0, $0, $0, $0, $0, $0, $0, $0, $0

byte $55, $55, $55, $55, $55, $55, $55, $55, $55, $55, $55, $55, $55, $55, $55, $55,

byte $aa, $aa, $aa, $aa, $aa, $aa, $aa, $aa, $aa, $aa, $aa, $aa, $aa, $aa, $aa, $aa,

byte $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff

**Bill of Materials**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **Circuit Label** | **Digi-Key or Parallax** | **Value** | **Mounting** | | U1 | LM2937IMP-3.3CT-ND |  | 4 pin SOT-223 | | U2 | P8X32A-Q44 |  | 44 pin QFP | | U3 | 24LC256T-I/SNCT-ND |  | 8 pin SOIC | | X1 | X1050-ND |  | 2 pin HC49U | | C1 | PCC1762CT-ND | 0.1uF | SMT-0603 | | C2 | PCC2367CT-ND | 10uF | SMT-0603 | | C3 | PCC1764CT-ND | 0.012uF | SMT-0603 | | R1, R11 | P240GCT-ND | 240 | SMT-0603 | | R2, R3, R4, R5, R6, R7, R8, R9, R10, R17,R19, R21, R23, R25 | P10KGCT-ND | 10K | SMT-0603 | | R12 | P270GCT-ND | 270 | SMT-0603 | | R13 | P560GCT-ND | 560 | SMT-0603 | | R14 | P1.1KGTR-ND | 1.1K | SMT-0603 | | R16, R18, R20, R22, R24, R26, R27, R28 | P20KGCT-ND | 20K | SMT-0603 | |  |  |  |  |

**Pictures**

<TOPHER Me and project>

<TOPHER Project inside 1>

<TOPHER Project inside 2>

<TOPHER Movie playing sea war>